ҚАЗАҚСТАН РЕСПУБЛИКАСЫ
БІЛІМ ЖӘНЕ ҒЫЛЫМ МИНИСТРЛІГІ

СОММЕРВИЛЬ

БАҒДАРЛАМАЛЫҚ ЖАСАҚТАМА

1

Алматы, 2013

ҚАЗАҚСТАН РЕСПУБЛИКАСЫ БІЛІМ ЖӘНЕ ҒЫЛЫМ МИНИСТРЛІГІ

ИАН СОММЕРВИЛЬ

БАҒДАРЛАМАЛЫҚ ЖАСАҚТАМА

1

Оқулық

Алматы, 2013

ӘОЖ 004.4 (075)
КБЖ 32.973-018 я 73
С 65

Қазақстан Республикасы Білім және ғылым министрлігінің «Оқулық» республикалық ғылыми-практикалық орталығы бекіткен

Қазақ тіліне аударған
Оразбеков С., Alfa Translation аударма агенттігі

Соммервиль И.
С 65 **Бағдарламалық жасақтама. 1-бөлім:** Оқулық / Ауд. С. Оразбеков, Alfa Translation. –. Алматы, 2013.

ISBN 978-601-7427-35-1
1-бөлім. – 460 б.
ISBN 978-601-7427-36-8

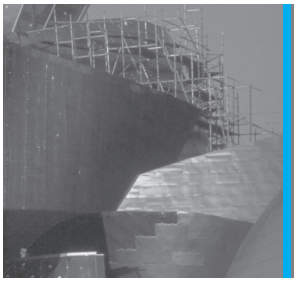
Оқулықтың авторы – мол бағдарламалық тәжірибесі бар, Шотландиядағы St. Andrews Университетінің профессоры Иан Соммервиль. Оқулық 26 тараудан тұрады. Оқулық өте көлемді болғандықтан, бөлімдер тақырыптарына сай кітаптың негізгі 4 тарауында біріктірілген. Оқулық қазіргі заманғы деңгейде жазылған, бағдарламалық жасақтама үдерістерін суреттейтін әртүрлі кестелерден тұрады. Оқулықтың ерекшелігі – әр тақырыпқа сай, теориялық білімді іс жүзінде қолданатын шынайы өмірден алынған мысалдары. Бұл мысалдар бағдарламалық өндірістен алынғандықтан, автор оқырманға бағдарламалық мәселені толығымен суреттейді де, сол мәселенің іс жүзінде қолданылған шешу жолдарын көрсетеді.

Қазақша басылымда оқулықтың 2 бөлімі, яғни алғашқы 15 тарауы шығады. Бұл тараулардан қазіргі замандағы бағдарламалық жасақтама үдерістері, үдерістердің қауіпсіздігі, бағдарламаны әзірлеудің икемді әдістері және әзірленген бағдарламаны сынау тәсілдері жөнінде толық мәлімет алуға болады.

Оқулық бағдарламалық жасақтамаға алғаш қадамдарын жасаған колледж және университет студенттеріне арналған. Алайда, өндірістегі бағдаламалаушыларға да жұмыс барысын жақсартатын көптеген пайдалы мәлімет табуға болады. Оқулық оқырмандары кемінде бір бағдарламалау тілімен және жалпы бағдарламалау терминологиясымен таныс болулары тиіс.

ӘОЖ 004.4 (075)
КБЖ 32.973-018 я 73

ISBN 978-601-7427-36-8 (1-б.)
ISBN 978-601-7427-35-1 (орт.)



КІРІСПЕ

Бағдарламалық жасақтама дайындау (software engineering) ісіне 40 жыл толғанын осы оқулықтың соңғы тарауларын жазу үстінде, 2009 жылдың жазында байқадым. “Бағдарламалық жасақтама” (“software engineering”) термині 1969 жылы САКҰ конференциясында ұсынылған, конференцияда бағдарламалық жасақтаманың дамуына байланысты: көлемді бағдарламалық жасақтамалардың тым кеш дамуы, пайдаланушы қажетті функционалдылықпен қамтамасыз етілмеуі, бағасының жоғары болуы, сенімсіздік тудыратын факторлар және тағы басқа мәселелер көтерілді. Мен конференцияға қатыса алмасам да, бір жылдан соң ең бірінші бағдарламамды жазып, бағдарламалық жасақтамада кәсіби өмірімді бастадым.

Кәсіби өмірімде бағдарламалық жасақтама жетістіктері белгілі орын алды. Біздің қоғам көлемді кәсіби бағдарламалық жасақтамасыз жұмыс істей алмайды. Бизнес жүйесін құру үшін, көлемді корпоративтік бағдарламаны дамыту мақсатында: J2EE, .NET, SaaS, SAP, BPEL4WS, SOAP, CBSE және т.б. технологиялар қолданылады. Халықтық пайдаланулар мен инфақұрылымдар – энергия, байланыс және транспорт барлығы да күрделі де сенімді компьютерлік жүйеге байланысты құрылған. Бағдарламалық жасақтама – адам тарихындағы ең маңызды ақпараттық жүйе. Дүниежүзілік торды (World Wide Web) құруға және кеңістікті зерттеуге зор мүмкіншілік тудырды.

Қазіргі таңда адамзат баласына ауа райының күрт өзгеруі, табиғи ресурстардың азаюы, әлем бойынша тұрғындардың көбеюі, халықтық терроризм, егде жастағы адамдарға көмек беру қажеттілігі жаңа қиындықтар туғызып отыр. Осындай қиындықтарды шешу мақсатында жаңа технологиялар қажет, ең маңызды рөлді бағдарламалық жасақтама атқарады.

Сондықтан бағдарламалық жасақтама болашақ адамзат болмысы үшін ең маңызды технология болып табылады. Күрделі компьютерлік жүйені құру мақсатында және осы бағытты дамыту үшін білікті мамандарды дайындауды жалғастыру керек. Бағдарламалық жасақтама жобасында әлі де қиындықтар туындайды. Қазіргі уақытта кейбір кейінгі пайда болған жасақтамалардың қымбат бағалануы жиі кездеседі. Бірақ біз осы мәселеге бой алдырмай, бағдарламалық жасақтаманың негізгі жетістіктерін дұрыс бағалай білуіміз керек.

Бағдарламалық жасақтама қазіргі таңда өте үлкен сала болғандықтан, барлық бағыттарын бір оқулыққа сыйғызу мүмкін емес. Сондықтан да мен барлық бағдарламалық жасақтамалардың дамуына негіз болатындай түйінді тақырыптарға көңіл бөлдім. Бұл жерде шапшаң амалдар мен жасақтаманы қайта пайдалану көңілге алынды. Шапшаң амалдармен қатар, дәстүрлі жасақтаманың

да орны бөлек. Бағдарламалық жасақтаманы жақсарту үшін осы екеуінің жақсы бағыттарын біріктіруіміз қажет.

Оқулық көбінесе, автордың өз пікірі мен ойлары бойынша жазылады. Кейбір оқырмандар менің пікіріммен және таңдаған материалмен келіспеуі мүмкін. Ондай келіспеушілік бұл күрделі бағыт үшін дұрыс әрекет болып табылады, осы эволюция үшін орынды. Алайда, барлық бағдарламалық жасақтама мамандары және жасақтама студенттері үшін қажетті мағлұмат табылатынына сенемін.

Вебпен бірігуі

Интернет жүйесінде бағдарламалық жасақтамаға қатысты өте көп мәлімет бар, кейбіреулер үшін осындай оқулықтың қажеттілігі жоқ. Бірақ кейбір ақпаратты табу қиыншылықтар туғызады, қажетті ресурстардың орнына өзге ақпараттың табылуы немесе материалдар төменгі сапалы болады. Сондықтан оқу үдерісінде оқулықтың маңызды рөл атқаратынына сенемін. Оқулық кейде карта рөлін де атқарады, ақпаратты жеткізу әдісін жеңілдетіп, оқуға тиімді етеді. Сол сияқты интернеттегі материалды терең оқып білуге бастама болып табылады.

Оқулықты интегралдау мүмкіндігі болса және интернеттегі материалдарға жаңа зат ашса ғана ол оқулықтың болашағы бар деп айтуға болады. Сондықтан бұл оқулық веб-текст және оқулық ретінде жазылған, оқулықтағы негізгі материал интернеттегі материалмен тығыз байланысты. Барлық тараулардың сол материалға қосымша ақпарат беретін веб-бөлігі, сол сияқты тақырыпта қамтылмаған 4 веб-тараулар бар.

Оқулыққа қатысты сайт:

http://www.SoftwareEngineering-9.com

Веб-оқулықтың төрт негізгі компоненттері:

1. *Веб-бөлігі.* Бұл секцияда оқулыққа сілтеме болатын қосымша материалдар орналасқан. Бұл веб-бөліктер оқулықтағы әр тараудың коммутациялы кестесімен байланысты.
2. *Веб-тараулар.* Формалды әдіс, әрекеттесу әрленімі, құжаттама және бағдарламалық архитектураға арналған төрт веб тарау бар. Оқулыққа басқа да жаңа тараулар қосуым мүмкін.
3. *Оқытушыларға арналған материалдар.* Бұл секцияда материалдар бағдарламалық жасақтаманы үйретететін оқытушыларға арналған. Кіріспеден «Қосымша материалдар» бөлігін қараңыз.
4. *Тақырыптық зерттеулер.* Оқулыққа байланысты тақырыптарға қосымша информация беретін бөлік. Мысалы, Ариан 5 іске қосқыш құрылғысының сәтсіздікке ұшырауы туралы тақырыпта.

Осы секциялармен қатар басқа да бағдарламалық жасақтамаға арналған веб-сайттар мен блогтарға сілтемелер көрсетілген.

Мен осы оқулыққа және веб-сайтқа байланысты пікірлеріңіз бен кеңестеріңізді күтемін. Менімен: ian@SoftwareEngineering-9.com мекенжайы арқылы байланыса аласыз. Хатта хабарламаңыздың атын [SE9] деп белгілеңіз. Әйтпесе, спам фильтрі хатты жойып жіберіп, сіз жауап ала алмауыңыз мүмкін.

Оқырмандарға

Бұл оқулық ең біріншіден бағдарламалық жасақтамада бастама және ілгері курстар оқитын университет және колледж студенттеріне арналған. Өнеркәсіпте бағдарламалық жасақтама мамандары үшін:

- бағдарамалық жасақтаманың қайтадан пайдалануы
- архитектуралық әрленім
- тәуелділік және қауіпсіздік
- үдерісті жақсарту

сияқты тақырыптары білімдерін нығайтуға және жалпы оқуға да пайдалы болады. Бұл оқулық оқырман бағдарламалау терминологиясы және бағдарламалау бастамасын біледі деп жобаланды.

Алдыңғы баспалардағы өзгерістер

Бұл баспада алдыңғы шығарулардың бағдарламалық жасақтамаға арналған негізгі материалдары сақталғанымен, мен тарауларды қайталап жаңаладым және басқа тақырыптарға жаңа материал қостым. Ең маңызды өзгертулер:

1. Басылым оқулықта орналасқан материалдардағы веб-бөліктер, басылым оқулықтан веб-оқулыққа аударылуы. Осының арқасында оқулықтағы тараулардың саны азайып, әр тараудағы маңызды тақырыптар қамтылды.
2. Оқулық жеңіл оқылуы үшін оқулықтың құрылымын өзгерту аяқталды. Қазір оқулықта 8 тараудың орнына 4 тарау бар және әр тақырып бір-бірімен байланысқан, басқа материалдармен де қолданылады. Төрт тарау бағдарламалық жасақтамаға, тәуелділік және қауіпсіздік, ілгері бағдарламалық жасақтама үшін, оны басқаруға бастама ретінде қолданылады.
3. Алдыңғы басылымдағы кейбір тақырыптар бір тарауда веб-ресурстарға сілтемесімен қысқаша түрде көрсетілген.
4. Осы оқулыққа қосылмаған алдыңғы басылымдағы қосымша веб-тараулар интернетте бар.
5. Барлық тарауларды қайталап, мазмұндарын жаңаладым.
6. Дамыған жасақтамалар үшін жаңа тараулар қостым.

7. Осы жаңа тарауларда жасақтама басқарылатын үлгілер, ашылған бастапқы кодты әзірлеу, жасақтаманы дамыту, Reason Swiss Cheese моделі, тәуелді архитектуралық жүйелер, тұрақты зерттеулер және үлгіні тексеру, COTS қайта пайдалану, жасақтаманы қызмет ету ретінде қолдану және жылдам жоспарлау тақырыптары жаңа материалдармен толықтырылды.
8. Кейбір тарауларда психикалық проблемасы бар аурулар үшін жаңа зерттеулер қолданылған.

Оқулықтың білім беру мақсатында қолданылуы

Мен оқулықты бағдарламалық жасақтаманың 3 түріне арнадым:

1. Бағдарламалық жасақтамаға жалпы бастама курсы. Оқулықтың бірінші бөлігі бағдарламалық жасақтаманың бастамасы ретінде 1 семестрге арналып жасалған.
2. Спецификалық бағдарламалық жасақтама үшін бастама немесе ортаңғы курстары. *2-,3-* және *4-тарауларды* пайдаланып, ілгері курстар құруға мүмкіншілігіңіз бар. Мысалы, мен *2-тарауды* қолдана отырып, сындық жасақтама жүйесі курсымен қоса сапа менеджменті мен құрылым менеджментін оқыдым.
3. Арнайы бағдарламалық жасақтамаға арналған ілгері курстар. Бұл курсқа осы оқулықтағы тараулар негіз болады. Тақырып оқу барысында толықтырылды. Мысалы, жасақтаманы қайта қолдану *16-,17-,18-* және *19-тарауларда* кездеседі.

Оқулықты оқыту мақсатында қолдану үшін арналған информация веб-сайтта орналасқан.

Қосымша материалдар

Қосымша материалдардың түрлілігі осы “Бағдарламалық жасақтама” кітабын оқытуға арналған. Мысалы:

- Осы оқулықтағы барлық тарауларға арналған Power Point презентациялары;
- Power Point кескіндері;
- Әртүрлі курстар үшін оқулықты қалай пайдалану нұсқаулары мен алдыңғы баспалармен байланысы туралы оқытушыға арналған нұсқау;
- Оқулықтағы тараулар туралы ақпарат;
- Бағдарламалық жасақтамада қолданылатын қосымша тақырыптық зерттеулер;
- Бағдарламалық жасақтамаға арналған қосымша PowerPoint презентациялары;

- Формальды әдіс, әрекет әрленімі, бағдарлама архитектурасы және құжаттама тақырыптары сияқты материалдар алдында айтып кеткен 4 веб-тарауларда қамтылған.

Осы материалдар оқулық веб-сайтында немесе төменде көрсетілген сілтемедегі Pearson веб сайтында бар және тегін. Бірақ:

- Кейбір тараудың соңындағы жаттығулардың жауаптары.
- Әр тарауға арналған сұрақтар мен жаттығулардың жауаптары сияқты мәлімет тек тіркелген оқытушыларға жеткілікті.

Барлық қосымша материалдар төменгі мекенжайдан алынған:

http://www.pearsonhighered.com/sommerville

Оқытушылар оқулықты білім беру мақсатында қолдана отырып, шектелген материалдарды қолдануға мүмкіндігі бар. Ол үшін Pearson веб сайтында Pearson өкілімен хабарласып немесе computing@aw.com электронды поштасына жазу керек. Автор құпия сөзбен қамтамасыз етпейді.

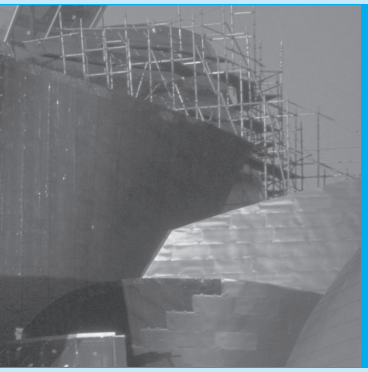
Алғыс

Осы оқулықтың шығуына себеп болған көптеген адамдарға алғысымды білдіремін, сол сияқты алдыңғы баспаларда өз пікірлерімен, кеңестерімен бөліскен адамдарға (студенттерге, кітап оқушыларға) алғыс айтамын.

Оқулықты жазу барысында көмектескен өз отбасыма (Анна, Али, Джейн) алғыс білдіремін. Ерекше алғысымды корректурада және түзетуде ерекше талантын көрсеткен өз қызым Джейнге айтамын. Ол оқулықты түгел оқып, көптеген қателерді және қате басылуларды табуға көмектесті.

Иан Соммервиль
2009 жыл, қазан

МАЗМҰНЫ



1 БӨЛІМ Бағдарламалық жасақтамаға кіріспе

Бағдарламалық жасақтамаға жалпы кіріспе жасау – оқулықтың осы бөлігіндегі көздеген мақсатым болып табылады. Бастапқы бағдарламалық қамтыма тізімінен бастап, жүйелік эволюцияға дейінгі бағдарламалық қамтыма үдерістері және икемді тәсілдер сияқты маңызды тұжырымдамалар енгіздім. Осы бөлімдегі тараулар бір семестрлік Бағдарламалық жасақтама курсына арналып жасалған.

1-тарау – кейбір бағдарламалық қамтыма тұжырымдамаларын анықтаумен қатар, кәсіби бағдарламалық инженерлік іске жалпы кіріспе болып табылады. Сонымен қатар мен бағдарламалық инженерлік іске қатысты этикалық мәселелер жайлы қысқаша пікірсайыс жаздым. Менің ойымша, бағдарламалық инженерлер үшін жұмыстарының нәтижесін жақсартуды ойлағаны маңызды болып табылады. Сол сияқты осы тарауда оқулықта кездесетін үш тақырыптық зерттеулер енгізілді, олар: психикалық денсаулығына байланысты ем алатын аурулар үшін аурулардың жазбаларын басқаратын жүйе, қолайлы инсулиндік помпаны басқару жүйесі және шөлді жердегі ауа райы жүйесі.

2- және 3-тарауларда бағдарламалық қамтыма үдерістері мен икемді даму тәсілдері қамтылған. *2-тарауда* сарқырама үлгісі сияқты, жалпы қолданылатын бағдарламалық қамтыма

үдерістерінің негізгі үлгілерін енгіздім және сол үдерістердің маңызды бөлігін құрайтын негізгі әрекеттерді талқыладым. *3-тарау* бағдараламалық қамтыманың икемді тәсілдерімен толықтырылған. Мен көбінесе, икемді тәсілдің мысалы ретінде экстремалды бағдарламалауды пайдаландым.

Осы бөлімдегі тараулардың қалған бөлімінде *2-тарауға* кіретін, бағдарламалық қамтыма үдерістері тереңірек баяндалды. *4-тарауда* жүйе қызметіне байланысты анықталған талаптарды, инженерлік іс талаптары тақырыбын қамтиды. *5-тарау* UML-ді қолдана отырып, жүйелік модельдеу тақырыптарын қамтиды, ол жерде мен бағдарламалық қамтыма жүйесін модельдеу үшін қолданудың нұсқасы диаграммасына, класс диаграммасына, дәйектілік және жай-күй диаграммаларына аса көңіл бөлдім. *6-тарау* – архитектуралы жобалауға кіріспе және архитектураның және бағдарламалық қамтыма жобалауында қолданылатын архитектуралық шаблондар жайлы баяндама.

7-тарау – нысанға бағытталған жобалауға және жобалау шаблондарын қолдануға кіріспе. Сол сияқты бағдарламалық қамтыманы қайта пайдалану, конфигурацияны басқару және ашық мәліметтерді өңдеу сияқты іске асыру мәселелерін енгіздім. *8-тарау* жүйені өңдеу барысындағы модульдік тестілеуден бағдарламалық қамтыма нұсқаларын тестілеуге дейінгі бағдарламалық тестілеуге арналған. Сонымен қатар, осы тарауда тестілеу арқылы өңдеуді талқыладым – бұл, ең алдымен, икемді тәсілде кеңінен қолданылған жақындау болып табылады. *9-тарау* – бағдарламалық қамтыма эволюциясына шолу. Мен эволюциялық үдерістер, бағдарламалық қызмет ету, мұралық жүйені басқару тақырыптарын шолдым.

1.
Кіріспе

Мақсаттары

Осы тараудың мақсаты – бағдарламалық қамтымаға кіріспе жасап, оқулықтың қалған бөлігін түсінуге бағыттау. Осы тарауды оқып болған соң, сіз:

- бағдарламалық қамтыманың не екенін және неліктен маңызды екенін түсіне аласыз;
- бағдарламаның түріне байланысты бағдарламалық жасақтама техникасының әртүрлі болу мүмкіндігін түсіне аласыз;
- бағдарламалық инженерлер үшін маңызды этикалық және кәсіби мәселелермен таныса аласыз;
- оқулыққа мысал ретінді енгізілген үш түрлі жүйемен танысасыз.

Мазмұны

Қазіргі таңда әлемді бағдарламалық қамтымасыз елестету мүмкін емес. Ұлттық инфрақұрылым және үй-жай тұрмыс шаралары да компьютерлік жүйенің қол астында, көптеген электротехникалық өнімдер компьютермен және бағдарламалық қамтыманы басқарумен тығыз байланысты. Өнеркәсіптік өндіріс және үлестіру де қаржы жүйесі секілді толығымен компьютерленген. Ойын-сауық түрлерінің де, соның ішінде: музыка индустриясы, компьютерлік ойындар, кино және телевизия да бағдарламалық қамтымамен тығыз байланысты. Сондықтан бағдарламалық қамтыманың ұлттық және халықаралық қоғамда маңызы зор.

Бағдарламалық қамтыма жүйелері – абстрактілі және бейматериалды. Олар физикалық құбылыстарға бағынатын материалдардың қасиеттерімен және өндірістік үдерістермен шектелмеген. Бұл бағдарламалық қамтыманы жеңілдетеді, өйткені бағдарламалық қамтыманың потенциалына ешқандай табиғи шектеу жоқ. Бірақ физикалық шектеулердің болмағанымен, бағдарламалық қамтыма жүйелері тез арада түсінуге қиындап, бағасы да қымбаттап кетуі мүмкін. Ақпараттық жүйе әлемінде қарапайым жүйеден күрделіге дейін көптеген түрлі бағдарламалық жүйелер бар. Бағдарламалық қамтыманы өңдеу үшін әртүрлі белгілерді, тәсілдерді немесе технологияларды іздеудің мәнісі жоқ, өйткені бағдарламалық қамтыманың түрлілігіне байланысты әрқайсысына бөлек әдіс қолданылады. Ұйымдастырылған ақпараттық жүйені өңдеудің және басқарғыш ғылыми құралды өңдеудің айырмашылығы зор. Бұл жүйелердің ешқайсысының да графикалық компьютерлік ойындарға еш қатысы жоқ. Барлық бұл қолданбалар бағдарламалық қамтыманы қажет етеді, бірақ оларға бағдарламалық қамтыманың әртүрлі әдістері қажет. Әлі де бағдарламалық қамтыма жобаларының қателіктері бар. Бағдарламалық жасақтама заманауи бағдарламалық қамтыманың дамуына сәйкес емес деп сыналған. Бірақ менің ойымша, осы аталған бағдарламалық қателіктерді туғызатын мынандай екі фактор бар:

1. *Сұраныстың өсуі.* Жаңа бағдарламалық қамтыма әдістерінің неғұрлым үлкен және күрделі жүйелерді құрастыруына байланысты сұраныс та өседі. Жүйе құрастырылып, тез арада жеткізілу керек; көлемі өзгерген сайын, күрделі жүйелерге сұраныс артады; жүйелер ертеде мүмкін емес болған жаңа мүмкіндіктерге ие болу керек. Бағдарламалық қамтыманың қазіргі қолданылатын әдістері тиімсіз, сондықтан жаңа бағдарламалық қамтыма әдістері осы сұраныстарға сәйкес құралуы тиіс.

2. *Шағын күту.* Бағдарламалық қамтыма тәсілдері мен әдістерін қолданбай компьютерлік бағдарламаларды құрастыру оңай. Көптеген компаниялар өнімдері мен қызметтерінің дамуына байланысты бірден бағдарламалық қамтыма өңдеу ісіне кірісіп кетті. Олар күнделікті жұмыста бағдарламалық қамтыма әдістерін қолданбайды. Сондықтан олардың өнімі қымбат және сенімсіз. Осындай мәселені шешу үшін біз бағдарламалық білімді және дайындықты жақсартуымыз қажет.

Бағдарламалық жасақтамаларды жобалау тарихы

«Бағдарламалық жасақтама» ұғымы «Бағдарламалық дағдарыс» атауы мәнін талқылау мақсатында өткізілген конференцияда алғаш рет 1968 жылы ұсынылды. (Наур және Рандэль, 1969). Бағдарламаларды жетілдіруге жеке тәсілдердің ірі және кешенді бағдарламалық жасақтамалар жүйесін кеңейте алмайтындығы айқындала бастады. Бұл сенімсіз, жоспарлағаннан әлдеқайда қымбат және уақытынан кеш жеткізілді.

1970 және 1980 жылдар айналасында модульдік бағдарламалау, ақпаратты жасыру және нысанға бағытталған жобалау сияқты бірнеше жаңа бағдарламалық жасақтамаларды жобалау техникалары мен әдістері дамыды. Құралдар мен стандартты шартты белгілер жиынтығы да жасақталды және олар қазіргі таңда кеңінен қолданылуда.

http://www.SoftwareEngineering-9.com/Web/History/

Бағдарламалық қамтыма қызметкерлері заңды түрде өз жетістіктерін мақтан тұта алады. Әлі де бағдарламалық қамтыманың дамуында қателіктер болғанымен, бағдарламалық қамтымасыз біз кеңістікті тану, интернетті қолдану немесе заманауи байланыс жүйелерін қолдану мүмкіншілітеріне ие болмас едік. Саяхаттың қандай да болмасын түрі қауіпті және шығынды болар еді. Бағдарламалық қамтыма үлкен жұмыс жасады және мен осы жұмыстың XXI ғасырда болашағы бар екеніне сенемін.

1.1. Бағдарламалық қамтыманы кәсіби өңдеу

Көптеген адамдар бағдарламалау ісімен айналысады. Бизнеспен айналысатындар адамдар жұмыстарын жеңілдету үшін электрондық кестелер құрастырады, ғалымдар мен инженерлер бағдарламаны тәжірибелік деректерін өңдеу үшін жазады, ал әуесқой адамдар өз қығығушылығына және қанағаттану үшін бағдарламалар жазады. Бірақ бағдарламалық қамтыма белгілі бизнес-мақсаттармен өңделген, басқа да құралдарға немесе ақпараттық жүйе секілді бағдарлама өнімдеріне қосу үшін жасалған, бағдарламалық қамтыманың көбі кәсіби әрекетті көздейді. Кәсіби бағдарламалық қамтыма жобалаушыдан басқа адамдарға арналып, көбінесе, жеке адамдармен емес, белгілі бір топта өңделіп жасалады. Ол әрқашан қолданылып және өзгертіліп отырады.

Бағдарламалық қамтыманы өңдеу жеке бағдарламалар үшін емес, кәсіби бағдарламалық қамтыманы қолдауға арналған. Ол жеке бағдарламалық қамтыма үшін маңызы жоқ, бағдарламалық сипат тізімге, әрлендіруіне және эволюциясына арналған тәсілдерді сақтайды. Сізге бағдарламалық қамтыманың не екенін түсіну үшін мен *1.1-суретте* жиі қойылатын сұрақтарды сараптадым.

Сұрақ	Жауап
Бағдарлама дегеніміз не?	Компьютерлік бағдарлама және құжаттандыру. Бағдарлама нақты тұтынушыға немесе ортақ нарыққа арналып әзірленеді.
Жақсы бағдарламаның қасиеттері қандай?	Жақсы бағдарлама тұтынушының барлық сұраныстарын қанағаттандырып, икемді, тәуелді және пайдалы болуы керек.
Бағдарламалық жасақтама дегеніміз не?	Бағдарламалық жасақтамаға байланысты барлық аспектілерді қамтитын пән.
Бағдарламалық жасақтаманың негізгі әрекеттері қандай?	Бағдарламалық сипаттама, бағдарламаны әзірлеу, мақұлдау және бағдарлама эволюциясы.
Бағдарламалық жасақтама мен информатиканың айырмашылығы қандай?	Информатика теория мен негізгі тұжырымдарды қамтиды. Бағдарламалық жасақтама тікелей бағдарламаны әзірлеуге және жеткізуге негізделеді.
Бағдарламалық жасақтама мен жүйелік жасақтаманың айырмашылығы қандай?	Жүйелік жасақтама бағдарлама, техникалық қамсыздандыру сияқты компьютерге байланысты барлық әзірлеулерді қамтиды. Ал бағдарламалық жасақтама жалпы бағдарламаны әзірлеуге ғана байланысты болады.
Бағдарламалық жасақтаманың негізгі қарсылықтары қандай?	Түрлілікпен жұмыс, әзірлеу уақытын қысқарту және шынайы сапалы бағдарлама әзірлеу.
Бағдарламалық жасақтаманың шығындары қандай?	60% әзірлеу шығындары, 40% тестілеу шығындары. Тұтынушылық бағдарламаның эволюциялық шығындары әзірлеу шығындарынан асып түседі.
Бағдарламалық жасақтаманың ең жақсы техникалары мен тәсілдері қандай?	Бағдарламалардың барлығы мамандырылған және әзірленген болуға тиіс болғандықтан, әртүрлі жүйелерге әртүрлі тәсілдер қолданады. Мысалы, ойын-сауық бағдарламаларын әзірлеуде прототип қолданса, қауіпсізлігі жоғары бақылау жүйелері толық анализге келетін сипаттаманы қажет етеді. Сондықтан бір тәсіл басқасынан жақсы деп айтуға болмайды.
Интернеттің бағдарламалық жасақтамаға әсері қандай?	Интернет бағдарламалық қызметтердің және кең таралған қызмет негізіндегі жүйелердің дамуына жол ашты. Вебке негізделген жүйелердің әзірленуі бағдарламалау тілдерін және бағдарламаны қайта қолдануды дамытты.

1.1-сурет. Бағдарлама туралы кең таралған сұрақтар

Көп адамдар бағдарламалық қамтыманы компьютерлік бағдарламаның басқа атауы деп ойлайды. Бірақ бағдарламалық қамтыманы өңдеу жайлы сөз қозғайтын болсақ, бағдарламалық қамтыма тек қана бағдарлама емес, сонымен қатар ол осы бағдарламалардың дұрыс жұмыс істеуіне негіз болатын құжаттама мен конфигурациялық деректерге қатысы бар. Кәсіби бағдарламалық қамтыма жүйесі көбінесе, бір емес бірнеше бағдарламадан тұрады. Әдетте, жүйе көптеген жеке бағдарламалардан және осы бағдарламаларды орнату үшін арналған конфигурациялық файлдардан тұрады. Ол жүйенің құрылымын сипаттайтын жүйелік құжаттамадан тұрады; жүйені қалай қолдану жайлы түсіндіретін пайдаланушы құжаттамасы мен жаңадан шыққан өнім жайлы дерек алуға арналған вебсайттардан тұрады.

Бұл – кәсіби және әуесқой бағдарламалық қамтыма өңдеу арасындағы ең маңызды ерекшелік. Егер сіз бағдарламаны өзіңіз үшін жазсаңыз, сізден басқа ешкім қолданбайды және бағдарлама үшін құжаттама немесе жоспар жазудың қажеттілігі жоқ. Бірақ егер сіз бағдарламалық қамтыманы басқа адамдар үшін жазсаңыз және басқа инженерлер бағдарламаңызды өзгертетін болса, сіз қосымша деректермен және құпиямен қамтамасыз етуіңіз керек.

Бағдарламалық қамтыма жобалаушылары бағдарламалық қамтыма өнімдерін (мысалы, сатылатын бағдарламалық қамтыма) өңдеумен айналысады. Бағдарламаық қамтыма өнімдерінің екі түрі бар:

1. *Жалпы өнімдер.* Бұл өңдеуші мекемеден шығарылатын автономдық жүйелер, ол өнімдерді әрбір сатып алушы нарықта сатып ала алады. Мысалы, мұндай өнімдер мәтіндік процессорлар, жобаны басқару құралдары, деректер базасы, суреттер кестесі сияқты компьютерлік бағдарламалар. Сонымен қатар бұл белгілі бір нысананы көздейтін кітапханалық-ақпараттық жүйелер, бухгалтерлік есеп және тіс емдейтін тік қолданбаларды қамтиды.
2. *Жеке өнімдер (немесе тапсырыс).* Бұл жүйелер жеке сатып алушыға арналған. Бағдарламалық мердігер сол сатып алушыға арнап бағдарламалық қамтыма жасайды. Осындай бағдарламалық қамтыма электрондық құралдарды басқаратын жүйелерден, нақты бизнес-үдерісті қолдайтын және әуе қозғалысын басқаратын жүйеден тұрады.

Осы екі бағдарламалық қамтымалар арасында ең басты ерекшелік – жалпы өнімдерді шығаратын мекемелер бағдарлама сипат тізімін бақылайды. Пайдаланушылық өнімдер үшін сатып алған мекеме арқылы сипат тізімдер өңделіп, бақыланады. Бағдарламалық қамтыма жобалаушылары сол сипат тізімге жұмыс істеу керек.

Бірақ сол өнімдік жүйелердің арасындағы айырмашылықтар азаюда. Қазір көптеген жүйелер өнімдерін жалпы өнімдер негізінде құрастырып, кейін сатып алушының сұранысына байланысты өзгертеді. Кәсіпорын Қорларын Жоспарлау (КҚЖ) жүйелері, SAP жүйелері осындай тәсілдің мысалы бола алады. Бұл жерде бизнес әдістері және үдерістері туралы ақпарат беру, қажетті есеп беру және т.б. арқылы компания күрделі үлкен жүйеге бейімделген.

Сипаттамалар	Баяндама
Жөнделу	Тұтынушы сұраныстарының өзгеруіне байланысты бағдарлама әрқашан өзгерістерге дайын болып әзірленуі тиіс. Бұл өте маңызды сипаттама. Өйткені, бағдарламаның жөнделуі қазіргі тұрақты емес кәсіпшіліктің маңызды сұранысы.
Тәуелділік және қауіпсіздік	Бағдарламаның тәуелділігі нақтылық, қауіпсіздік және сенімділік сияқты сипаттамаларды қамтиды. Тәуелді бағдарлама жүйелік сәтсіздікке ұшыраған жағдайында да физикалық немесе экономикалық шығындарға әкелмеуі керек. Жаман ойлы адамның жүйені зақымдауға мүмкіншілігі болмауы керек.
Нәтижелілік	Бағдарлама жүйенің жадын немесе процессордың айналымдарын керегінен артық жұмсамауы керек. Және де нәтижелілік уақытты, жадты тазалауды қамтиды.
Жарамдылық	Әр бағдарлама оны қолданатын тұтынушыға жарамды болуы керек. Яғни, ол түсінікті, пайдалы және тұтынушы қолданатын басқа жүйелермен үйлесімді болуы керек.

1.2-сурет. Жақсы бағдарламаның сипаттары

Біз кәсіби бағдарламалық қамтыманың сапасы жайлы айтатын болсақ, бағдарламалық қамтыманың жобалаушыдан басқа адамдардың да әрекетін ескеруіміз қажет. Сондықтан бағдарламалық қамтыманың әрекетіне қатысы жоқ. Бұл бағдарламалық қамтыманың сапасы мен бағдарламалық қамтыма төл сипаттарының функцияларында байқалды. Осы сипаттардың мысалы ретінде пайдаланушының сұранысына арналған бағдарламаға жауап беру уақытын және бағдарламалық құпияның түсініктілігін алсақ болады.

Бағдарламалық қамтыма жүйесінен күтетін нақты белгіленген сипаттар тікелей оның қолданылуына байланысты. Сондықтан да банк жүйесі қауіпсізденген, интерактивті ойындар жауапты, телефон жүйесі қауіпсізденген және т.б болуы керек. Бұл *1.2-суретте* көрсетілген сипаттардың жинағына кіре алады.

1.1.1. Бағдарламалық жасақтама

Бағдарламалық жасақтама – инженерлік пән, ол бағдарламалық қамтыманың бастапқы дәрежеден эксплуатация өткеннен кейін де қолдайтын бағдарламалық қамтыманың бүкіл аспектілеріне қатысы бар. Бұл ережеде мынандай екі кілтті сөздер бар:

1. *Инженерлік пән.* Инженерлер затты жұмыс істетеді. Олар керек жерде теорияны, тәсілдерді және құралдарды қолданады. Бірақ олар таңдамалы түрде қолданылады, кейде мәселенің шешімін табу барысында ыңғайлы теориялар немесе тәсілдер болмаған жағдайда инженерлер өздері шешім табады. Инженерлер сол сияқты өздерін ұйымдастырушылық және есеп қиындықтарын шешу үшін жұмыс істеу керек екендігін мойындайды, сондықтан олар шешімді осы шектеудің ішінде ғана іздейді.
2. *Бағдарламалық қамтыманы өңдеу аспектілері.* Бағдарламалық жасақтама бағдарламалық қамтыманың техникалық үдерістерімен байланыспаған, ол сол сияқты бағдарламалық өнімдерді қолдау мақсатында бағдарламалық жобаны басқару және құралдарының дамуы, тәсілдер және теориялар сияқты әрекеттерді қамтиды.

Инженерлік іс – бұл белгілі уақытта қажетті сапасымен нәтиже алу. Көбінесе, бұл инженерлер жетілдірушілер емес деген тұжырымға сәйкес келеді. Бағдарламаны өз қажетіне жазатын адамдар бағдарламаны дамыту үшін өз қалауынша уақытты жұмсай алады.

Жалпы түрде сапалы бағдарламалық қамтыма жасау үшін инженерлер жүйелік және құрылымды тәсіл қолданған. Бірақ инженерлік іс арнайы жағдайларға шығармашылық тұрғыдан тиімді тәсіл табу, кейбір жағдайларда мұндай тәсілді қолдану өте тиімді болуы мүмкін. Шығармашылық тұрғыдан графикалық әрлендіру және бағдарламалық қамтымада тәжірибе қажет ететін веб-жүйелерді дамыту үшін қолайлы.

Бағдарламалық жасақтама екі түрлі себеп үшін маңызды:

1. Көптеген адамдар да, қоғам да заманауи бағдарламалық қамтыма жүйесіне сенім артады. Біз сапалы жүйені тез және үнемді құрастыра алуымыз керек.
2. Бағдарламалық қамтымада бағдарламаның ұзақ өмір сүруі үшін бағдарламалық әдістерді қолданады, ол тек қана бағдарламаны құраудан тұрмайды. Жүйелердің нағыз шығыны көбіне, эксплуатациядан өткеннен кейін ғана байқалады.

Бағдарламалық жасақтамада қолданылатын жүйелік тәсілді кейде бағдарламалық қамтыма үдерістері деп атайды. Бағдарламалық қамтыма үдерістері – бағдарламалық қамтыманы өңдеуге арналған әрекеттердің нәтижесі. Барлық бағдарламалық қамтымаларға ортақ төрт іргелі әрекеттер бар. Олар:

1. Бағдарламалық қамтыма сипат тізімінде сатып алушылар мен инженерлер өңделіп шығатын өнімді анықтап және өнімнің әрекеттеріне шек қоя алады;
2. Бағдарламалық қамтыманың өңделуі, бұл жерде бағдарламалық қамтыма жобаланып, бағдарламаланады;
3. Сәйкестікке тексеру, бұл жерде бағдарламалық қамтыма тексеріледі, сатып алушының сұранысына сәйкес тексеріледі;

4. Бағдарламалық эволюция, бұл жерде сатып алушы мен нарықтың сұранысына сәйкес бағдарламалық қамтыманы өзгертеді;

Әртүрлі жүйе үшін сәйкес түрлі даму үдерістері бар. Мысалы, ұшақтағы шынайы уақыт тәртібіндегі бағдарламалық қамтыманы өңдеуден бұрын құрылысын анықтау қажет. Электрондық коммерцияда тізім мен бағдарлама бірге құрастырылады. Демек, осы ортақ қызметтер әртүрлі құрастырылып, бағдарламалық қамтыма түріне қарап тәптіштеу деңгейі де әртүрлі болады. Бағдарламалық қамтыма үдерістері туралы *2-тарауда* толығырақ баяндадым.

Бағдарламалық инженерлік іс *информатикамен* де, инженерлік жүйемен де тығыз байланысты:

1. Информатика компьютерге және бағдарламалық жүйелерге негіз болатын теориямен және тәсілдермен тығыз байланысты, ал Бағдарламалық жасақтама бағдарламалық қамтыманың тәжірибелік қателіктерімен жұмыс істейді. Информатиканың кейбір заттарын білу бағдарламалық қамтыма инженерлері үшін өте маңызды. Информатика теориялары көбінесе, кіші бағдарламаларға пайдаланылады. Үлкен және күрделі бағдарламалар үшін информатиканың теориялары тиімсіз.
2. Инженерлік жүйе бағдарламалық қамтымадағы маңызды орын алатын күрделі жүйелердің даму аспектілерімен және эволюциясымен айналысады. Инженерлік жүйе, сонымен қатар аппараттық жүйелерде, саясатта, жобалау үдерістерінде, жою жүйелерінде және бағдарламалық қамтымада қолданылады. Жүйе инженерлері жүйені анықтауға, жалпы құрылысын анықтауға және әртүрлі бөлігін интегралдауға қатысады. Олардың жүйе компонеттерінің (аппараттық, бағдарламалық қамтыма, т.б) инженерлік ісіне қатысы жоқ.

Келесі бөлімде айтылғандай бағдарламалық қамтыманың бірнеше түрі бар. Бағдарламалық қамтыма өңдеуде барлығына бірдей қолданылатын әмбебап тәсілдер жоқ. Сонда да бағдарламалық қамтыманың көбіне сәйкес келетін үш жалпы тұжырымдамасы бар

1. *Гетерогендік.* Қазір компьютер және ұялы телефонға жауап беретін, желіде қажет, үлестірілген жүйе ретінде жұмыс істейтін жүйелер көптеп кездеседі. Олар компьютерлерде жұмыс істегеннен кейін, ұялы телефондарға да енгізілуі мүмкін. Көбіне, сіз әртүрлі бағдарламалық тілдерде жазылған ескі жүйелерді жаңа бағдарламалық қамтымаға интегралдаумен айналысасыз. Бұл жерде басты мәселе – сенімді гетерогендікті жеңу үшін икемді бағдарламалық қамтыма жасау үшін тәсілдерді өңдеу.
2. *Бизнес және қоғамдық өзгерістер.* Шапшаң дамып жатқан экономика мен жаңа технологиялар секілді бизнес пен қоғам да жылдам алға жылжуда. Олар өздерінің бар бағдарламалық қамтымаларын тез арада өзгертеуге және жаңа бағдарлама өңдеуге қабілетті болуы қажет. Бағдарламалық қамтыма

әдістерінің көбісі ұзақ уақытта өндіріледі және белгіленген уақытынан кеш жеткізіледі. Олар кешіккен өнім үшін сатып алушыға бағасын арзандатуы қажет.

3. *Қауіпсіздік және сенімділік.* Бағдарламалық қамтыма өміріміздің барлық аспектілерімен тығыз байланысты болғандықтан, бағдарламалық қамтымаға сенім арта алуыңыз өте маңызды. Бұл әсіресе, веб-интерфейс және веб-беттер арқылы шығатын қашықталған бағдарламалық қамтыма жүйесіне қатысы бар. Біз бағдарламалық қамтымаға қаскөйлердің шабуыл жасай алмауына және ақпараттық қауіпсіздігіне сенімді болуымыз қажет.

Әрине, бұл мәселелер өзара тәуелсіз мәселелер. Мысалы, бұл тез арада ескі жүйеге веб-қызмет интерфейсін енгізуде қажет болуы мүмкін. Бұл мәселелерді шешу үшін бізге жаңа әдістер мен құралдар, инновациялық біріктіру тәсілдер мен бар бағдарламалық қамтыма әдістері қажет болады.

1.1.2. Бағдарламалық жасақтаманың көпбейнелілігі

Бағдарламалық жасақтама – тәжірибелік шығындарды қамтамасыз ететін, уақыт аралығын және мәселенің сенімділігіне, бағдарламалық қамтымаға қойылған сатып алушы мен жобалаушының сұраныстарына қарайтын, бағдарламалық қамтыманың өнімін шығаруға жүйелік бағыт. Бұл жүйенің қалай іске асқаны бағдарламалық қамтыманы өндірген мекемеге, бағдарламаның түріне және даму үдерісіне қатысқан адамдарға тікелей байланысты. Бағдарламалық қамтыма инженерлік ісінде барлық жүйелер мен компанияларға сәйкес келетін әмбебап әдістер жоқ. Бағдарламалық қамтыманың инженерлік әдістері тек соңғы 50 жылда дамыды.

Қандай бағдарламалық қамтыма әдісін қолдануды шешу қолданбаның түріне байланысты. Қолданбаның бірнеше түрі бар:

1. *Автономды қолданбалар.* Бұл – жергілікті компьютерлерде дербес компоютерлер сияқты орнатылған қолданба жүйелер. Оларда барлық қажетті функционалдар бар және олар желіге қосылуды қажет етпейді. Ондай қолданбаның мысалы ретінде компьютерлердегі офистік қолданбаларды, CAD бағдарламасын, фотосуреттерді өңдеу бағдарламасын және т.б алуға болады.
2. *Интерактивті транзакция негізіндегі қолданбалар.* Бұл қашықталған компьютерлерде орнатылған қолданбалар, пайдаланушылар өздерінің компьютерлерінен немесе терминалдан кіре алады. Әлбетте, сіз тауар және қызметті сатып алу үшін қашықталған жүйемен қатынасатын е-коммерция сияқты веб-қолданбалардан тұрады. Бұл класс, сонымен қатар бизнес-жүйе қолданбаларынан тұрады, бизнес – өз жүйелеріне веб-браузер немесе белгіленген клиенттік бағдарлама арқылы және пошта және суретпен алмасу сияқты бұлттық қызметтеріне жол ашады. Интерактивті қолданбалар әр

транзакция сайын жаңаланып отыратын үлкен деректер сақтау орны болып табылады.

3. *Ендіруші жүйелер.* Ақпараттық құралдарды бақылайтын және басқаратын бағдарламалық бақылау жүйелері. Бұл жүйеде басқа жүйелерге қарағанда ендіруші жүйелер көп. Ендіруші жүйелердің мысалы ретінде ұялы телефондағы бағдарламалық қамтыма, көліктегі тежегішті антибұғаттау бағдарламасы, микротолқындық пеште тамақ пісіру үдерісін бақылайтын бағдарламаны алсақ болады.
4. *Жүйені кестелі өңдеу.* Бұл жерде деректерді үлкен кесте түрінде өңдейтін бизнес жүйелер бар. Олар сәйкес шығуды құру үшін, көптеген жеке кірулерді өңдейді. Кестелі жүйелердің мысалы ретінде периодтық биллинг жүйелерін, телефондық жүйелер және еңбек ақысын төлеу жүйесін алуға болады.
5. *Ойын-сауық жүйелері.* Бұл – жеке қолданыс үшін және пайдаланушының көңілін аулау үшін арналған жүйелер. Олардың көбісі ойынның бір түрі немесе әртүрлі болуы мүмкін. Ойын-сауық жүйесінде пайдаланушыға қарым-қатынастың сапалығы өте маңызды болып табылады.
6. *Үлгілеу мен сылтау іздеу жүйелері.* Бұл бір-бірімен өзара қарым-қатынаста болатын физикалық жағдайлар мен үдерістерді ғалымдар мен инженерлер құрастырған. Оған көбінесе, қарқынды есептік және іске қосылу үшін жоғарғы өнімділік қажет.
7. *Деректер жинау жүйесі.* Бұл жүйелер қоршаған ортадан бергіш жиынтықтардың көмегімен деректер жинап, оны басқа жүйелерге өңдеуге жібереді. Бағдарламалық қамтыма бергіш құралымен қатынасуы керек, олар қашықталған немесе өзгеретін қауіпті жерлерде орналасуы мүмкін.
8. *Жүйелердің жүйесі.* Бұл – басқа бағдарламалық жүйелерден тұратын жүйе. Олардың кейбірі әмбебап бағдарламалық өнім болуы мүмкін, мысалы, электрондық кесте. Жинақтағы басқа жүйелер осы ортаға арналып жазылуы мүмкін.

Әрине, бұл жүйелердің шекаралары анық емес. Егер сіз ұялы телефондарға ойын құрастыратын болсаңыз, сіз ұялы телефондар жобалаушылары көңіл аударатын кейбір шектеулерге көңіл аударуыңыз керек. Кестелік үдерісті жүйелер көбінесе веб-жүйелермен бірге қолданылады. Мысалы, мекемеде жол шығынын өтеу веб-қолданба арқылы жіберілуі мүмкін, бірақ кестелік қолданба да бір айлық төлемге өңделеді.

Бағдарламалық қамтыманың әртүрлі сипаттамасы болғандықтан, бағдарламалық қамтыманың түріне байланысты әртүрлі бағдарламалық қамтыма әдістерін қолданасыз. Мысалы, көлікте орналасқан ендіруші жүйенің қауіпсіздік кепілі болып табылады және көлікке орнатқан жағдайда Тұрақты Жадтау Құрылғысында (ТЖҚ) сақталады. Сондықтан оны ауыстыру өте қымбат. Ондай жүйе сатылымнан кейін қиындықтар туындамауы үшін, сапалы тексеруден және бекітуден өтуі керек. Пайдаланушымен қарым-қатынас өте аз, сондықтан пайдаланушы сенетін интерфейсті түпнұсқалау даму үдерісінің қажеті жоқ. Веб-

жүйелер үшін жеткізу мен итерациялық өңдеу тәсілдері қолайлы, бұл жүйе қайта пайдаланған компоненттерден тұрады. Бірақ ондай тәсіл жүйелердің жүйесі үшін тиімсіз, ол жерде жүйелердің жеке өндірілуі үшін алдымен жүйелердің қарым-қатынасы жайлы толық баяндалуы қажет.

Солай болса да, бағдарламалық қамтыманың барлық жүйелеріне қолданылатын, бағдарламалық қамтыма өңдеудің негізгі ұстанымдары бар:

1. Олар басқарылатын және түсінікті даму үдерісімен өндірілуі қажет. Бағдарламалық қамтыма өндіретін мекеме бағдарламаның даму үдерісін жоспарлап, қандай өнім шығаратынын және шығару мерзімін білуі қажет. Әрине, әртүрлі бағдарламалық қамтыманың түріне байланысты сәйкес үдерістер қолданылады.
2. Барлық жүйелер үшін сенімділік және өнімділік өте маңызды. Бағдарламалық қамтыма қажет кезде қолдануға дайын болуы тиіс, қателіксіз дұрыс жұмыс істеуі керек. Ол сыртқы шабуылдардан қорғалып, жұмыс істеу үдерісінде қауіпсіз болуы қажет. Жүйе ресурстарды жұмсамай, нәтижелі жұмыс істеуі тиіс.
3. Бағдарламалық қамтыманың тізімін басқару және түсіну өте маңызды. Сіз әр сатып алушы мен жүйе пайдаланушыларының осы өнімнен не күтетінін білуіңіз қажет, бюджетке және графикке сәйкес келетіндей етіп, олардың күтімін басқаруыңыз керек.
4. Сіз бар ресурсты тиімді пайдалануыңыз керек. Ол қажет жерде өндірілген бағдарламалық қамтыманы қайта пайдаланып, одан әрі дамыту дегенді білдіреді.

Бұл үдерістің іргелі тұжырымдамалары: сенімділік, сұраныс, басқару және қайта пайдалану оқулықтың маңызды тақырыптары болып табылады. Олар әртүрлі әдістерде басқаша көрінгенімен, бағдарламалық жасақтаманың кәсіби негізінде жатыр.

Бұл іргетастар бағдарламалау мен іске асыруды қамтымайтынын білуіңіз қажет. Бұл оқулықта бағдарламалаудың нақты тәсілдері қамтылмаған, өйткені бір жүйені басқа жүйемен араластыру өте қиын. Мысалы, Ruby бағдарламалау тілі веб-негізіндегі жүйелерге қолайлы болғанымен, ендіруші жүйелер үшін тиімсіз.

1.1.3. Бағдарламалық жасақтама және Веб

Дүниежүзілік тордың дамуы өмірімізге айтарлықтай әсер етті. Бастапқыда веб-ақпарат сақтайтын жер болатын және бағдарламалық қамтыма жүйелеріне әсері аз болған. Бұл жүйелер жергілікті компьютерлерде жұмыс істеп, тек мекеменің ішінде ғана қолданылды. 2000 жылдардан браузерлерге жаңа функциялар қосылып, дами бастады. Бұл веб-жүйелердің өндірілетінін білдіреді, бірақ бұл жүйелер интерфейс арқылы емес, веб-браузер арқылы қолданылады. Бұл интернет арқылы инновациялық қызметтерді жеткізетін, жаңа жүйелердің көлемді дамуына әкелді.

Олар көбінесе, пайдаланушылардың экрандарындағы хабарламалармен төленеді және пайдаланушыдан нақты төлемді қажет етпейді.

Осы жүйелердің өнімдері сияқты, веб-браузерлердің дамуы бағдарламалық қамтыманың ұйымдастырылуына және бизнестің эволюциялық дамуына әсер етеді. Бағдарламаны жазып, оны дербес компьютерлерде орнатудың орнына, бағдарламалық қамтыма веб-серверлерге ендірілді. Осыдан кейін бағдарламалық қамтыманы ауыстыру және жаңалау арзанға түсті, өйткені әр компьютерге бағдарламалық қамтыма орнатудың қажеті жоқ. Сонымен қатар, бұл шығындарды да азайтты, себебі пайдаланушылық интерфейсті дамыту қымбат. Сонымен, мүмкіншіліктері бар көптеген мекемелер компаниялардың бағдарламалық қамтыма жүйелерімен веб-тік қарым-қатынасқа көшті.

Веб-тік жүйелердің дамуынан кейін веб-қызметтері орын алды. Веб-қызметтер – бағдарламалық қамтыма компоненттері, олар нақты, пайдалы функцияларды интернет арқылы атқарады. Қолданбалар әртүрлі компаниялар арқылы жеткілізген веб-қызметтерді интегралдау арқылы құрылады. Бұл байланысты динамикалық десек те болады, өйткені қолданба жұмыс істеген сайын әртүрлі веб-қызметтерді қолдана алады. Мен бұл тәсілді *19-тарауда* баяндадым.

Соңғы бес жылда «бағдарламалық қамтыма қызмет ретінде» деген тұжырымдама пайда болды. Осыған байланысты бағдарламалық қамтыма жергілікті компьютерлерде емес, интернет арқылы істейтін «есептік бұлттар» арқылы жұмыс істейді деген болжамдар болған. Егер веб-пошта сияқты қызметтерді қолдансаңыз, сіз бұлттық жүйелерді қолданатыныңызды білдіреді. Бұлттық есептеулер – бұл көптеген пайдаланушылар бөлісетін байланысқан компьютерлер жүйесі. Пайдаланушылар бағдарламалық қамтыманы сатып алмайды, бірақ төлемақысы бағдарламалық қамтыманың қолданылуына байланысты төленеді.

Сондықтан желінің пайда болуы бағдарламалық қамтыма бизнесінің айтарлықтай өзгеруіне ықпал етті. Интернеттің пайда болуынан бұрын, монолиттік бизнес-қолданбалар қолданылған, яғни жеке бағдарламалар әрқайсысы жеке компьтерлерде немесе компьютерлік кластерлерде орнатылған. Қарым-қатынас жергілікті, демек, мекеме қабырғасында болды. Қазір бағдарламалық қамтыма бүкіл әлемге тиімді үлестірілген. Бизнес-қолданбалар басынан бастап бағдарламаланбаса да, көптеген құраушылардың және бағдарламалардың қайта пайдалануынан тұрады.

Бұл бағдарламалық ұйымдастырулардағы радикалды өзгерістер веб-жүйелердің өзгеруіне әкелді. Мысалы:

1. Веб-жүйелерді құрастыруда бағдарламалық қамтыманың қайта қолданылуы басты тәсілге айналды. Бұл жүйелерді құрастырғанда сіз біріншіден, бар жүйелер мен бағдарламалық құраушылардан қалай жаңа жүйені құрастыруды ойластырасыз.
2. Қазір ондай жүйелерден алдын ала сұраныстар талап ету тиімсіз деп танылған. Веб-жүйелер бірте-бірте өңделіп, іске асқаны жөн.

3. Пайдаланушы интерфейсі веб-браузерлердің мүмкіндіктерін шектейді. Бірақ *AJAX* технологияларында пайдаланушы интерфейстері веб-браузерлерде құрастырылады, дегенмен, бұл технологияларды қолдану қиындық туды-рады. Көбінесе, жергілікті сценарийлері бар веб-формалар қолданылады. Веб-жүйелеріндегі интерфейстер компьютерге арналған пайдаланушылар интерфейсіне қарағанда қарапайым келеді.

Алдыңғы бөлімдерде айтылғандай, бағдарламалық қамтыманың іргелі тұжырымдамалары веб-жүйелерге де қолданылады. XX ғасырда күрделі жүйелердің дамуы қазір де вебтік бағдарламалық қамтыма үшін маңызы зор.

1.2. Бағдарламалық жасақтама этикасы

Басқа да инженерлік іс пәндері секілді, бағдарламалық жасақтаманы өңдеу сол аумақта жұмыс істейтін адамдардың бостандығын шектейтін, әлеуметтік-құқықтық негізде іске асады. Инженер-бағдарламалаушы ретінде сіздің жұмысыңыз тек қана техникалық қателіктерді жөндеу емес, одан да кең өкілеттікті қамтитынын білуіңіз қажет. Сонымен қатар кәсіби инженер ретінде сіз этикалық және адамгершілік жағынан жауапты болуыңыз қажет.

Сол сияқты сіз шындықтың қалыпты стандартын және біртұтастығын қолдауыңыз қажет. Бағдарламалық қамтыманың кәсіби инженері болуы үшін, сіз дағдыларыңызды пайдаланып, әділетсіз жолмен емес, өз еңбегіңізбен жетуіңіз тиіс. Сонда да кейбір аумақтарда қолайлы тәртіп стандартының заңмен байланысы жоқ, әсіресе, кәсіби жауапкершіліктің жоқтығы білінеді. Олар:

1. *Құпиялық.* Құпиялық туралы ресми келісімге қол қойылмаса да, сіз жұмыс беруші мен клиенттің құпиялығын сақтауыңыз қажет.
2. *Құзіреттілік.* Сіз құзіреттілік деңгейіңізді бұрмаламауыңыз керек. Сіз өз құзіреттілік деңгейінен асатын жұмыстан бас тартпауыңыз қажет.
3. *Зияткерлік меншіктік құқық.* Сіз патент, авторлық құқық секілді зияткерлік құқықты басқаратын жергілікті заңдарды білуіңіз қажет. Сіз жұмыс берушілер мен клиенттердің зияткерлік меншігін қауіпсіздікпен қамтамасыз етуіңіз керек.
4. *Компьютерді теріс пайдалану.* Сіз техникалық икеміңізді бөгде адамның компьютерін теріс пайдалану үшін қолданбауыңыз керек. Компьютерлік технологияларды теріс пайдалану ауқымы дағдылы пайдаланудан аса маңызды (вирусты тарату немесе басқа қауіпті бағдарламалар) пайдалануға дейін барады.

Кәсіби қауымдар мен мекемелер этикалық нормаларды орнатуда маңызды рөл атқарады. Есептеу Техникасы Ассоциациясы (ағылш. Association for Computing Machinery, ACM), Электроника және Электроника Инженерлер Институты- IEEE

(ағылш. Institute of Electrical and Electronics Engineers) және Британ компьютерлік қауымы секілді мекемелер тәртіп кәсіби-кодын немесе этика кодексін жариялайды. Сол мекемелердің өкілдері өкілдікке жазылғанда сол кодтың қағидаларын сақтауға міндетті. Бұл тәртіп кодекстері іргелі этикалық тәртіппен тығыз байланысты.

Бағдарламалық жасақтаманың этикалық коды және мамандырылған тәжірибе.

ACM/IEE-нің Бағдарламалық жасақтама этикасының және мамандырылған тәжірибенің тапсырмалары.

КІРІСПЕ
Кодтың қысқаша нұсқасы жақсы тәжірибенің негізгі тараптарын суреттейді. Толық нұсқада бұл тапсырмалардың мысалдары беріліп, кеңірек талданады. Бағдарламалық жасақтаманың мамандары төменде көрсетілген сегіз қағидаттарды ұстану керек:

1. Қоғам. Бағдарламалық жасақтаманың мамандары қоғамның сұраныстарына мән беру керек.
2. Тұтынушы және жұмыс беруші. Бағдарламалық жасақтаманың мамандары тұтынушының және жұмыс беруші адамның сұраныстарын ең жоғарғы деңгейде орындау керек.
3. Өнім. Бағдарламалық жасақтаманың мамандары даярлаған өнімінің сапасы ең жоғарғы денгейде екеніне сенімді болу керек.
4. Талқылау. Бағдарламалық жасақтаманың мамандары талқылауларында ешкімнен тәуелсіз болу керек.
5. Басқару. Бағдарламалық жасақтаманы басқаратын адамдар және кошбасшылар бағдарлама әзірлеудің және жөнделудің этикасының негізгі қағидаттарын ұстану керек
6. Мамандық. Бағдарламалық жасақтаманың мамандары өзінің мамандығының репутациясын қоғамдағы қызығушылықты арттыруға тырысу керек.
7. Әріптестер. Бағдарламалық жасақтаманың мамандары әріптестеріне адал және қайырымды болу керек.
8. Өзі. Бағдарламалық жасақтаманың мамандары өмір бойы өзінің маманының жақсы тәжірибелерін зерттеп, өзіне пайдасын алу керек.

1.3-сурет. ACM/IEE-нің этикалық коды (© IEEE/ACM 1999)

ACM және IEEE сияқты кәсіби ассоциациялар этика кодексін және кәсіби іс-тәжірибені дайындау үшін бірге ынтымақтасты. 1.3-суретте қысқаша түрде көрсетілген. Бұл кодтың мәні бірінші екі пунктта толығырақ түрінде:

Компьютерлер саудада, өнеркәсіпте, өкімет басқаруда, медицинада, білім алуда, ойын-сауықта және бүкіл қауым үшін орталық және негізгі рөл атқарады. Бағдарламалық қамтыма инженерлер: бағдарлама жүйесіне талдау жасаумен, жобалаумен, сипат тізім жасаумен, өңдеумен, сертификаттаумен, техникалық қызмет көрсетумен және тестілеумен ай-

налысады. Бағдарлама жүйесін дамытуда айтарлықтай рөл атқаратын бағдарлама инженерлері – жақсылық жасау немесе зиян келтіру сияқты немесе басқа адамдардың сондай әрекет жасауына ықпал ету сияқты маңызды мүмкіндіктерге ие. Олар өз жігерлерінің жақсылыққа жұмсалуы үшін, бағдарлама инженерлері бағдарламалық қамтыманың пайдалы және қадірлі кәсіп болуы үшін күш салуы керек. Осы жауапкершілікке байланысты, бағдарламалық қамтыма жобалаушыларының келесі этика кодексі мен кәсіби іс-тәжірибені ұстануы керек.

Кодекс тәртіпке және шешімдерге байланысты сегіз ұстанымдардан тұрады, олар кәсіби бағдарламалық инженерлерімен қабылданады, олардың ішіне оқытушылар, тәжірибешілер, менеджерлер, басқарушылар мен саясаткерлер, сол сияқты осы кәсіпке оқитын оқушылар мен студенттер де кіреді. Ұстанымдар жеке адамдардың, топтың және мекемелердің қарым-қатынастағы этикалық жауапкершілігін анықтайды. Әрбір ұстанымның тармағы осы қарым-қатынастардағы тиісті суреттерімен көркемделеді. Бұл міндеттер бағдарлама инженерінің адамгершілігіне негізделген, әсіресе, бағдарламалық инженерлердің жұмысынан зиян шеккен адамдарға ерекше күтім жасауды және бағдарламалық қамтыма іс-тәжірибесінің бірегей элементтері де күтімді қажет етеді. Бұл кодекс бағдарламалық инженер болам деушілер мен инженер болғысы келетіндерге арналған.

Қай жағдайда болса да, әртүрлі адамдардың түрлі көзқарастары мен мақсаттары болады, сіз сондай этикалық дилеммаларды кездестіресіз. Мысалы, егер сіз мекемедегі лауазымы жоғары саясаткермен келіспей қалсаңыз, сіз не істейсіз? Бұл нақты адамдармен келіспеушілік себебіне байланысты. Сіздің мекемедегі жағдайыңыз үшін дәлел келтіру маңызды ма, әлде қызметтен кеткен дұрыс па? Егер сіз бағдарламалық қамтыма жобасында қиыншылықтар бар екенін білсеңіз, сіз ол туралы басшылыққа қай уақытта айтасыз? Егер сіз олар жай ғана күдік тудырғанда айтсаңыз, сіз бұл жағдайда тым эмоционалды әсер тигізуіңіз мүмкін; егер сіз оны кеш қалдырсаңыз, ол қиыншылықтар шешілмеуі де мүмкін.

Мұндай этикалық дилеммарды біз бүкіл кәсіби өмірімізде кездестіреміз, олардың көбісі шағын немесе қиындықсыз шешілетін мәселелер. Олар шешілмейтін жағдайда инженерлер тағы бір қиындықты кездестіреді. Жұмыстан кету жайлы қағидатты шешім қабылданса, бұл басқаларға әсер етуі мүмкін, мысалы, серіктесіне немесе балаларына әсер етуі мүмкін.

Жұмыс беруші әдепсіз жолмен әрекет жасаса, кәсіби инженерлер үшін тым күрделі жағдай туындайды. Егер бір компания қауіпсіздік жүйесін жасауға жауапты болып, уақыт жетпегендіктен тексергенде қауіпсіздік көрсеткіші бұрмаланып тұрса, бұл жүйені орналастырғанда қауіпсіз болмауын, алдын ала айту немесе құпиялықты сақтау инженердің жауапкершілігіне кіреді ме, әлде жоқ па?

Бұл жерде қауіпсіздікке келгенде, ешнәрсе абсолютті бола алмайды. Бірақ жүйе алдын ала белгіленген критерийлермен тексерілмеуі мүмкін, алайда, ол критерийлер тым қатал болуы мүмкін. Жүйе қауіпсіз жұмыс істей беруі де мүмкін.

Сол сияқты қатесіз дұрыс жасалған жүйенің өзі де, кейде істемей қалуы мүмкін. Ерте анықталған мәселелер жұмыс беруші мен жұмыскерлерге кері әсерін тигізуі мүмкін, анықталмаған мәселелер басқаларға кері әсерін тигізеді.

Осындай сұрақтарда өз шешіміңіздің болғаны дұрыс. Бұл жерде қолайлы этикалық жағдай қатысатын адамдардың пікіріне байланысты. Ондай жағдайда жәбірленген адамдар, жәбірленген заттың көлемі шешімге ықпал етуі керек. Егер жағдай қауіпті болса, ұлттық баспасөз арқылы жариялануы мүмкін. Сонда да, сіз мәселені жұмыс берушінің құқығын бұзбай шешуіңіз қажет.

Келесі этикалық сұрақ ол – әскери-ядролық жүйенің дамуына қатысу. Кейбір адамдар мұндай мәселелерден өздерін жайсыз сезінгеннен, әскери жүйелермен байланысқысы келмейді. Ал кейбіреулер әскери жүйелерге жұмыс істегенімен, қарулану жүйелерінде жұмыс жасамайды. Бірақ басқалары үшін ұлттық қауіпсіздік – негізгі ұстаным болып табылады және қаруландыру жүйесіне қарсылық білдірмейді.

Мұндай жағдайда жұмыс беруші мен жұмыскер алдын ала бір-біріне өз пікірлерін айтқаны дұрыс. Егер мекеме әскери немесе ядролық бағытта жұмыс істесе, олар жұмыскерлердің қандай да болмасын тапсырманы орындауға дайын екенін анықтауы қажет. Егер жұмыскер жұмыс алғаннан кейін, ондай жүйеде жұмыс істегісі келмейтінін көрсетсе, жұмыс беруші тарапынан оны кейінгі уақытта да жұмысқа араластырмауы тиіс.

Этика мен кәсіби жауапкершіліктің бағдарламалық қамтыма жүйелерінің барлық жерде пайда болуына байланысты маңызы артты. Бұған этиканың барлық ұстанымдары қаралатын философиялық көзқараспен қарауға болады. Бағдарламалық қамтыма этикасы да сол ұстанымдарға қарайды. Бұл көзқарасты Лаудан (1995 ж.) және аз мөлшерде Хафф пен Мартин (1995) Джонсонның “компьютер этикасында” (2001) ұстанады.

Дегенмен, менің ойымша, бұл философиялық көзқарас – дерексіз және күнделікті тәжірибемен байланыссыз. Мен тәртіп және іс-тәжірибе кодексінде белгіленген нақты тәсілдерді қалаймын. Менің ойымша, этиканы бағдарламалық қамтыманы өңдеу барысында талқылаған жөн. Сондықтан мен бұл оқулықта дерексіз этикалық талқылауларды қоспадым, бірақ жаттығуларда көрсетілген.

1.3. Тақырыптық зерттеулер

Бағдарламалық қамтыма тұжырымдамаларды көрсету үшін, оқулықтағы үш түрлі жүйені мысал ретінде қолдандым. Бір ғана тақырыптық зерттеу таңдамаудың себебі – бағдарламалық қамтыманы өңдеу іс-тәжірибесі шығарылатын жүйенің түріне байланысты. Сондықтан мен қауіпсіздік және сенімділік, жүйені үлгілеу, қайта пайдалану және т.б. тақырыптарды таңдадым.

Тақырыптық зерттеулер ретінде үш жүйені таңдадым:

1. *Кіріктірілген жүйелер.* Бұл жүйе бағдарламалық қамтымада орналасып, оны басқарады. Кіріктірілген жүйелер физикалық өлшемдерден, ықыластылықтан, тамақтануды басқарудан және т.б.-дан тұрады. Кіріктірілген жүйенің мысалы ретінде мен медициналық құралдарды басқаратын бағдарламалық қамтыманы алдым.
2. *Ақпараттық жүйелер.* Бұл жүйенің негізгі мақсаты – ақпараттық деректер базасына мүмкіндікпен қамтамасыз ету және басқару. Ақпараттық жүйелер қауіпсіздік, ыңғайлылық, құпиялық және деректердің тұтастылығынан тұрады. Мен ақпараттық жүйенің мысалы ретінде медициналық жазбалар жүйесін қолдандым.
3. *Бергіштер негізінде деректер жинау жүйесі.* Бұл жүйенің негізгі мақсаты – бергіштер жиынтығынан деректер жинау және кейде сол деректерді өңдеу. Ондай жүйелерге арналған талаптар: сенімділік, қоршаған ортаның қолайсыз жағдайларында да сенімділікпен қамтамасыз ету, жөндеуге жарамдылық. Мен деректер жинау жүйесінің мысалы ретінде шөлдегі ауа райы станциясын қолдандым.

Мен бұл тарауда осы жүйелердің әрқайсысын толығымен баяндадым, одан да толығырақ сіз интернеттен қарай аласыз.

1.3.1. Инсулиндік помпа басқару жүйесі

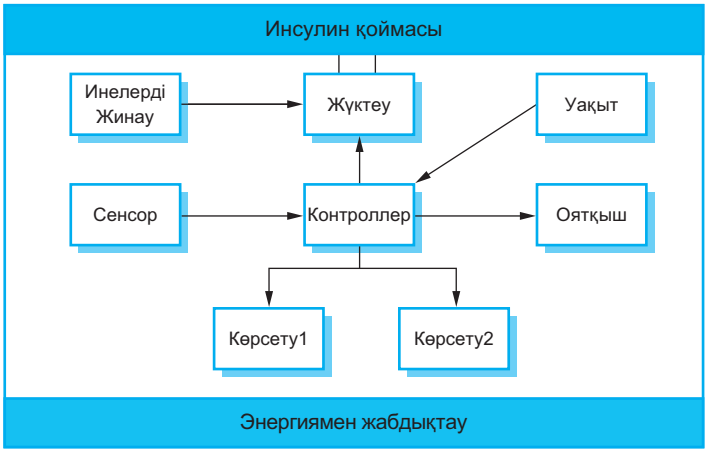
Инсулиндік помпа – ұйқы безінің жұмысын реттейтін (ішкі органдар), медициналық жүйе. Бұл жүйені басқаратын бағдарламалық қамтыма – бергіштерден деректер жинайтын және инсулиннің дозалық мөлшерін бақылайтын, помпаны басқаратын кіріктірілген жүйе.

Жүйені сусамыр ауруымен ауыратын адамдар қолданады. Сусамырда ұйқы безіндегі инсулин гормонының жеткілікті мөлшерде шығарылмағандықтан пайда болған ортақ жағдай. Инсулин қандағы глюкозаны (қантты) сіңіреді. Сусамырды емдеу әрдайым гендік-инженерлік инсулин инъекцияларын қабылдаудан тұрады. Сусамырмен ауыратындар алдымен қандағы қанттың мөлшерін тексеріп алып, кейін қабылдайтын инсулин мөлшерін есептейді.

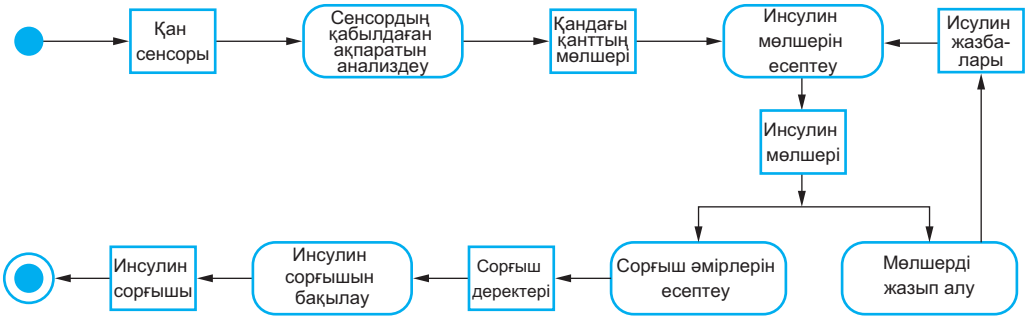
Бұл емдеудің қиындықтары: қабылдайтын инсулин мөлшері тек қан құрамындағы қанттың мөлшерінен ғана емес, соңғы инсулинді қабылдаған уақытына да тәуелді. Бұл қандағы қанттың тым аз мөлшерде (инсулин дозасы көп болып кетсе) немесе глюкозаның қандағы көп мөлшерде (инсулин аз болып қалса) болуына алып келуі мүмкін. Глюкозаның қанда аз мөлшерде болуы – бұл күрделі жағдай, өйткені нәтижесінде бұл мидың дәлсіздігіне, кейін есін жоғалтуға және өлімге әкеледі. Уақыт өте қандағы глюкозаның көп болуы көздің зақымдануына, бүйректің және жүректің дұрыс жұмыс істемеуіне алып келуі мүмкін.

Қазіргі кішігірім бергіштерді өңдеу аймағында жетістікке жетуі, инсулинді енгізудің автоматтандырылған жүйесін дамытуға болатындығын көрсетеді. Бұл жүйелер қан құрамындағы қанттың мөлшерін бақылап, керек кезінде инсулиннің

сәйкес мөлшерін береді. Қазір осындай жүйелер ауруханаларда бар. Болашақта көптеген сусамырмен ауыратындарға осындай жүйелерді денесіне жапсыруы мүмкін.



1.4-сурет. Инсулин сорғышының техникалық қамсыздандыруы



1.5-сурет. Инсулин сорғышының әрекеттер үлгісі (моделі)

Бағдарламалық басқарылатын инсулин жүйесі ауруға орналасқан микробергіш арқылы сәйкес қанттың мөлшерін тексере алады. Одан кейін помпа бақылағышына жіберіледі. Бұл құрал қажетті қант мөлшері мен инсулин мөлшерін есептейді. Сосын помпа кішігірім помпаға инсулинді орнына жеткізу үшін белгі береді.

1.4-суретте бағдарламалық қамтыма құраушылары мен инсулин помпасының мекемесі көрсетілген. Осы оқулықтағы мысалдарды түсіну үшін, қан бергіштің қанның әр жағдайдағы электрөткізгіштігін өлшейтінін және бұл өлшемдердің қандағы қанттың мөлшеріне байланысты екенін білу қажет. Басқарғыштың берген импульсіне байланысты помпа инсулині бірлік өлшемінде инсулин жіберіп отырады. Сондықтан 10 бірлік инсулинін жіберу үшін басқарғыш помпаға 10 импульс жібереді. 1.5-суретте UML үлгісі бағадарламалық қамтыманың келген қант

мөлшерін инсулин помпасын басқаратын келесі деңгейге қалай айналдыратынын көрсетеді.

Бұл жүйенің қауіпсіз болуы өте маңызды. Егер помпа жұмыс істемесе немесе дұрыс жұмыс істемесе науқастың денсаулығы нашарлауы мүмкін немесе қанның құрамында глюкозаның көп мөлшерде немесе аз мөлшерде болуынан есінен танып қалуы мүмкін. Бұл жерде осы жүйеге арналған екі маңызды талап бар:

1. Қажет кезінде жүйе инсулинмен қамтамасыз етуге дайын болуға тиіс.
2. Жүйе нық жұмыс істеп, қанның құрамындағы қантты азайту үшін, инсулинді дұрыс мөлшерде жеткізіп отыруы керек.

Сондықтан жүйе осы талаптарды орындайтындай өңделіп, іске асуы керек. Келесі тарауларда жүйенің қауіпсіздігін тексеру туралы толығырақ баяндалады.

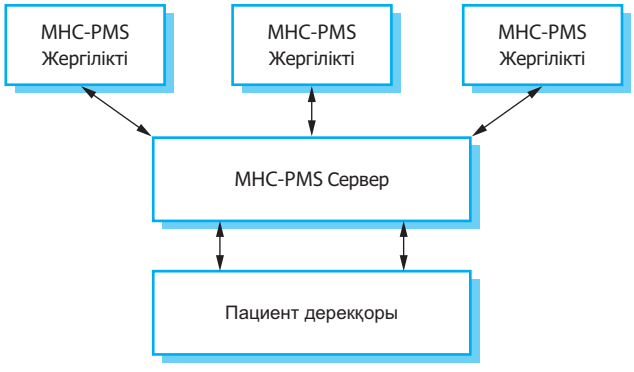
1.3.2. Ақпараттық психикалық жүйе емделушісі

Ақпараттық психикалық жүйе емделушісі – психикалық денсаулығынан зардап шеккен емделушілер туралы деректерді өңдейтін және емдейтін медициналық ақпараттық жүйе. Психикалық денсаулығынан зардап шеккен адамдарға тұрақты ем қабылдаудың қажеті болмағанымен, олар арнайы ауруханаларға барып дәрігермен кездесіп тұруы керек. Емделушілерге ем алу оңай болуы үшін, бұл ауруханаларда жұмыс істеу әдеттегі ауруханалардағыдай жеңіл емес. Олар сол сияқты жергілікті медициналық тәжірибелік жерінде немесе қауымдық жерлерде өткізілуі мүмкін.

Емделушінің психикалық денсаулығын басқаратын жүйенің (ЕПДБЖ) клиникаларға арналған ақпараттық жүйесі бар. Бұл емделушілердің орталықтандырылған деректер базасын қолданғанымен, бұл компьютерде жұмыс істеуге де арналған, сондықтан ол желіге қосылмаған сайттарға да қосыла алады. Жергілікті жүйелер қауіпсіз желіге шыға алатын болса, олар деректер базасында емделуші туралы ақпаратты көріп, оны сақтап, кейін өшіп тұрғанда қолдана алады. Бұл жүйе – толық медициналық жазбаларды сақтайтын жүйе емес, сондықтан бұл жүйе басқа аурулар жайлы ақпаратпен қамтамасыз етпейді. Бірақ ол басқа клиникалық ақпараттық жүйелерімен қатынасып, деректермен алмаса алады. 1.6-суреттен ЕПДБЖ-нің мекемесін көре аласыз.

ЕПДБЖ:

1. Басқару жайлы ақпаратты қалыптастыру, денсаулық басқарушыларына мемлекеттік және жергілікті ұстанымдарға қарамастан тиімділікті дұрыс бағалай білу.
2. Медицина жұмыскерлерін қажетті ақпаратпен қамтамасыз ету сияқты екі мақсатты көздейді



1.6-сурет. МНС-PMS-тің құрылымы

Психикалық денсаулық мәселерінің бірі – ол жауапкершіліктің болмауы, мысалы, олар кездесулерге келмей немесе рецептілері мен дәрі-дәрмектерін жоғалтып, нұсқауларды ұмытып, дәрігерлерге негізсіз талаптар қоюы мүмкін. Олар күтпеген жерден клиникаға келуі де мүмкін. Олар көбінесе, өздеріне және басқаларға да қауіп төндіруі мүмкін. Олар уақытша немесе аз уақытқа мекен-жайларын өзгертуі мүмкін. Қауіпті аурулар қауіпсіз ауруханаларға жатқызылуы қажет.

Жүйенің пайдаланушылары – клиниканың қызметкерлері, дәрігерлер мен медбикелер. Медицина қызметкерлерін тіркейтін, жүйе есепшотын жүргізетін тіркегіштер мен есептерді жинайтын әкімшілік қызметкерлер медициналық пайдаланушылар қатарына жатпайды.

Жүйе емделушілер туралы ақпаратты (аты, мекен-жайы, жасы, жақын туыстары және т.б), кеңестерді (күні, дәрігердің қарауы, аурудың әсері, т.б), жағдайларды және емдеулерді жазу үшін қолданылады. Медицина қызметкерлері үшін қалыпты негізде және денсаулық сақтау мүшелері үшін есеп беру құралады. Көбінесе, медицина қызметкерлері үшін жасалған есеп беру жеке аурулар туралы ақпараттан тұрады, ал әкімшілік есеп беру жасырын түрде және ол жерде болған жағдайлар және емдеуге кеткен шығындар туралы жазылады.

Жүйенің маңызды ерекшеліктері:

1. *Жеке күтуді басқару.* Дәрігерлер емделушілер үшін жазба жасап, жүйедегі ақпаратты өңдеп, емделушінің ауру тарихын көре алады. Жүйе деректерді жақындату функциясымен қамтылған, сондықтан бұрын емдемеген дәрігерлер барлық ауруларын және қабылданған емін біле алады.
2. *Емделушілерді бақылау.* Жүйе емделушілердің аурулар тарихын қарап, егер қиыншылықтарды байқаған жағдайда алдын ала білдіреді. Сондықтан егер емделуші көп уақыт көрінбесе, жүйе ескерту жасайды. Бұл бақылау жүйесінің маңызды бөлігінің бірі – емделушілерге қарау және қажетті уақытта тексерумен қамтамасыз ету.

3. *Әкімшілік есеп беру.* Жүйе басқарудың айлық есебін, яғни әр клиникадағы емделушілер санын және ауруханаларда емделген адамдар санын, емделу бағасын көрсетеді.

Жүйеге екі заңдылық әсер етеді. Бұл – заңдар жеке деректердің құпиялығын сақтайтын, деректерді сақтау заңы және өздеріне және өзгелерге қауіпті бақылау астындағы мәжбүрлі күтімдегі науқастарды басқаратын, денсаулық сақтау ісіндегі психикалық заңдар. Психикалық денсаулық – медицинадағы бірегей мамандық, өйткені емделушінің еркіне қарсы бөгеуді ұсынады. Бұл өте қатаң заңнамалық кепілдікке тәуелді. ЕПДБЖ-ның мақсаттарының бірі – жұмыскерлерінің әрдайым заңға сәйкес жұмысын істеп, қабылдаған шешімдерін соттық қарауға тіркеу.

Барлық медициналық жүйелердегідей жүйенің негізгі талаптарының бірі – ол жеке өмірге араласпау. Емделуші жайлы ақпараттың құпиялы болуы және емделушінің өзінен басқа ешкімге ашылмауы маңызды. ЕПДБЖ – бұл да қауіпсіздікті қажет ететін жүйе. Кейбір психикалық аурулар емделушілердің өзіне-өзі қол жұмсауына немесе басқа адамдарға қауіп төндіретін жағдайға жеткізуі мүмкін. Барлық жағдайда жүйе медицина қызметкерлерін әлеуеті қауіпті емделушілер туралы ескерту керек. Жүйе құпиялықпен және қауіпсіздікпен қамтамасыз етуі тиіс. Жүйе қажет уақытта бос болуы қажет, өйткені емделушілерге уақытында дұрыс дәрі-дәрмек берілуі қажет. Бұл жерде әлеуетті қарсылық туады, себебі жүйе деректерінің бірнеше көшірмелерінің болуы – құпиялықтың бұзылуына алып келуі мүмкін. Бірақ егер желі дұрыс жұмыс істемесе, деректердің көшірмелерінің болғаны дұрыс. Мен бұл талаптардың арасындағы келісімдерді келесі тарауларда талқыладым.

1.3.3. Шөлді ауа райы станциясы

Шалғай орналасқан аймақтардағы ауа райы болжамын туралау үшін және ауа райы өзгерісін бақылау үшін, үкімет шалғай аймақтарға ауа райы бекеттерін тұрғызуға шешім қабылдайды. Бұл ауа райы станциялары температураны және қысымды өлшейтін, күнді, жауын-шашынды, желдің жылдамдығын және желдің бағытын анықтайтын құралдар арқылы деректер жинайды.

Шөлді ауа райы станциясы үлкен жүйенің бір бөлігі (1.7-сурет), ол – ауа райы станцияларынан деректер жинап, басқа жүйелерге өңдеуге қолайлы ететін ауа райы ақпарат жүйесі. 1.7-суреттегі жүйелер:

1. *Ауа райы станция жүйесі.* Бұл ауа райы жайлы деректерді жинауға, кейбір деректерді өңдеуге және деректерді басқару жүйесіне беруге жауап береді.
2. *Деректерді басқару және мұрағаттандырылған жүйе.* Бұл жүйе ауа райы станцияларынан деректер жинап, деректерді өңдеп, талдаумен айналысады және ақпаратты басқа жүйелерге де қолайлы етіп мұрағаттанады.
3. *Станция техникалық қызметі.* Бұл – жүйе серік арқылы жүйелердің күйін сараптайтын және шұғыл мәселелер жайлы хабарлайтын ауа райының

барлық станцияларымен байланыса алады. Ол осы жүйелерге кіріктірілген бағдарламалық қамтыманы өңдей алады. Егер жүйеде қателіктер пайда болатын болса, бұл жүйе сол сияқты алыстатылған басқарумен де жұмыс істей алады.

1.7-сурет. Ауа райы бекетінің қоршаған ортасы

1.7-суретте мен UML кестесі арқылы әрбір жүйенің құрағыштар жиынтығы екендігін көрсететін UML-дегі «жүйе» стереотипін қолдандым.

Әрбір ауа райы станциясында ауа райы параметрлерін өлшейтін құралдар бар, олар – желдің жылдамдығы мен бағытын өлшейтін, жер мен ауа температураларын, атмосфералық қысымды, 24-тәуліктегі жауын-шашын мөлшерін өлшейтін құралдар. Бұл құралдардың арқайсысы бағдарламалық қамтымамен басқарылады, ол әрбір құралдан әрдайым параметрлерді қабылдап, деректерді басқарады.

Ауа райы станция жүйесі аз уақыт аралығына арналған метеорологиялық бақылау деректерін жинаумен айналысады. Бұл жерде температура әр минут сайын өлшенеді. Бірақ серіктің өткізгіштік қасиетінің төмендігінен ауа райы станциясы кейбір жергілікті өңдеумен және деректерді агрегациялаумен айналысады. Одан кейін агрегацияланған деректерді деректер жинау жүйесіне тапсырады. Егер қандай да бір себептермен байланыс ұстай алмаса, ауа райы бекеттері деректерді жергілікті желіде сақтап, байланыс орнағанша тұрады.

Әрбір ауа райы станциясы батареямен жұмыс істейді және толығымен автономиялы болуы, яғни сырттай тоқтың немесе желі кабелінің болмауы керек. Барлық кіші жылдамдықтағы серікпен байланыс арқылы өтетін қатынас-байланыстар және ауа райы бекеттері батареяларын зарядтайтын бір тетікті (жел немесе жел энергиясы) қолдануы тиіс. Шөлді жерлерде таралып, қоршаған ортаның қатаң жағдайларын кездестіріп, жануарлар оны зақымдауы мүмкін. Сондықтан станция бағдарламалық жасақтамасы тек қана деректер жинаумен айналыспайды. Сонымен қатар:

1. Басқару жүйесіне қиыншылықтар туралы есеп беріп және құралдарды сараптау, тоқ және байланыс жабдықтарын тексереді.
2. Жүйені энергетика жүйесімен қамтамасыз ету, батареялардың зарядталуын бақылау немесе ауа райының қолайсыз жағдайында, қатты желден генераторлардың жұмыс істемей қалуына қарамастан жүйені басқарады.

3. Бағдарламалық қамтыманың жаңа нұсқамен толықтырылған бөліктерінде және жүйенің өшіп қалғанда қосылатын резервтік көшіру құралдарына динамикалық реконфигурация жасауды қолдайды.

Сондықтан ауа райы бекеттері автономиялы және қараусыз, яғни күрделі бағдарламалық қамтыманың болғанымен, деректер жинау функциясы жеңіл болуы тиіс.

НЕГІЗГІ ТҰЖЫРЫМДАР

- Бағдарламалық жасақтама – бұл бағдарламалық қамтыманы өндірудің барлық аспектілерімен байланысқан инженерлік пән.
- Бағдарламалық қамтыма тек қана бағдарламалармен ғана емес сол сияқты құжаттамалармен де айналысады. Әдеттегі бағдарламалық қамтыма сипаттарына: жөндеуге жарамдылық, сенімділік, қауіпсіздік, тиімділік және жарамдылық жатады.
- Бағдарламалық қамтыма үдерістері бағдарламаның дамуына байланысты әрекеттермен тығыз байланысты. Жоғарғы сатыдағы: сипаттардың тізімін өңдеу, тексеру және эволюция сияқты қызметтер бағдарламалық жасақтаманың маңызды бөлігі болып табылады.
- Бағдарламалық жасақтама іргелі тұжырымдамалары жүйенің қандай да болмасын түріне лайықты. Бұл негіздер бағдарламалық қамтыма үдерістерін, сенімділікі, қауіпсіздікті, талаптарды және қайта пайдалануды құрайды.
- Бұл жерде жүйенің бірнеше түрі бар және олардың әрқайсысы бағдарламалық қамтыма құралдарын және әдістерін қажет етеді. Барлық жүйелерге тиімді бірнеше нақты өңдеулер мен әдістер бар.
- Бағдарламалық жасақтамасының іргелі нысаналары бағдарламалық қамтыма жүйелерінің барлық түріне сәйкес келеді. Бұл іргетастар басқарылатын бағдарлама үдерістерінен, бағдарлама сенімділігінен және қауіпсіздігінен, өңдеу талаптарынан және қайта пайдаланудан тұрады.
- Бағдарлама жобалаушылары инженерлік мамандыққа және қауымға жауапты. Олар тек қана техникалық мәселермен айналыспау керек.
- Кәсіби қауымдар өз мүшелеріне арналған тәртіп кодексін шығарады.

ҚОСЫМША ӘДЕБИЕТТЕР

‘No silver bullet: Essence and accidents of software engineering’. In spite of its age, this paper is a good general introduction to the problems of software engineering. The essential message of the paper still hasn’t changed. (F. P. Brooks, *IEEE Computer*, **20** (4), April 1987.) http://doi.ieeecomputersociety.org/10.1109/MC.1987.1663532.

‘Software engineering code of ethics is approved’. An article that discusses the background to the development of the ACM/IEEE Code of Ethics and that includes both the short and

long form of the code. (*Comm.* ACM, D. Gotterbarn, K. Miller, and S. Rogerson, October 1999.) http://portal.acm.org/citation.cfm?doid=317665.317682.

Professional Issues in Software Engineering. This is an excellent book discussing legal and professional issues as well as ethics. I prefer its practical approach to more theoretical texts on ethics. (F. Bott, A. Coleman, J. Eaton and D. Rowland, 3rd edition, 2000, Taylor and Francis.)

IEEE Software, March/April 2002. This is a special issue of the magazine devoted to the development of Web-based software. This area has changed very quickly so some articles are a little dated but most are still relevant. (*IEEE Software,* **19** (2), 2002.) http://www2.computer.org/portal/web/software.

'A View of 20th and 21st Century Software Engineering'. A backward and forward look at software engineering from one of the first and most distinguished software engineers. Barry Boehm identifies timeless software engineering principles but also suggests that some commonly used practices are obsolete. (B. Boehm, *Proc. 28th Software Engineering Conf.,* Shanghai. 2006.) http://doi.ieeecomputersociety.org/10.1145/1134285.1134288.

'Software Engineering Ethics'. Special issue of IEEE Computer, with a number of papers on the topic. (*IEEE Computer,* **42** (6), June 2009.)

ЖАТТЫҒУЛАР

1.1. Неге кәсіби бағдарламалық қамтыма сатып алушыға арналған жай ғана бағдарлама емес, түсіндіріңіз.

1.2. Жалпы бағдарламаның дамуы мен сатып алушыға арналған бағдарламалық қамтыманың басты айырмашылығы неде? Жалпы бағдарлама пайдаланушыларына қолданғанда бұл нені білдіруі мүмкін?

1.3. Барлық кәсіби бағдарламалық қамтымада болатын негізгі сипаттарды, кейде қолданылатын басқа төрт сипаттарды атаңыз.

1.4. Гетерогендік, қауымдық және іскерлік, сенімділік және қауіпсіздік сияқты XXI ғасырда бағдарламалық қамтыманы жобалауда қандай қиындықтарды (қоршаған орта туралы) кездестіруге болады?

1.5. *1.1.2-бөлімде* баяндалған қолданбаларды өз біліміңізді қолданып, неліктен қолданбаларды жобалау үшін бағдарламалық қамтыманың арнайы тәсілдерді қолданылатынын мысалдар келтіріп түсіндіріңіз.

1.6. Барлық қамтыма жүйелеріне қолданылатын Бағдарламалық жасақтамасының іргелі нысаналарының болуын түсіндіріңіз.

1.7.Вебтің жан-жақты қолданғаннан бағдарлама жүйелерінің қалай өзгергенін түсіндіріңіз.

1.8. Кәсіби инженерлер дәрігерлер немесе заңгерлер секілді сертификатталған болуы керек, осыны талқылаңыз.

1.9. *1.3-суретте* ACM / IEEE кодексі этикасы көрсетілген, осы жағдайды көрсететін мысал келтіріңіз.

1.10. Терроризмге қарсы тұру үшін көптеген мемлекеттер әрбір азаматтың іс-әрекетін бақылайтын компьютерлік жүйе өңдеп шығарды. Әрине бұл құпиялы түрде сақталады. Осындай жүйенің даму этикасы туралы талқылаңыз.

ПАЙДАЛАНЫЛҒАН ӘДЕБИЕТТЕР

Gotterbarn, D., Miller, K. and Rogerson, S. (1999). Software Engineering Code of Ethics is Approved. Comm. ACM, **42** (10), 102–7.

Holdener, A. T. (2008). Ajax: *The Definitive Guide*. Sebastopol, Ca.: O'Reilly and Associates.

Huff, C. and Martin, C. D. (1995). Computing Consequences: A Framework for Teaching Ethical Computing. Comm. ACM, **38** (12), 75–84.

Johnson, D. G. (2001). *Computer Ethics*. Englewood Cliffs, NJ: Prentice Hall.

Laudon, K. (1995). Ethical Concepts and Information Technology. Comm. ACM, **38** (12), 33–9.

Naur, P. and Randell, B. (1969). Software Engineering: Report on a Conference sponsored by the NATO Science Committee, Garmisch, Germany. 7th to 11th October 1968.

2.

Бағдарламалық қамтамасыз ету үдерістері

Мақсаттары

Осы тараудың мақсаты – Сізге бағдарламалық қамтамасыз ету идеясын, яғни бағдарламалық қамтамасыз ету өндірісі үшін қызмет түрлерінің жүйелі жиынтығын ұсыну. Осы тарауды оқу барысында Сіз:

- бағдарламалық қамтамасыз ету үдерістері ұғымын және бағдарламалық қамтамасыз ету үдерісінің үлгілерін түсінесіз;
- бағдарламалық қамтамасыз ету үдерісінің үш жалпы үлгісімен және оның қолданылуымен танысасыз;
- бағдарламалық қамтамасыз етуге қойылатын негізгі техникалық талаптар, бағдарламалық қамтамасыз етуді құрастыру, тестілеу және дамыту туралы білесіз;
- үдерістердің бағдарламалық қамтамасыз ету талаптары мен дизайнындағы өзгерістерді орындайтындай болуы керектігін түсінесіз;
- бағдарламалық қамтамасыз етудің ыңғайлы үдерістерін жасау үшін Оңтайлы Біріктірілген Үдеріс техникалық қамтамасыз етудің жақсы тәжірибесін қалай интегралдайтынын түсінесіз;

Мазмұны

Бағдарламалық қамтамасыз ету үдерісі – бұл бағдарламалық қамтамасыз етуді жүзеге асыратын байланысқан әрекеттер қатары. Бұл әрекеттерге бағдарламалық қамтамасыз етудің бастапқы, Ява немесе C тәрізді бағдарламалаудың стандартты тілінде дамуы жатқызылуы мүмкін. Алайда, бизнес-қосымшаларды мұндай тәсілмен дамыту міндетті емес. Бизнеске арналған жаңа бағдарламалық қамтамасыз ету әдетте қолданыстағы жүйені кеңейту немесе өзгерту жолымен немесе стандартты бағдарламалық қамтамасыз етуді, жүйе компоненттерін құру және біріктіру жолымен дамиды.

Бағдарламалық қамтамасыз етудің көптеген үдерістері бар, бірақ олардың барлығы бағдарламалау үшін негіз болып табылатын төрт әрекеттен тұруы қажет:

1. *Бағдарламалық қамтамасыз етудің спецификациясы.* Бағдарламалық қамтамасыз етудің функционалдық мүмкіндіктері мен қолдану жөніндегі шектеулері анықталуы тиіс.
2. *Бағдарламалық қамтамасыз етуді енгізу және жобалау.* Бағдарламалық қамтамасыз ету спецификацияға сәйкес өндірілуі қажет.
3. *Бағдарламалық қамтамасыз етудің ұйғарымдылығын тексеру.* Бағдарламалық қамтамасыз ету клиенттің барлық қойған талаптарына сәйкес болуы қажет.
4. *Бағдарламалық қамтамасыз етудің дамуы.* Бағдарламалық қамтамасыз ету тұтынушылық талап өзгерістеріне сәйкес құрастырылуы қажет.

Белгілі бір нысанда бұл әрекеттер бағдарламалық қамтамасыз етудің барлық үдерістерінің бір бөлігі блып табылады. Іс жүзінде бұл ұйғарымдылықты тексеруге талаптар, сәулеттік дизайн, бірлікті тестілеу және т.б. әрекеттерден тұратын күрделі әрекет. Сондай-ақ, БҚЕ үйлестіруді құжаттау және басқару тәрізді үдерістің қолдамалы әрекеттері бар.

Біз үдерістерді сипаттап, талқылаған кезде әдетте сол үдерістердегі әрекеттер туралы айтамыз, олар: мәліметтер үлгісін анықтау, қолданбалы интерфейсті құрастыру, т.б. және осы әрекеттерді реттеу. Алайда, әрекеттер тәрізді, үдерістің сипаттамасы келесілерден тұруы мүмкін:

1. Үдеріс қызметінің нәтижесі болып табылатын өнімдер. Мысалы, сәулеттік дизайн қызметінің нәтижесі бағдарламалық қамтамасыз ету сәулет үлгісі болуы мүмкін.
2. Үдеріске қатысушы адамдардың міндеттерін айқындайтын рөлдер. Рөлдердің мысалы – жобалық менеджер, үйлесім менеджері, бағдарламашы және т.б.
3. Жұмыстың алдыңғы және кейінгі орындалуының кепілі болып табылатын алдыңғы және кейінгі шарттар. Мысалы, сәулеттік дизайнды құрастырардан бұрын алдын ала қойылатын шарт – бұл барлық талаптардың клиентпен мақұлдануы. Осы қызмет аяқталған соң кейінгі шарт – сәулетті сипаттайтын UML үлгісі орындалады.

Бағдарламалық қамтамасыз ету үдерістері күрделі және барлық зияткерлік және шығармашылық үдерістер тәрізді шешімдер және пікірлер шығаратын адамдарға сүйенеді. Мінсіз үдеріс болмайды және ұйымдардың көпшілігі бағдарламалық қамтамасыз етуді жасаудың өзіндік үдерістерін ойлап тапты. Үдерістер өз мүддесі үшін құрастырылатын жүйе ерекшеліктерін ұйымдастыру және анықтауда адамдардың қабілеттіліктерін пайдаланатындай етіп дамыды. Критикалық жүйелер сияқты, кейбір жүйелер үшін өңдеудің құрылымдық үдерісі талап етіледі. Талаптары жедел өзгеретін бизнес-жүйелер үшін аса ресми емес, икемді үдеріс тиімдірек болуы мүмкін.

Кейде бағдарламалық үдерістер жоспарлы басқарылатын немесе жедел және икемді болып санаттандырылады. Жоспарлы басқарылатын үдерістер – бұл барлық әрекеттер алдын ала жоспарланған және үдеу өлшенетін үдерістер. 3-тарауда қарастырылатын жедел және икемді үдерістерде жоспарлау кезең-кезеңмен жүреді және өзгермелі тұтынушылық талаптарға жауап беру үшін үдерісті өзгерткен жеңілдеу. Боем мен Тернердің пікірінше (2003), бағдарламалық қамтамасыз етудің әр түрлі типі үшін әрбір амал жарамды. Әдетте басқармалы жоспар мен жедел және икемді үдерістердің арасында теңгерім анықталуы қажет.

Мінсіз бағдарламалық қамтамасыз ету болмаса да, көптеген ұйымдарда бағдарламалық үдерісті жақсарту мүмкіндіктері бар. Үдерістер көнерген әдістерден тұруы мүмкін немесе бағдарламалық қамтамасыз етуді өнеркәсіптік құрастыруда өз мүддесі үшін тиімді тәжірибені пайдаланбауы мүмкін. Шын мәнінде де ұйымдардың көпшілігі бағдарламалық қамтамасыз етудің құрастыру әдістерін қолданбайды.

Бағдарламалық үдерістер үдерістің стандартизациясымен жақсартылуы мүмкін, мұнда ұйымдастыру бойынша бағдарламалық үдерістердегі әр түрлілік азайған. Бұл байланыстың жақсаруына және оқыту уақытының қысқаруына әкеледі, үдерістің автоматтандырылған сүйемелдеуін үнемді етеді. Сондай-ақ, стандартизация бағдарламалық қамтамасыз ету тәжірибесін құрастырудың жаңа әдісін ұсынудағы бірінші маңызды қадам болып табылады. Бағдарламалық қамтамасыз етуді жақсарту мәселесі 26-тарауда толығырақ талқыланады.

2.1 Бағдарламалық үдерістің үлгілері

1-тарауда айтып өткендей, бағдарламлық үдерістің үлгісі – бұл бағдарламалық үдерістің жеңілдетілген көрінісі. Үдерістің әрбір үлгісі үдерісті белгілі бір перспективадан ұсынады, осылайша бұл үдеріс туралы тек жарым-жартылай ақпарат ұсынады. Мысалы, үдерістің әрекет үлгісі әрекеттер мен олардың бірізділігін көрсетеді, бірақ сондай-ақ бұл әрекеттерге қатысушы адамдардың рөлдерін көрсетпеуі мүмкін. Осы бөлімде мен үдерістердің көптеген жалпы үлгілерін ұсынамын (кейде «үдеріс парадигмасы» деп аталады) және оларды сәулеттік көзқарас тұрғысынан көрсетемін. Яғни біз үдерістің платформасын көреміз, ал белгілі бір әрекеттердің бөлшектерін байқамаймыз.

Бұл әмбебап үлгілер бағдарламалық үдерістердің түпкілікті сипаттамасы емес. Олар бағдарламалық қамтамасыз етуді құрастырудың әр түрлі жолдарын түсіндіретін үдерістің абстракциялары. Сіз оларды бағдарламалық қамтамасыз етуді құрастырудың анықталған үдерістерін жасау үшін кеңейтіліп, икемделген үдеріс платформалары ретінде қабылдай аласыз.

Аталып өтетін үдерістердің моделдері:

1. *Сарқырама үлгі.* Детализация, құрастыру, жетімділігін тексеру және даму үдерістерінің негізгі әрекеттерінен тұрады және оларды талаптардың ерекшеліктері, бағдарламалық қамтамасыз етуді жобалау, жүзеге асыру, тестілеу және т.б. сияқты үдерістің жеке фазалары ретінде ұсынады.
2. *Сатылық құрастыру*. Бұл тәсіл детализация, құрастыру және шектік тексеру әрекеттерін кезектестіреді. Жүйе нұсқалар (сатылар) сериясы ретінде құрастырылған, әр нұсқа алдыңғы нұсқаға функционалдылық қосады.
3. *Бағдарламалық қамтамасыз етуді қайта құрастыру*. Бұл тәсіл компоненттерді көп мәрте қайта қолдануға негізделген. Құрастырудың жүйелік үдерісі бұл компоненттерді қайта құрастырудың орнына, жүйеге интегралдауға бағытталған.

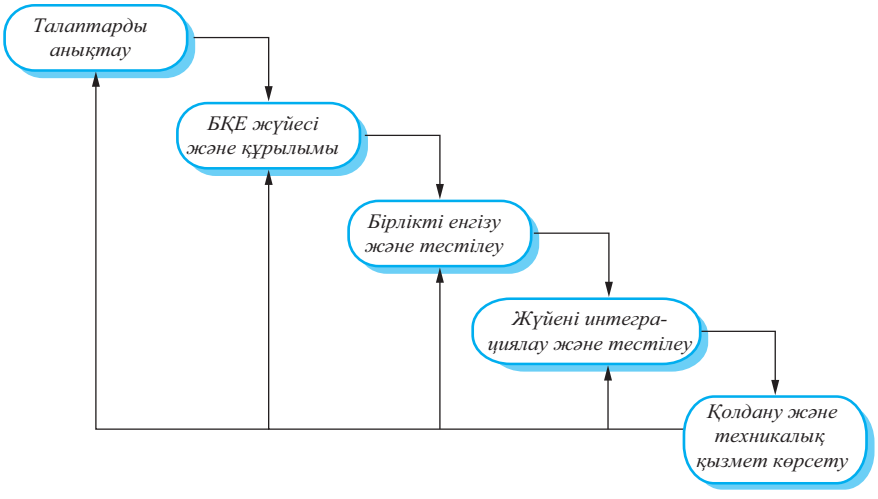
Бұл үлгілер бір-бірін өзара алмастырмайды және көбінесе, әсіресе құрастырудың ірі жүйелерінде бірге қолданады. Ірі жүйелер үшін сарқырама мен сатылық құрастыру үлгілерінің ең жақсы ерекшеліктерін үйлестірген мақсатты. Осы талаптарға сай болу үшін сізде сәулеттік бағдарламалық қамтамасыз етуді құрастыруға арналған мәнді жүйелік талаптар туралы ақпарат болуы тиіс. Сіз оны сатылай құрастыра алмайсыз. Үлкен жүйедегі жеке жүйелер әр түрлі тәсілдермен құрастырылуы мүмкін. Түсінікті болған жүйе бөліктері сарқырамаға негізделген үдерістің көмегімен анықталып, құрастырылуы мүмкін. Алдын ала анықтауға қиын болатын жүйе бөліктерін, мысалы, қолданбалы интерфейсті сатылық тәсілді қолдана отырып құрастыру қажет.

2.1.1. Сарқырама үлгі

Бағдарламалық қамтамасыз етуді құрастыру үдерісінің алғашқы жарияланған үлгісі жүйелік жобалаудың жалпы үдерістерінен алынған болатын (Ройс, 1970). Бұл үлгі 2.1-суретте келтірілген. Бір фазадан екіншіге ауыспалылығынан бұл үлгі «сарқырама үлгі» немесе бағдарламалық қамтамасыз етудің тіршілік циклі атауымен белгілі. Сарқырама үлгі – бұл жоспарлы басқарылатын үдерістің мысалы, яғни Сіз жұмыс алдында үдерістің барлық әрекеттерін жоспарлап, жазуыңыз қажет.

Сарқырама үлгінің негізгі сатылары тікелей негізгі құрылымдарды көрсетеді:

1. *Талаптардың талдауы мен анықтамасы.* Қызметтік жүйелер, шектеулер мен мақсаттар жүйелік тұтынушылармен кеңесу арқылы бекітілген. Мұндайда олар егжей-тегжейлі анықталып, жүйелік спецификация қызметін атқарады.



2.1-сурет. Сарқырама үлгі

2. *Жүйе және бағдарламалық қамтамасыз етуді жобалау*. Жүйені жобалау үдерісі жүйенің толық сәулетін орната отырып, аппараттық құралдарға немесе бағдарламалық қамтамасыз ету жүйесіне талаптар қояды. Бағдарламалық қамтамасыз етуді жобалауға бағдарламалық қамтамасыз етудің негізгі жүйелік абстракциялары мен олардың қатынастарын сәйкестендіреді және сипаттайды.
3. *Бірлікті енгізу және тестілеу.* Осы саты барысында бағдарламалық қамтамасыз етудің жобалануы бағдарламалар қатары немесе бағдарламалық бөлімшелер ретінде жүзеге асады. Бірлікті тестілеуге әрбір модульдің өз спецификациясына жауап беруін тексеру жатады.
4. *Жүйені енгізу және тестілеу*. Бағдарламаның жеке бірліктері немесе бағдарламалар бағдарламалық қамтамасыз етуге қойылған талаптардың қанағаттандырылуын кепілдендіру үшін толыққанды жүйе ретінде интегралданып, тестіленген. Тестілеуден кейін бағдарламалық қамтамасыз етудің жүйесі клиентке жеткізілген.
5. *Жұмысы және қызмет көрсету.* Әдетте бұл тіршілік циклінің ең ұзақ фазасы (міндетті болмаса да). Жүйе практикалық қолдану үшін орнатылған. Қызмет көрсетуге тіршілік циклінің бастапқы сатыларында анықталмаған қателіктерді түзету жатады, бұл жүйелік модульдердің енгізілуін және жүйелік қызметтерді жаңа талаптар ретінде жақсартады.

Түптеп келгенде, әр фазаның нәтижесі – бұл құпталып, қол қойылған бір немесе одан артық құжат. Келесі фаза алдыңғысы аяқталмай басталмауы тиіс. Іс жүзінде бұл сатылар жарым-жартылай сәйкес келеді. Жоба барысында кейде талаптарға қатысты мәселелер туындайды. Кодтау кезінде жобалаумен мәселелер туындайды және т.б. Бағдарламалық үдеріс – бұл фазалар арасындағы кері байланысты қарастырмайтын қарапайым желілік үлгі. Әр фазада келтірілген құжаттар енгізілген түзетулерді көрсету үшін кейін өзгертілуі мүмкін.

Cleanroom бағдарламалық қамтамасыз етуін құрастыру

Cleanroom үдерісі бастапқыда жасалған IBM-нің формальды жасалған үдерісінің мысалы болып табылады. Cleanroom үдерісінде бағдарламалық қамтамасыз етудің әр сатысы формальды анықталған, және бұл спецификация енгізуге түрленген. Бағдарламалық қамтамасыз етуді түзету формальды жасалған. Үдерістегі ақауларды тестілу жасалмаған, жүйені тестілеу жүйенің беріктігін бағалауға бағытталған.

Cleanroom үдерісінің мақсаты – сенімділіктің жоғары деңгейіне арналған ақаусыз бағдарламалық қамтамасыз ету.

http://www.SoftwareEngineering-9.com/Web/Cleanroom/

Құжаттарды әзірлеп, мақұлдауға кеткен шығынға байланысты итерациялар қымбат болуы мүмкін және мәнді толықтыруларды қажет етеді. Сондықтан, аздаған мөлшердегі итерациядан кейін спецификация сияқты құрылымның бөліктерін тоқтатып, кейінгі сатыларды жалғастыру қалыпты жағдай болып табылады. Мәселелер кейінге қалдырылады, ескерілмейді немесе бағдарламаланады. Бұл уақытынан бұрын талаптарды тоқтату жүйенің тұтынушы тілектерін орындамайтынын білдіреді. Бұл сондай-ақ нашар құрылған жүйелерге әкеліп соқтырады, себебі жобалау мәселелері енгізу тәсілдері бойынша жоспарларды бұзады.

Тіршілік циклының қорытынды фазасында (жұмыс және қызмет көрсету) бағдарламалық қамтамасыз етуді қолдану мақұлданады. Бағдарламалық қамтамасыз етудің бастапқы талаптарындағы қателіктер мен кемшіліктер анықталады. Егер бағдарлама мен жоба қателіктері анықталса, жаңа функционалдыққа деген мұқтаждық артады. Демек, жүйе пайдалы болып қалуы үшін дамуы тиіс. Бұл өзгерістерді жасау (бағдарламалық қамтамасыз етуді қолдау) үдерістің алдыңғы сатыларының қайталануынан тұруы мүмкін.

Сарқырама үлгі технологиялық үдерістің басқа үлгілерімен үйлеседі және құжаттамасы әр фазада жасалған.

Бұл үдерісті көрнекі етеді, осылайша менеджерлер ілгерілікті құрастыру жоспарымен бақылай алады. Оның басты проблемасы – жобаны мәндес сатыларға бөлуінде. Міндеттемелер өзгермелі тұтынушылық талаптарға жауап беруге кедергі болатын үдерістің бастапқы кезеңінде қабылдануы қажет.

Түптеп келгенде, сарқырама үлгісі тек талаптар толыққанды түсінікті болып, жүйелік құрастыру кезінде радикалды өзгерістер пайда болмайтын кезде қолданылуы керек. Алайда, сарқырама үлгі басқа техникалық жобаларда қолданылатын үдеріс типін көрсетеді. Мұндайда жобаны басқару үшін жалпы үлгіні қолданған жеңіл, әдетте сарқырама үлгіге негізделген бағдарламалық үдерістер әлі де қолданылуда.

Сарқырама үлгінің маңызды нұсқасы – бұл жүйелік спецификацияның математикалық үлгісі жасалатын формальды жүйелік құрылыс. Бұл үлгі нәтижесінде

орындалатын математикалық түрленулерді қолдана отырып жетілдіріледі. Сіздің математикалық түрлендірулеріңіз дұрыс болады деген болжамыңыздың негізінде, мұндай тәсілмен жасалған бағдарлама оның спецификациясына сәйкес келетінін дәлелдей аласыз.

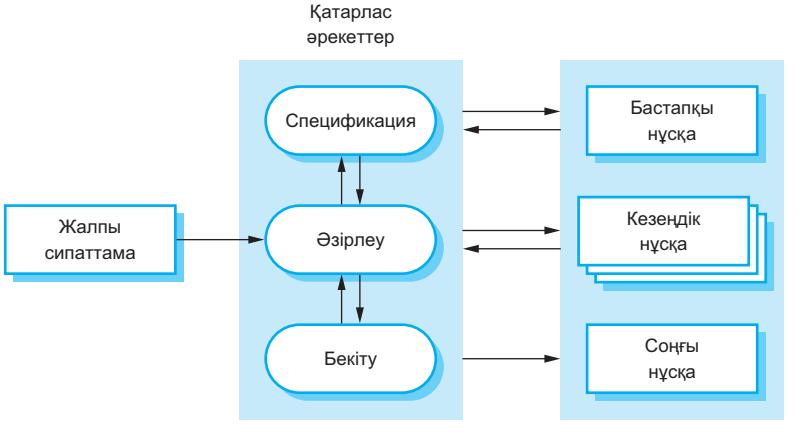
В әдісіне негізделген құрылымның формальды үдерістері (Шнейдер, 2001; Вордсворт, 1996) әсіресе қатаң қауіпсіздігі, беріктігі және қорғауға арналған белгілі бір талаптары бар жүйелерді құрастыруға жарамды. Формальды әдіс қауіпсіздікті жеңілдетеді. Бұл клиенттерге немесе реттеушілерге жүйе іс жүзінде техникалық қауіпсіздік немесе қорғау талаптарын сақтайтынын көрсетеді.

Формальды түрленуге негізделген үдерістер әдетте тек қауіпсіздікке қатысты сыни ережелерді жасауда қолданылды. Олар мамандандырылған білім мен тәжірибені талап етеді.

2.1.2. Сатылық құрастыру

Сатылық құрастыру – бұл бастапқы жүзеге асыруды құрастыру, тұтынушының қарауына ұсыну (яғни оның пікірі), және оның бірнеше нұсқадан кейін дамуы, мұның барлығы тиісті жүйе құрылғанға дейін қайталанады (2.2-сурет). Спецификация, құрастыру және шекті тексеру әрекеттер арқылы жедел кері байланыспен бөлінбей, алмасады.

Бизнес, электронды сауда және дербес жүйелердің көпшілігі үшін сарқырамалық тәсілден басқа, жедел және икемді тәсілдердің негізгі бөлігі болатын бағдарламалық қамтамасыз етудің инкременттік құрылымы тиімдірек болады. Сатылық құрастыру мәселелерді шешуге қолданатын тәсілді көрсетеді. Біз жалпы мәселені сирек шешеміз, бірақ қателік жасағанда оны шешу кері тәртіпте үлкен қадамдар жасаймыз. Біртіндеп бағдарламалық қамтамасыз етуді жасағанда өзгерістер енгізу үшін арзан және жеңіл шешімдер іздейміз.



2.2-сурет. Сатылық құрастыру

Сатылық құрастырумен байланысты мәселелер

Сатылық құрастырудың көптеген артықшылықтарының болуына қарамастан, проблемалары да туындайды. Мәселенің негізгі себебі – бұл ірі ұйымдардың ұзық уақыт бойы дамыған бюрократиялық тәртібінің болуында, сондықтан осы тәртіптер мен бейресми итеративті немесе жедел және икемді үдеріс арасында сәйкессіздік болуы мүмкін.

Кейде бұл тәртіптердің маңызды негіздері болады – мысалы, бағдарламалық қамтамасыз етудің сыртқы нұсқаулықтарды тиісті түрде жүзеге асыратынына берілетін кепілдер үшін арналған тәртіптер болады (мысалы, АҚШ-та, Sarbanes-Oxley нұсқаулықтар есебі). Бұл тәртіпті өзгерту мүмкін емес болып есептеледі, осылайша үдеріс қақтығыстары орын алады.

http://www.SoftwareEngineering-9.com/Web/IncrementalDev/

Жүйенің әрбір сатысы немесе нұсқасы клиентке қажетті белгілі бір функционалдықты жинақтайды. Әдетте бастапқы жүйелік сатылар ең маңызды немесе жедел талап етілетін функционалдықтан тұрады. Бұл клиенттің талаптарына жауап беретінін байқау үшін жүйені салыстырмалы ерте сатыда бағалай алатынын білдіреді. Ал кері жағдайда ағымдық инкремент өзгертіліп, кейінгі инкременттердің жаңа функциялары анықталуы мүмкін.

Сарқырама үлгімен салыстырғанда сатылық құрастырудың үш маңызды артықшылығы бар:

1. Өзгермелі тұтынушылық талаптардың құралының құны азаюда. Өзгертуге жататын талдау мен құжаттаманың сомасы сарқырама үлгісіне талап етілетін сомадан әлдеқайда аз.
2. Орындалған техникалық құрастыру бойынша тұтынушымен кері байланыс орнату жеңіл. Клиенттер бағдарламалық қамтамасыз ету жайында пікірлерін білдіріп, жұмыс нәтижесін көре алады. Клиенттер бағдарламалық қамтамасыз етуді жобалау құжаттары бойынша ілгерілікті бағалау қиын деп есептейді.
3. Барлық қызметтер қосылмаса да, пайдалы бағдарламалық қамтамасыз етуді жеткізу және орнату жылдам жүреді. Сарқырама үлгісімен салыстырғанда клиенттердің бағдарламлық қамтамасыз етуден пайда көріп, қолдануға мүмкіндіктері артығырақ.

Белгілі бір нысандағы сатылық құрастыру қолданбалы жүйелердің қолданылуына арналған кең тараған тәсіл болып табылады. Бұл тәсіл жоспарлы басқарылатын, жедел және икемді болады немесе көбінде осы тәсілдердің жиынтығы болады. Жоспарлы басқарылатын тәсілде жүйелік инкременттер алдын ала сәйкестендірілген; егер жедел және икемді тәсіл қолданса, бастапқы инкре-

менттер сәйкестендіріледі, бірақ кейінгі инкременттердің құрылысы тұтынушылық басымдылық пен ілгерілікке байланысты.

Басқару көзқарасы бойынша сатылы тәсілдің екі мәселесі бар:

1. Үдеріс көзге көрінбейді. Менеджерлер ілгерілікті өлшеу үшін үнемі соңғы нәтижені талап етеді. Егер жүйелер жылдам құрастырылса, жүйенің әр нұсқасын көрсететін құжаттарды ұсыну экономикалық тұрғыдан тиімсіз.
2. Жаңа инкременттер қосылғандықтан, құрылымдық жүйе нашарлау үдерісіне душар болады. Егер уақыт пен қаражат бағдарламалық қамтамасыз етуді жақсартуға арналған рефакторингке жұмсалса, тұрақты өзгеріс өз құрылымын бұзу үдерісіне ие болады. Бағдарламалық қамтамасыз ету өзгерістерінің әрі қарай қосылуы барған сайын қиындап, қымбаттайды.

Сатылық құрастыру мәселелері әр түрлі командалар жүйенің әр түрлі бөліктерін жасайтын ірі, күрделі, ұзақ мерзімді жүйелері үшін ерекше маңызды болып келеді. Ірі жүйелер тұрақты платформа немесе сәулетті талап етеді және сол сәулетке қатысты нақты анықталуы қажет. Ол алдын ала жоспарлануы тиіс және инкрементті құрастырылмайды.

Сіз іс жүзінде клиент ортасына жеткізбей және орнатпай, жүйені сатылы құрастырып, пікірлесу үшін клиенттерге ұсына аласыз. Инкременттік жеткізу және орнату бағдарламалық қамтамасыз етудің шынайы, операциялық үдерістерде қолданылатынын білдіреді. Бұл үнемі мүмкін болмайды, себебі жаңа бағдарламалық қамтамасыз етумен жасалатын тәжірибелер қалыпты бизнес-үдерістердің бұзылуына соқтыруы мүмкін. Инкременттік жеткізудің артықшылықтары мен кемшіліктері 2.3.2-бөлімде талқыланады.

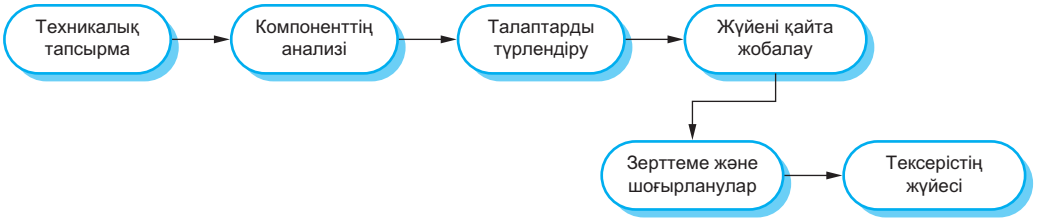
2.1.3. Қайта пайдалануға бағытталған бағдарламалық қамтамасыз етуді құрастыру

Бағдарламалық қамтамасыз ету жобаларының көпшілігінде оны қайта қолдану бар. Бұл көбінде бейресми түрде, жобамен жұмыс істейтін адамдар талап етілетіндерге ұқсайтын жобалар мен кодтар туралы білген жағдайда орын алады. Олар ізденіп, қажет болған жағдайда өзгертіп, өз жүйелеріне енгізеді.

Бұл бейресми қайта қолдану құрастыру үдерісіне тәуелсіз жүзеге асады. Алайда XXI ғасырда қолданыстағы бағдарламалық қамтамасыз етуді қайта қолдануға бағытталған бағдарламалық қамтамасыз етуді құрастыру үдерістері кең етек жайған. Қайта қолдануға бағдарланған тәсілдер осы компоненттер құрамы үшін бағдарламалық қамтамасыз ету мен интегралданған платформаны қайта қолдануға жол беретін негізге сүйенеді.

Кейде бұл компоненттер мәтінді өңдеу немесе электрондық кесте тәрізді белгілі бір қызметті жүзеге асыратын дербес жүйе бола алады (COTS немесе коммерциялық сериялық шығарылатын құрылғы).

Құрастыруды қайта қолдануға негізделген үдерістің жалпы үлгісі 2.3-суретте көрсетілген.



2.3-сурет. Бағдарламалық қамтамасыз етуді қайта қолдануға бағытталған құрастыру.

Техникалық тапсырманың бастапқы сатысы мен шекті тексеру сатысы басқа бағдарламалық үдерістермен салыстырмалы болғанымен, үдерісті қайта қолдануға бағдарланған аралық сатылар өзгеше болады. Бұл сатылар:

1. *Құраушы талдау*. Спецификация талаптарының есебімен, іздеу осы спецификацияны жүзеге асыру үшін компоненттер бойынша орындалады. Әдетте тура сәйкестік болмай, кейбір қызметтер үшін қолданылатын компоненттер бар.
2. *Модификацияға қойылатын талаптар*. Бұл саты барысында талаптар, анықталған компоненттер туралы ақпаратты қолдана отырып, талданады. Кейін олар қолжетімді компоненттерді көрсететіндей болып өзгертіледі. Модификация мүмкіндігі болмаған жағдайда, баламалы шешім табу үшін құрамдас талдамалы әрекет қайта енгізілуі мүмкін.
3. *Қайта қолдана отырып жүйені жобалау*. Бұл фаза барысында жүйе платформасы құрастырылады немесе қолданыстағы платформа қайта пайдаланылады. Құрастырушалыр қайта қолданылатын компоненттерді назарға алып, тиісті қамтамасыз ету үшін платформаны ұйымдастырады. Егер қайта қолданылатын компоненттер қолжетімсіз болса, жаңа бағдарламалық қамтамасыз етуді құрастыру қажет болады.
4. *Құрастыру және интеграция*. Сырттай жабдықталмайтын бағдарламалық қамтамасыз ету құрастырылған және COTS компоненттері мен жүйелері жаңа жүйе жасау үшін интегралданған. Бұл үлгідегі жүйелік интеграция жеке әрекеттің емес, құрастыру үдерісінің бір бөлігі болады.

Қайта қолдануға бағдарланған үдерісте пайдалануға болатын бағдарламалық қамтамасыз ету компонентінің үш типі бар:

1. Сервис стандарттарына сәйкес құрастырылған және алыстағы шақырту үшін қолжетімді Веб-сервистер.
2. NET немесе J2EE тәрізді құрамдас платформамен бірге интегралданған, пакет түрінде құрастырылған нысандар жиынтығы.
3. Белгілі бір ортада пайдалануға жасақталған бағдарламалық қамтамасыз етудің автономды жүйелері.

БҚЕ қайта қолдануға бағдарланған құрастырудың өз артықшылығы бар, нақ осы жағдайда бұл бағдарламалық қамтамасыз етудің орта санының қысқаруы, осылайша шығындар мен тәуекелдің азаюы. Бұл әдетте бағдарламалық қамтамасыз етуді жылдам ұсынуға мүмкіндік береді. Алайда ымыралы талаптар орын алып, бұл тұтынушы талаптарына жауап бермейтін жүйенің жасалуына әкеледі. Сондай-ақ, жүйелік дамудың үстінен бақылау жасалмайды, себебі компоненттерді қайта қолдануға жол беретін жаңа нұсқалар оны қолданатын ұйымдардың басқару нысаны болмайды.

Бағдарламалық қамтамасыз етуді қайта қолдану өте маңызды, сол себепті мен бұл тақырыпқа осы кітаптың үшінші бөлімдегі бірнеше тарауды арнадым. Бағдарламалық қамтамасыз етуді қайта қолданудың жалпы сұрақтары мен COTS қайта қолдану 16-тарауда, бағдарламалық қамтамасыз етуді компонентті-бағдарлы құрастыру 17 және 18-тарауларда және кең шеңберге қызмет көрсету жүйелері 19-тарауда қарастырылған.

2.2. Үдеріс әрекеттері

Шынайы бағдарламалық үдерістер бағдарламалық қамтамасыз ету жүйесін анықтау, құрастыру және тестілеу мақсатында техникалық, бірлескен және ұйымдастырушылық әрекеттер бірізділігімен алмасады. Бағдарламалық қамтамасыз етуді құрастырушылар өз жұмысында түрлі бағдарламалық аспаптар қолданады. Аспаптар әсіресе түрлі типтегі құжаттарды түзетуге және бағдарламалық қамтамасыз етудің үлкен жобасында жасалған үлкен көлемдегі ақпаратты басқаруға қажетті.

Төрт негізгі әрекеттер – спецификация, құрастыру, шекті тексеру және дамыту – жұмыстың әр түрлі үдерістерінде әр түрлі ұйымдастырылған. Сарқырама үлгісінде олар бірізділікпен ұйымдастырылған, ал сатылы құрастыруда олар алмасқан. Бұл әрекеттердің орындалуы бағдарламалық қамтамасыз етудің типіне, тартылған адамдар мен ұйымдастырушылық құрылымдарға байланысты. Мысалы, экстремалды бағдарламалауда спецификациялар карталарға жазылған. Тестер бағдарламаның алдында орындалып, құрастырылады. Түрлендіру нәтижесінде жүйелік қайта құру немесе рефакторинг орын алуы мүмкін.

2.2.1. Бағдарламалық қамтамасыз етуге қойылатын талаптар

Бағдарламалық қамтамасыз етуге талаптар құрастыру немесе талаптар – бұл жүйені дамыту мен құрастыру үшін жүйеден және шектеу сәйкестігінен қандай қызметтер талап етілетінін түсіну және анықтау үдерісі. Талаптарды құрастыру, әсіресе бағдарламалық үдерістің критикалық сатысы, бұл сатыдағы қателіктер алдағы уақыттағы жүйені жобалау және жүзеге асыру проблемаларына әкеліп соғады. Талаптарды құрастыру үдерісі (2.4-сурет) мүдделі тараптардың талаптарын қанағаттандыратын жүйені анықтайтын келісілген құжатты ұсынуға тырысады.

Бағдарламалық қамтамасыз етуді құрастыру аспаптары

Бағдарламалық қамтамасыз етуді құрастыру аспаптарына (кейде БҚЕ компьютерлік құрастыру немесе КЕЙСАМИ деп аталады) бағдарламалық қамтамасыз етуді құрастыру үдерісінің әрекеттерін қолдауға қолданылатын бағдарламалар жатады. Сондықтан бұл аспаптарға жобалық редакторлар, мәліметтер сөздігі, компиляторлар, құрастырудың жүйелік аспаптары және т.б. жатқызылады.

Бағдарламалық құралдар үдерісті қолдайды және үдерістің кейбір әрекеттерін автоматтандырады, құрастыру сатысындағы бағдарламалық қамтамасыз ету туралы ақпарат ұсынады. Автоматтандырылатын әрекеттер мысалына келесілер жатады:

- талаптар спецификациясының бір бөлігі немесе бағдарламалық қамтамасыз етуді жобалау ретінде графикалық жүйелік үлгілерді құрастыру;
- бұл графикалық үлгілердің кодтарын шығару;
- тұтынушымен интерактивті тәртіпте жасалатын графикалық интерфейс сипатынан қолданбалы интерфейстер шығару;
- орындау бағдарламасы туралы ақпарат ұсыну арқылы Бағдарламалық қосу;
- бағдарламаларды автоматты аудару, бағдарламалау тілінің жаңа нұсқасын пайдалану.

Аспаптар Құрастырудың интерактивті ортасы немесе IDE деп аталатын платформада біріктіріледі. Бұл интегралды тәсілмен байланысуға және жұмыс істеуге жеңіл болатындай етіп аспаптарды қолданатын бірыңғай құралдар жиынтығын қамтамасыз етеді.

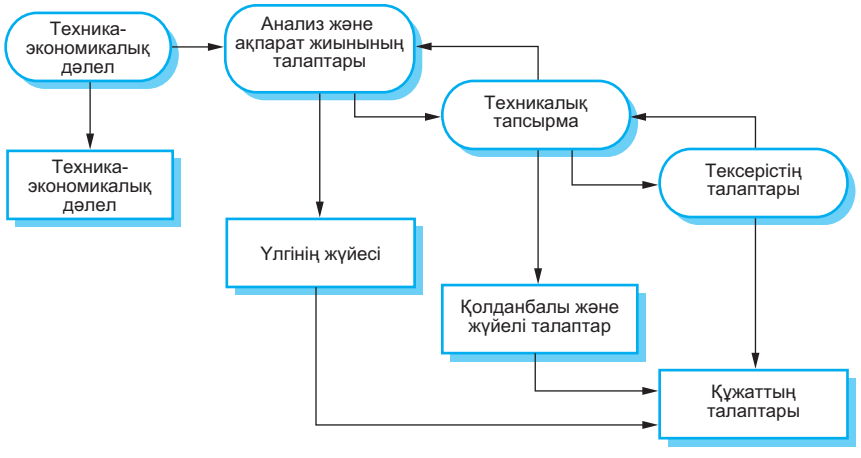
IDE КҮҢГІРТТЕУ кең қолданылады және бағдарламалық аспаптардың түрлі типтерін қосу үшін құрастырылған.

http://www.SoftwareEngineering-9.com/Web/CASE/

Талаптар әдетте детализацияның екі деңгейінде ұсынылады. Соңғы тұтынушылар мен клиенттер талаптарының жоғары деңгейлі сәйкестігін талап етеді; жүйе құрастырушылары толыққанды жүйелік спецификацияны қажет етеді.

Талаптарды құрастыру үдерісінде қызметтің төрт негізгі түрі бар:

1. *Техникалық-экономикалық негіздеме*. Бағалау белгілі бір тұтынушы бағдарламалық және аппараттық қамтамасыз етудің ағымдық технологияларына қанағаттануын анықтауға негізделген. Зерттеу ұсынылған жүйенің іскер көзқарас тұрғысынан экономикалық тиімді болатынын және бюджеттік шектеулер аясында құрастырылу мүмкіндігін қарастырады. Техникалық-экономикалық негіздеме салыстырмалы арзан және жедел болуы тиіс. Нәтиже бізге әрі қарай толық талдау жасаудың қажеттігін көрсетеді.



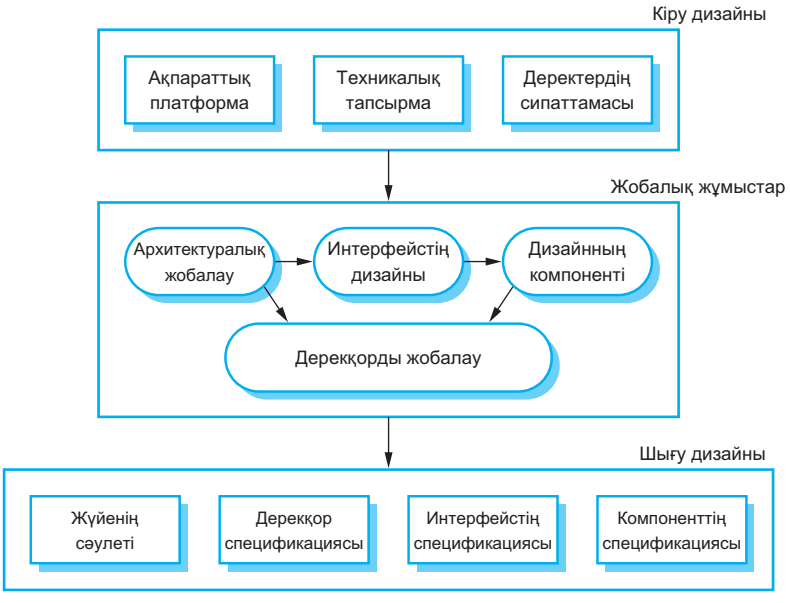
2.4-сурет. Құрастыру үдерісіне қойылатын талаптар

2. *Талаптар мен талдау*. Бұл – қолданыстағы жүйелерді бақылау, тұтынушылар және жабдықтаушылармен талқылау, тапсырмаларды талдау және т.б. жүргізу арқылы жүйелік талаптарды алу үдерісі. Бұл бір немесе бірнеше жүйелік үлгілер мен прототиптерді құрастырудан тұрады.
3. *Талаптардың спецификациясы*. Талаптардың спецификациясы – бұл талаптар қатарын анықтайтын талдамалы әрекет барысында ортақ құжатқа жинақталған ақпаратты аудару. Бұл құжатқа талаптардың екі типі енгізілуі мүмкін. Тұтынушының талаптары – клиент пен жүйені соңғы қолданушыға арналған жүйелік талаптарды абстрактілі бекіту; жүйелік талаптар – қызметінің толық сипаттамасы ұсынылатын болады.
4. *Талаптардың шектілігін тексеру*. Бұл әрекет орындаушылық, бірізділік және толыққандылық талаптарын тексереді. Бұл үдеріс барысында құжатта қателіктер анықталған. Және олар түзетілуі тиіс.

Оның себебі, әрекеттер қатаң бірізділікпен орындалмаған. Сондай-ақ, үдеріс барысында жаңа талаптар анықталады. Сондықтан, талдау, анықтау және спецификация әрекеттері алмасқан. Экстремалды бағдарламалау тәрізді жедел және икемді әдістерде талаптар тұтынушылық басымдылықтарға сәйкес сатылы құрастырылған, ал талаптар туралы ақпаратты жинау құрастырушылар командасының мүшесі болып табылатын тұтынушылардан түседі.

2.2.2. БҚЕ-ні жобалау және жүзеге асыру

Бағдарламалық қамтамасыз ету құрылысын жүзеге асру сатысы – бұл жүйелік спецификацияны орындалатын жүйеге түрлендіру үдерісі. Бұл үнемі бағдарламалық қамтамасыз етуді жобалау және бағдарламалау үдерістерінен тұрады, бірақ құрастырудың сатылы тәсілі қолданса, бұл бағдарламалық қамтамасыз етуге талаптарды жетілдіруді де енгізеді.



2.5-сурет. Құрастыру үдерісінің жалпы үлгісі

Бағдарламалық қамтамасыз етуді жобалау – бұл оның жүзеге асырылатын құрылысын, жүйеде қолданылатын үлгі мен құрылымның деректерін, жүйелік компоненттер арасындағы интерфейстерді және кейде қолданылатын алгоритмдерді сипаттау. Құрастырушылар аяқталған жобаға бірден қол жеткізбейді, бірақ жобаны көп мәрте құрастырады. Олар бастапқы жобаларды түзету үшін кері тәртіпте бақылаумен өз жобасын құрастыратындықтан, формальдылық қосып, тәптіштейді.

Сұлба бойынша жобалау үдерісінің сатылары бірізді. Іс жүзінде жобалау үдерісінің әрекеттері кезектескен. Бір сатының екіншісімен кері байланысы және жобаны бірізді өңдеу жобалаудың барлық үдерістерінде міндетті болып табылады.

Бағдарламалық қамтамасыз етудің көпшілігі интерфейс арқылы басқа бағдарламалық қамтамасыз етумен өзара әрекеттеседі. Оларға операциялық жүйе, мәліметтер базасы, аралық бағдарламалық қамтамасыз етулер және басқа да қолданбалы жүйелер жатады. Олар «бағдарламалық платформа», яғни бағдарламалық қамтамасыз ету орындалатын орта құрады. Бұл платформа туралы ақпарат – бұл жобалау үдерісіне мәнді ену, себебі құрастырушылар оны бағдарламалық қамтамасыз ету ортасына қалай тиімді интегралдау қажеттігін шешуі қажет. Талаптар спецификациясы – бұл бағдарламалық қамтамасыз ету қамтамасыз ететін қызметтіліктің сипаты және оның өнімділігі, сенімділік талаптары. Егер жүйе қолданыстағы мәліметтерді өңдеуі қажет болса, онда бұл мәліметтердің сипаты платформа спецификациясына енгізілуі мүмкін; керісінше, мәліметтерді жүйелік ұйымдастыру анықталуы үшін мәліметтердің сипаты жобалау үдерісіне енгізілуі қажет.

Құрамдастырылған әдістер

Құрамдастырылған әдістер – бұл бағдарламалық қамтамасыз етуді жобалауға ыңғайлы, мұнда графикалық үлгілер жобалау үдерісінің бөлігі ретінде құрастыру. Әдіс сонымен қатар үлгіні құрастыру үдерісі мен үлгінің әр типіне қолданылатын ережелерді анықтай алады. Құрамдастырылған әдістер жүйелер үшін құжаттаманы стандарттауға, әсіресе тәжірибесіз бағдарламалық қамтамасыз етуді құрастырушыларға пайдалы негізгі құрастыруларды қолдануға әкеліп соғады.

http://www.SoftwareEngineering-9.com/Web/Structured-methods/

Жобалау үдерісіндегі әрекеттер құрастырылған жүйеге байланысты әр түрлі болады. Мысалы, шынайы уақыттағы жүйелер үйлестіру жобасын талап етеді, бірақ мәліметтер базасын енгізе алмайды, осылайша, мәліметтер басазысының енгізілген жобалауы болмайды. 2.5-суретте информациялық жүйелер үшін жобалау үдерісінің бір бөлігі болатын төрт әрекет көрсетілген:

1. *Сәулеттік жоба*, мұнда Сіз жүйенің толық құрылымын, негізгі компоненттерді (кейде олар жүйеасты немесе модульдер деп аталады), олардың қатынасын және бөлінуін сәйкестендіресіз.
2. *Интерфейс дизайны*, мұнда Сіз жүйелік компоненттер арасындағы интерфейстерді анықтайсыз. Бұл интерфейстік спецификация бір мәнді болуы қажет. Нақты интерфейс болғанда бір компонент басқа компоненттерсіз қолданыла алады. Интерфейстік спецификациялар келісілгеннен кейін, компоненттер бір уақытта ресімделіп, құрастырылуы мүмкін.
3. *Құрамдас жоба,* мұнда Сіз әр жүйелік компонентті алып, құрастырасыз. Бұл бағдарламашыға қалдырылған белгілі бір жобаның функционалдығын жүзеге асыру болып табылады. Балама ретінде, бұл қайта қолданылатын компонентке немесе детальды жобалау үлгісіне жасалған өзгерістер тізімі болуы мүмкін. Жоба үлгісі автоматты түрде орындауды жүзеге асыру үшін қолданылуы мүмкін.
4. *Мәліметтер базасын жобалау*, мұнда Сіз мәліметтердің жүйелік құрылымын және базада ұсынылған үлгісін құрастырасыз. Мұнда да жұмыс қолданыстағы базаны қолдануға немесе жаңа мәліметтер базасын жасауға байланысты жүреді.

Бұл әрекеттер 2.5-суретте көрсетілген жобалық қуат қатарына әкеледі. Тетік пен ұсынылуы мәнді өзгеріп отырады. Критикалық жүйелер үшін жүйенің нақты сипаттамасын көрсететін егжей-тегжейлі жобалау құжаттары ұсынылуы қажет. Егер үлгімен басқарылатын әдіс қолданылса, нәтижелер негізінен сұлба түрінде болады. Құрастырудың жедел және икемді әдістері қолданылғанда, жобалау

үдерісінің нәтижелері спецификацияның жеке құжаттары, бірақ бағдарлама кодында ұсынылуы мүмкін.

Жобаның құрамдастырылған әдістері 1970, 1980 жылдары құрастырылып, UML және нысанды-бағдарланған жобаның ізашары болған (Budgen, 2003). Олар жүйенің графикалық үлгілерін жасауға сенеді және көбінесе, бұл үлгілердің кодын автоматты түрде жасайды. Бағдарламалық қамтамасыз етудің үлгілері абстракцияның әр түрлі деңгейінде жасалатын (MDD) немесе (Шмидт, 2006) үлгімен басқарылатын құрастыру құрамдастырылған әдістердің дамуы болып табылады. MDD үлгінің абстрактілі және спецификалық жүзеге асырылуын ажырата отырып, сәулеттік үлгілерге аса назар аударады. Үлгілер орындаушы жүйе содан жасалатындай етіп құрастырылған. Мен бұл құрастыру тәсілін 5-тарауда қарастырамын.

Жүйені жүзеге асыратын бағдарламаны құрастыру әлбетте жүйені жобалау үдерістерінен алынады. Қауіпсіз жүйелер тәрізді бағдарламаның кейбір кластары толық құрастырылатынына қарамастан, олардың жұмысы басталмас бұрын, жоба мен бағдарламаның кейінгі сатылары кезектеседі. Бағдарламалық қамтамасыз етуді құрастыру аспаптары жобаның қаңқалық бағдарламасын жасау үшін қолданылады. Бұған интерфейстерді анықтау және жүзеге асыру коды кіреді және көптеген жағдайда құрастырушы бағдарламаның әр компонентінің детальдарын тек қосу керек.

Бағдарламалау – дербес әрекет және оған ілесетін жалпы үдеріс болмайды. Кейбір бағдарламашылар жақсы түсінетін компоненттерін құрастырудан бастап, кейін аз түсінетіндеріне көшеді. Басқалары кері әдісті қолданады, яғни таныс компоненттерді соңына қалдырады, себебі олардың құрастырылуын біледі. Кейбір құрастырушылар мәліметтерді алдын ала анықтауға тырысады, басқалары мәліметтерді ұзақ уақыт көрсетпей қалдырады.

Әдетте, бағдарламашылар өздері құрастырған кодты тестілейді. Бұл жойылуға тиісті бағдарлама ақауларын жиі көрсетеді. Бұл тоқтату деп аталады. Ақауларды тестілеу және тоқтату – бұл әр түрлі үдерістер. Тестілеу ақаулардың болуын анықтайды. Тоқтату бұл ақаулардың орнын және оны жоюды орындайды.

Сіз ақауларды жойған кезде, бағдарламаның бақыланатын әрекеті туралы болжамдар жасап, сол болжамдарды тестілейсіз. Болжамдарды тестілеуге бағдарлама кодын қолмен іздеу жатады. Бұл мәселені анықтау үшін жаңа тестілеуді қажет етуі мүмкін. Ауыспалы аралық мәндерді көрсететін және орындалатын әрекеттерді бақылайтын тоқтатудың интерактивті құралдары тоқтату үдерісін қолдау үшін қолданылуы мүмкін.

2.2.3. БҚЕ шегін тексеру

Бағдарламалық қамтамасыз етудің шегін тексеру және бақылау (V&V) жүйенің спецификациясына сәйкестігін және клиент талаптарына жауап беруін көрсету үшін арналған. Жүйені орындаудағы бағдарламаны тестілеу – шектілікті тексерудің негізгі әдісі. Шектілікті тексеру сонымен қатар тұтынушы талаптарын анықтаудан

2.6-сурет. Тестілеу сатылары

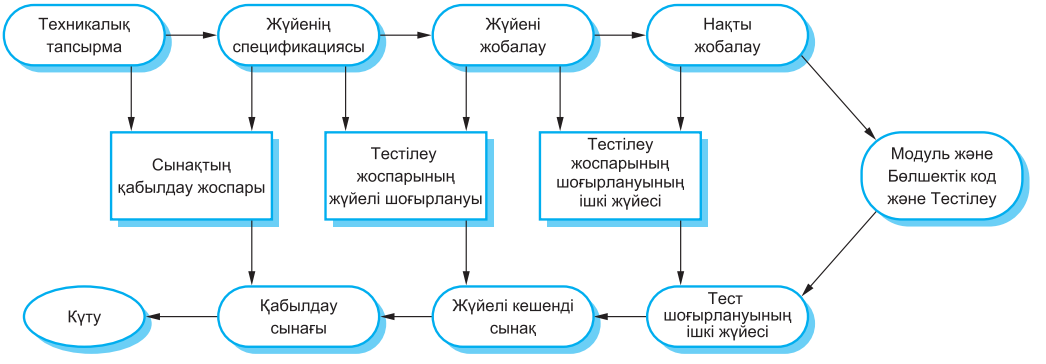
бағдарламаны құрастыруға дейінгі бағдарламалық үдерістің әр сатысында бақылау мен пікір сияқты үдерістерді тексеруден тұрады. Тестілеудің басымдылығынан шектілікті тексеру шығындарының көпшілігі оны жүзеге асыру барысында және соңында жасалады.

Ұсақ бағдарламалардан басқа, жүйелер жалғыз, тұтас модуль ретінде тестіленбеуі тиіс. 2.6-суретте тестілеудің үш сатылы үдерісі көрсетілген, алдымен жүйелік компоненттер, содан соң интегралданған жүйе, соңында клиент туралы мәліметтер бар жүйе тестіленген. Мінсіз жағдайда, интерфейстің ақаулары мен мәселелері бастапқы сатыда жүйе интегралданғанда анықталуы қажет. Бірақ ақаулар анықталса, бағдарлама түзетілуі тиіс, бұл тестілеу үдерісіндегі басқа сатылардың қайталануын талап етеді. Бағдарлама компоненттеріндегі қателіктер жүйені тестілеу кезінде анықталуы мүмкін. Сондықтан үдеріс – бұл кейінгі сатылардан үдеріс басына қайтып келетін ақпараттық қайталау.

Тестілеу үдерісіндегі сатылар:

1. *Тестілеуді дамыту*. Жүйені құрайтын компоненттер жүйені құрастыратын адамдармен тестіленеді. Әр компонент басқа жүйелік компоненттерге тәуелсіз тестіленген. Компоненттер нысандардың функциялары немесе кластары тәрізді қарапайым болуы мүмкін, немесе осы объектілердің когерентті топтарын құрайды. JUnit (Massol и Husted, 2003) тәрізді тестілеуді автоматтандыру аспаптары компоненттің жаңа нұсқасын жасағанда қайта орындалуы мүмкін.
2. *Жүйені тестілеу*. Жүйелік компоненттер жаңа жүйе жасау үшін интегралданған. Бұл үдеріс компоненттер арасындағы кездейсоқ әрекеттесу мен интерфейстік мәселелерден туындайтын қателіктерді анықтаумен байланысты. Бұл сондай-ақ, жүйенің функционалдық және функционалдық емес талаптарға жауап беруіне және жасалу сатысындағы жүйелік қасиеттерді тестілеуге қатысты. Ірі жүйелер үшін бұл көп сатылы үдеріс болады, мұнда компоненттер жүйе астын жасау үшін интегралданған және қорытынды жүйе құру үшін жекелей тестіленген.
3. *Қабылдау сынағы*. Бұл жүйе операциялық қолдануға қабылданбас бұрынғы тестілеудің қорытынды сатысы. Жүйе тестілеудің өзгермелі мәліметтерімен емес, жүйелік клиент мәліметтерімен тестіленген. Қабылдау сынағы жүйелік талаптарды анықтауда қателіктер мен кемшіліктерді көрсетуі мүмкін, себебі шынайы мәліметтер тестілеу мәліметтерінен өзгеше бо-

лады. Қабылдау сынағы сондай-ақ, тұтынушы талаптарын шын мәнінде қанағаттандырмайтын немесе жүйенің өнімділігіне мүмкіндік болмайтын мәселелерді көрсетеді.



2.7-сурет. БҚЕ жоспарымен басқарылатын тестілеу үдерісінің фазалары

Әдетте құрамдас құрастыру мен тестілеу үдерістері алмасып отырады. Бағдарламашылар өздерінің тестілеу мәліметтерін жасайды және біртіндеп кодты тестілейді, себебі ол бұрын құрастырылған. Бұл экономикалық саналы үдеріс, себебі бағдарламашы компонентті біледі және тестілеу сценарийін жасайтын жалғыз адам болып табылады.

Егер құрастыру үшін сатылы әдіс қолданылса, әр инкремент тестіленуі қажет, себебі сол инкременттің талабына сай жасалған. Экстремалды бағдарламалауда тесттер талаптармен бірге құрастырылған. Бұл тест жүргізушілер мен құрастырушыларға талаптарды түсінуге көмектеседі және тест кейстері жасалатындықтан кідірістер болмайтынына кепіл береді.

Жоспармен басқарылатын бағдарламалық үдеріс қолданылғанда (мысалы, критикалық жүйелік құрастыру) тестілеу бірқатар тестілеу жоспарларымен басқарылады. Тестілеушілердің тәуелсіз командасы жүйелік специфика мен жобадан құрастырылған осы тестілеу жоспарларымен жұмыс істейді. 2.7-сурет тестілеу жоспарларының тестілеумен және тәжірибелік-конструкторлық құрастырулармен бірігуін көрсетеді. Бұл кейде құрастырудың V-үлгісі деп аталады (V көру үшін бұрыңыз).

Қабылдау сынағын кейде «альфа-тестілеу» деп атайды. Қолданбалы жүйелер жалғыз клиентке арналып құрастырылған. Альфа-нұсқауды тестілеу үдерісі жүйе құрастырушысы мен клиент барлық талаптарды орындауға тиімді жүйе болатынын мақұлдағанша жалғасады.

Жүйені бағдарламалық өнім ретінде сату қажет болғанда, тестілеу үдерісі «бета-нұсқаны тестілеу» деп аталады. Бета-нұсқаны тестілеуге осы жүйені қолдануға келісім беретін клиенттерге жеткізу жатады. Олар мәселелер туралы жүйе құрастырушыларына хабарлайды. Бұл шынайы тұтынушыға өнімді көрсетеді және жүйелік құрастырушылар күтпеген қателіктерді анықтайды. Осы кері байла-

ныстан кейін жүйе өзгертіліп, бета-нұсқаны әрі қарай тестілеуге немесе жалпы сату үшін шығарылады.

2.2.4. БҚЕ дамуы

Бағдарламалық қамтамасыз ету жүйелерінің икемділігі – бұл бағдарламалық қамтамасыз етудің үлкен, күрделі жүйелерге қосылуының басты себептерінің бірі. Аппараттық құралдарды өндіру туралы шешім қабылданған кезде бұл жобаның қымбатқа түсетіні анықталды. Алайда бағдарламалық қамтамасыз етуге өзгерістер кез келген уақытта, жүйелік құрастыру кезінде және соңында енгізіле береді. Көлемдік өзгерістер жүйелік аппараттық құралдардағы тиісті өзгерістерден әлдеқайда арзан болады.

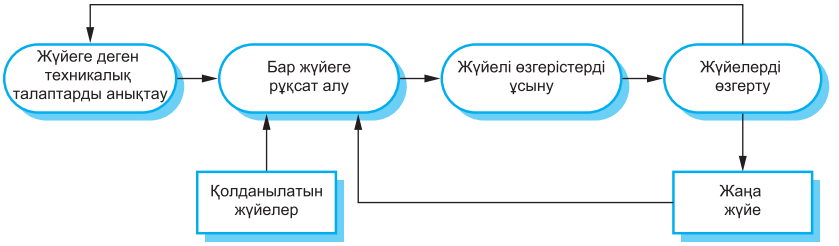
Тарихи тұрғыдан, бағдарламалық қамтамасыз етуді құрастыру үдерісі мен оның даму үдерісі (бағдарламалық қамтамасыз етуді қолдау) үнемі бөлініп қарастырылған. Адамдар бағдарламалық қамтамасыз етуді құрастыру туралы бағдарламалық қамтамасыз ету жүйесі басынан аяғына дейін құрастырылған шығармашылық әрекет деп ойлайды. Алайда, олар бағдарламалық қамтамасыз етуді қолдауды қызықсыз және күңгірт деп есептейді. Қолдану үдерісі бағдарламалық қамтамасыз етуді бастапқы құрастырудан жеңілдеу.

Осы құрастыру мен қолдану арасындағы бөлу барған сайын орынсыз болуда. Бағдарламалық қамтамасыз етудің кез келген жүйелері мүлдем жаңа және құрастыру мен қызмет көрсетуді континуум ретінде қарастыру орынды. Екі жеке үдерістің орнына бағдарламалық қамтамасыз етуді құрастыру туралы эволюциялық үдеріс ретінде қабылдаған шындыққа жақын (2.8-сурет), мұнда бағдарламалық қамтамасыз ету өзгермелі талаптар мен тұтынушылық талаптарға жауап ретінде өмір ағымында өзгеріп отырады.

2.3. Өзгерістермен күрес

Өзгерістер бағдарламалық қамтамасыз етудің барлық ірі жобаларында орын алады. Жүйелік талаптардың өзгеруі жүйені қамтамасыз ететін бизнес тәрізді, сыртқы қысым мен басқарудың басым өзгерістеріне жауап береді. Жаңа технолгиялар қолжетімді болғандықтан, құрастыру мен жүзеге асырудың жаңа мүмкіндіктері пайда болуда. Сондықтан бағдарламалық үдеріс үлгімінің қолданылуына тәуелсіз, құрастырылған бағдарламалық қамтамасыз етуді өзгерткен маңызды.

Өзгерістер бағдарламалық қамтамасыз етуді құрастыру шығындарына қосылады, себебі бұл аяқталған жұмысты қайта жасауды білдіреді. Оны түзету деп атайды. Мысалы, жүйедегі талаптардың арасындағы қатынастар талданып, жаңа талаптар сәйкестендірілсе, талдаудың кейбір аспектілері және талаптардың барлық талдауы қайталануы тиіс. Мұндайда жаңа талаптарды қамтамасз ету, құрастырылған кез келген бағдарламаны өзгерту және жүйені қайта тестілеу үшін жүйені қайта жобалау қажет.



2.8-сурет. Жүйе эволюциясы

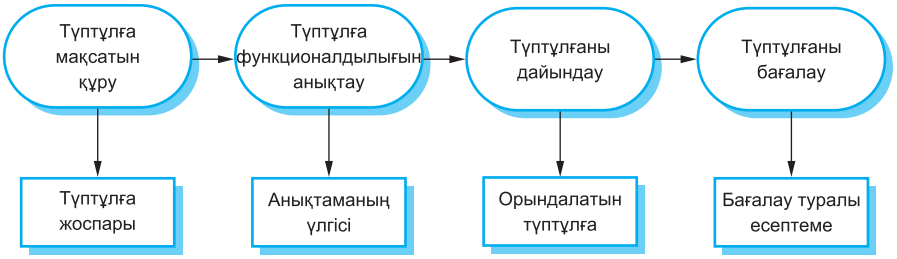
Түзету бойынша шығындарды азайтатын екі өзара байланысқан тәсіл бар:

1. Мәнді түзету жасалмас бұрын өзгерістерге әкелетін әрекеттерден тұратын бағдарламалық үдерістегі өзгерісті болдырмау керек. Мысалы, клиенттерге жүйенің басты ерекшеліктерін көрсету үшін түптұлғалы жүйе құрастырылуы мүмкін. Олар түптұлғамен тәжірибелер жасап, бағдарламалық қамтамасыз етудің жоғарғы бағасын бекітуде өз талаптарын жетілдіре алады.
2. Өзгеру рұқсаты, мұндағы үдеріс өзгерістер салыстырмалы төмен бағамен қаралатындай етіп құрастырылған. Бұған әдетте сатылы құрастырудың белгілі бір нысаны жатады. Ұсынылған өзгерістер әлі құрастырылмаған инкременттер түрінде жүзеге асырылады. Егер бұл мүмкін болмаса, өзгеріс енгізу үшін жалғыз инкрементті өзгерту қажет болады (жүйенің кіші бөлігі).

Осы бөлімде мен өзгерістерге төтеп беріп, осылайша жүйелік талаптарды өзгертудің екі әдісін талқылаймын. Бұлар:

1. Түптұлғаның жүйелік талдануы, мұндағы жүйе нұсқасы немесе жүйе бөлігі клиент талаптарын және кейбір жобалық шешім орындалуын тексеру үшін жедел құрастырылған. Бұл өзгерістердің болдырмауын қолдайды, себебі ол тұтынушыларға жүйені жеткізу алдында тексеруге мүмкіндік береді, осылайша өз талаптарын жетілдіреді. Талаптар саны жеткізуден кейін енгізілген ұсыныстарды өзгертеді, сондықтан оларды азайтқан жөн.

2. Инкременттік жеткізу – бұл клиентке пікір білдіру және тексеру үшін жеткізілетін жүйелік инкременттер. Бұл өзгерістің болдырмауын және болдыруын қолдайды. Бұл бүтін жүйе үшін уақытынан бұрын қойылған талаптарды болдырмауға және өзгерістердің арзан бағамен кейінгі инкременттерге енгізілуіне мүмкіндік береді. Рефакторинг түсінігі, яғни бағдарламаның құрылымы мен ұйымдастырылуын жақсарту өзгеріске жол беретін тағы бір маңызды механизм болып табылады. Мен бұны жедел және икемді әдістерге қатысты 3-тарауда талқылаймын.



2.9-сурет. Түптұлғаны құрастыру үдерісі

2.3.1. Түптұлға талдауы

Түптұлға – бұл ұғымдарды көрсету үшін, жобалық шешімдерді талдау үшін және мәселелер мен оларды шешу жолдары туралы білу үшін қолданылатын бағдарламалық қамтамасыз ету жүйесінің бастапқы нұсқасы. Түптұлғаны жедел, итерациялық құрастыру шығынды басқару үшін маңызды және мүдделі тараптар түптұлғамен алдын ала бағдарламалық үдерісте тәжірибе жасай алады.

Бағдарламалық қамтамасыз ету түптұлғасы талап етілуі мүмкін өзгерістерді жасау үшін бағдарламалық қамтамасыз етуді құрастыру үдерісінде қолданылады:

1. Талаптарды құрастыру үдерісі кезінде түптұлға ақпарат жинауда және жүйелік талаптардың шегін тексеруде көмектеседі.
2. Жүйені жобалау үдерісі кезінде түптұлға белгілі бір бағдарламалық өнімдерді зерттеу және қолданбалы интерфейс жобасын қолдау үшін қолданылады.

Жүйелік түптұлғалар тұтынушыларға жүйенің жұмысты жақсы қолдайтынын көрсетеді. Олар талаптар үшін жаңа идеялар қабылдап, бағдарламалық қамтамасыз етудегі артықшылықтар мен кемшіліктерді анықтай алады. Мұндайда олар жаңа жүйелік талаптар ұсынады. Сондай-ақ, түптұлға құрастырылғандықтан, ол ұсынылған талаптардағы қателіктер мен кемшіліктерге көрсете алады. Спецификацияда сипатталған функция пайдалы және нақты анықталған болып көрінеді. Алайда, бұл функция басқалармен қосылғанда, тұтынушылар олардың бастапқы түсінігі бұрыс және толық емес болғанын түсінеді. Бұл жағдайда талаптардың өзгертілген түсінігін көрсету үшін жүйелік спецификация өзгертілуі мүмкін.

Жүйелік түптұлға жоба тәжірибесін орындау үшін және ұсынылған жоба орындалуын тексеру үшін жүйені құрастыру кезінде қолданылады. Мысалы, мәліметтер базасын жобалау тұтынушылардың көп тараған сұрауларына тиісті қолжеткізу мүмкіндігін тексеру үшін жасалып, тестіленеді.

Түптұлға талдауы – қолданбалы интерфейсті жобалау үдерісінің бір бөлігі. Қолданбалы интерфейстердің динамикалық сипатына байланысты, мәтіндік сипаттар мен сұлбалар қолданбалы интерфейс талаптарын көрсету үшін жеткіліксіз.

Сондықтан, соңғы тұтынушының қатысуымен жедел моделдеу – бағдарламалық қамтамасыз ету жүйелері үшін тұтынушының графикалық интерфейсін құрастырудың жалғыз орынды әдісі.

Түптұлғалы құрастыру үшін үдеріс үлгісі 2.9-суретте көрсетілген. Түптұлғаны талдау мақсаттары үдерістің басынан анық болуы тиіс. Олар қолданбалы интерфейсті моделдеу үшін, сондай-ақ, функционалдық жүйелік талаптарды тексеріп, оны менеджерлердің қолдану мақсаттылығын көрсету үшін жүйе құрастыруы тиіс. Бір түптұлға барлық мақсаттарға жете алмайды. Егер мақсаттар анықталмаған болса, соңғы тұтынушылар түптұлғаның қызметін дұрыс түсіне алмайды. Демек, олар күткен пайданы ала алмайды.

Үдерістің келесі сатысында түптұлғалы жүйеге нені орналастыру және тіпті нені жою туралы шешім қабылдау қажет. Талдау шығындарын азайтып, жеткізу кестесін жылдамдату үшін Сіз түптұлғаның белгілі бір қызметін қалдыра аласыз. Сіз жады мен дыбыс беру уақыты тәрізді қажетсіз талаптарды қолдануды азайта аласыз. Түптұлғаның мақсаты тұтынушылық интерфейс орнату болмаса, қателерді өңдеу және басқару еленбеуі мүмкін. Бағдарламаның сенімділік пен сапа стандарттары азаюы мүмкін.

Үдерістің соңғы сатысы – түптұлғаны бағалау.

Сақтық шаралары пайдаланушыларды оқыту үшін берілген кезең барысында қолданылуы тиіс және де түптұлғалық мақсаттар бағалауға қатысты жоспарды алу үшін қолданылуы керек. Қолданушыларға жаңа жүйені мақұлдау үшін және қалыпты қолдану үлгісіне бейімделу үшін уақыт қажет. Әдетте, олар жүйені қолдана бастаған кезде, талаптардың қателігі мен кейбір кемшіліктерін анықтайды.

Түптұлғаны талдаудың жалпы мәселесі оның қорытынды жүйе ретінде дәл солай қолданыла алмайтындығында. Түптұлға тестері жүйелік қолданушыларға әдеттегідей бола алмайды. Тұптұлғалық бағалау кезіндегі оқу мерзімі жеткіліксіз болуы мүмкін. Егер тұптұлға баяу болса, онда талдау құралдары өзінің жұмыс тәсілін түзеп, жауап беру уақыты баяу жүйелер сипаттарынан құтыла алады. Қорытынды жүйеде ең жақсы жауаппен жабдықталған олар оны басқаша қолдануы мүмкін.

Құрастырушыларға кейбір кездері менеджерлер, әсіресе, бағдарламалық қамтамасыз етудің соңғы нұсқасын жеткізуде кідірістер болған кезде бос түптұлғаларды жеткізу үшін қысым көрсетеді. Бірақ бұл, әдетте, ақылға қонбайды:

1. Түптұлғалық құрастыру кезінде еленбей қалған өндірушілік, қауіпсіздік, тұрақтылық және сенімділік талаптары тәрізді функционалдық емес талаптарды қанағаттандыру үшін түптұлғаны реттеу мүмкіндігінің болмауы ықтимал.
2. Құрастыру кезіндегі жедел өзгеріс түптұлғаның құжатталмағанын білдіреді. Жобаның жалғыз спцификациясы – түптұлғалық код. Бұл ұзақ мерзімді қызмет көрсету үшін тиімсіз.
3. Түптұлғалық құрастыру кезінде енгізілген өзгерістер жүйе құрылымын нашарлатады. Жүйе күрделі және қымбат болады.

4. Түптұлғалық құрастыруға арналған сапаның ұйымдастырушылық стандарттары әдетте әлсірейді.

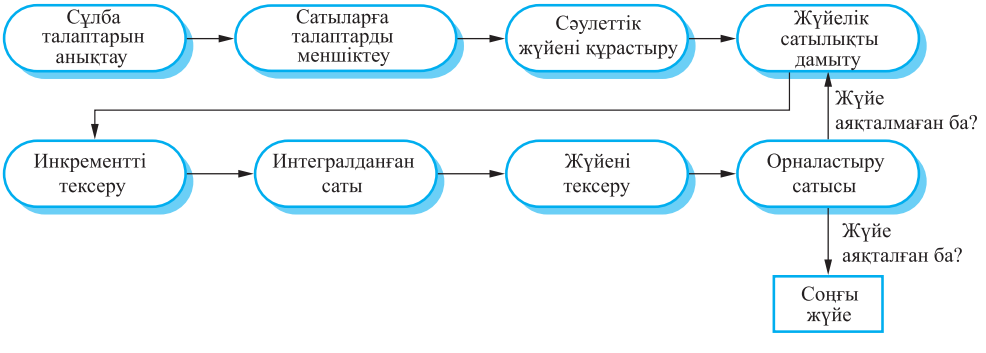
Түптұлғалар тиімді болу мақсатында орындалатын болуы тиісті емес. Жүйелік қолданбалы интерфейстің қағаз макеттері (Rettig, 1994) тұтынушының интерфейс дизайнын жетілдіру және қолдану сценарийі арқылы жұмыс кезінде тиімді болуы мүмкін. Олардың құрастырылуы арзан және бірнеше күнде жасалуы мүмкін. Бұл әдістің кеңеюі – бұл тек тұтынушылық интерфейс құрастырылған Оз еліндегі Сиқыршының түптұлғасы. Пайдаланушылар бұл интерфейспен өзара әрекеттеседі, бірақ олардың сұраулары оларды талдап, тиісті жауап табатын адамға беріледі.

2.3.2. Сатылы жеткізу

Инкременттік жеткізу (2.10-сурет) бағдарламалық қамтамасыз етуді құрастыру әдісі болып табылады, кейбір құрастырылған инкременттер клиентке жеткізіліп, операциялық ортада қолдану үшін орнатылады. Жеткізудің инкременттік үдерісінде клиенттер жүйемен қамтылатын қызметтерді жалпы белгілері бойынша сәйкестендіреді. Олар қызметтің қайсысы маңызды, қайсысы аса маңызды емес болатынын анықтайды. Жеткізудің көптеген инкременттері әр саты бойынша кейін анықталады және жүйелік функционалдықты қамтамасыз етеді. Қызметтерді инкременттерге бөлу қызмет басымдылығына, басында ұсынылған ең жоғарғы басымдылық қызметіне байланысты.

Құрастыру кезінде кейінгі инкеременттерге қойылатын талаптарды талдау орын алады, бірақ ағымдық инкременттің талаптары өзгертілмейді.

Саты аяқталып, қойылған соң клиенттер оны қолданысқа енгізе алады. Бұл оларға жүйелік функционалдық бөлігінің ерте жеткізілуін білдіреді. Олар жүйені сынай алады, және бұл кейінгі жүйелік инкременттерге талаптарын жеткізуге көмектеседі.



2.10-сурет. Біртіндеп жеткізу

Жаңа инкременттер аяқталғандықтан, олар қолданыстағы инкременттермен жүйелік функционалдық жақсарған сайын интегралданады. Инкременттік жеткізудің көптеген артықшылықтары бар:

1. Клиенттер бастапқы инкременттерді түптұлға ретінде пайдаланып, кейінгі жүйелік инкременттерге талаптарын жеткізетін тәжірибе ала алады. Түптұлғалардан ерекшелігі, бұл шынайы жүйе бөлігі, осылайша, қайта оқудың қажеті жоқ, жүйе толықтай қолжетімді.
2. Клиенттер пайда көрмес бұрын, барлық жүйе орнатылғанын күтуге тиісті емес. Алғашқы инкремент олардың ең критикалық талаптарын қанағаттандырады, осылайша олар бірден бағдарламалық қамтамасыз етуді қолдана алады.
3. Үдеріс сатылы құрастыру артықшылығын қолдайды, мұнда жүйелік өзгерістерді енгізу салыстырмалы жеңіл.
4. Ең жоғары басымдылық қызметі бастапқыда ұсынылып, инкременттер интегралданатындықтан, ең маңызды жүйелік қызметтер тестілеудің ең үлкен бөлігін алады. Бұл клиенттің жүйенің маңызды бөлігінде қателіктерді кездестіру ықтималдығының кемитінін білдіреді.

Алайда инкременттік жеткізудің мәселелері бар:

1. Жүйелердің көпшілігі жүйенің әр түрлі бөлігінде қолданылатын негізгі құралдар қатарын талап етеді. Талаптар түбегейлі анықталмайынша, инкремент жүзеге аспауы тиіс, содан соң барлық инкременттерге қажетті жалпы құралдарды сәйкестендіру қиынға түседі.
2. Итерациялық құрастыру ауыстырмалы жүйе құрастырылғанда да қиын болуы мүмкін. Қолданушылар ескі жүйенің барлық қызметтерін талап етеді, және толық емес жаңаны қабылдамай жатады. Сондықтан, тұтынушылармен пайдалы кері байланыс орнату қиын.
3. Итеративті үдерістердің мәні – спецификацияның бағдарламалық қамтамасыз етумен бірге құрастырылуында. Алайда, толық жүйенің спецификациясы жүйелік контрактінің тек бір бөлігі болатын, көптеген ұйымдар алатын үлгілермен сиыспайды. Сатылы тәсілде инкремент анықталмайынша, толық жүйенің спецификациясы болмайды. Бұл контрактінің жаңа нысанын талап етеді, бірақ үкіметтік мекемелер тәрізді ірі клиенттер мұны бейімдеу қиын деп санайды.

Жүйенің кейбір типтері үшін сатылы құрастыру және жеткізу тиімсіз. Бұл әр түрлі жерлерде істейтін командалардан тұратын үлкен жүйелер. Бұл жүйелердің кейбіреуінің бағдарламалық қамтамасыз етулері аппараттық құрастыру мен кейбір критикалық жүйелерге байланысты, мұнда барлық талаптар талдануы қажет, себебі кейбір әрекеттестіктер жүйенің қауіпсіздігіне әсер етеді.

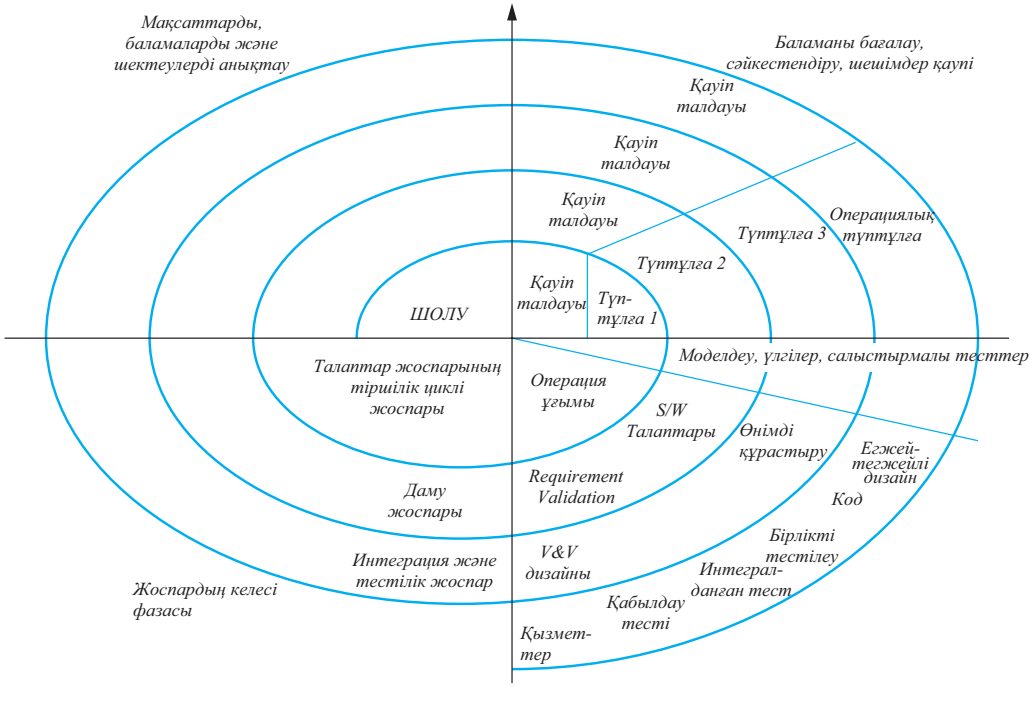
Бұл жүйелер әрине сенімсіз және өзгермелі талаптардан зардап шегеді. Сондықтан бұл мәселелерді шешіп, сатылы құрастырудың пайдасын көру үшін,

жүйелік талаптар мен жобасы бар тәжірибелерге арналған платформа ретінде көпмәртелі жүйелік түптұлға құрастырылатын үдеріс қолданылады. Түптұлғадан алынған тәжірибемен категориялы талаптар келісіледі.

2.3.3. Боэмның шиыршықты үлгісі

Бағдарламалық үдерістің тәуекел басқаруымен платформасын (шиыршықты үлгі) Боэм ұсынған (1988). Ол 2.11-суретте көрсетілген. Мұнда бағдарламалық үдеріс бірізді әрекет емес, шиыршықты түрде бір әрекеттен екіншіге кері бағыттағы бақылаумен ұсынылады. Осылайша, ішкі цикл жүйелік атқарушылықпен сипатталады, яғни талаптарды анықтайтын келесі цикл, жүйені жобалайтын келесі цикл және т.б.

Шиыршықты үлгі өзгерістердің жойылуын өзгерістердің болуымен үйлестіреді. Бұл өзгерістердің жоба тәуекелдерінің нәтижесінде болатынын білдіреді, оған өзгерісті кеміту үшін тәуекелді басқару жатады.



2.11-сурет. Боэмның бағдарламалық қамтамасыз етудің шиыршықты үлгісі (© IEEE 1988)

Шиыршықтағы әр цикл төрт секторға бөлінген:

1. *Объективті белгілеу*. Жобаның бұл фазасына нақты мақсаттар анықталған. Үдеріс пен өнім шектеулері сәйкестендірілген. Басқарудың егжей-тегжейлі

жоспары құрылған. Жоба тәуекелдері сәйкестендірілген. Осы тәуекелдерге байланысты баламалы стратегиялар жоспарланады.

2. *Тәуекел дәрежесін бағалау және қысқарту*. Жобаның сәйкестендірілген тәуекелдерінің әр қайсысына толық талдау жасалған. Тәуекелді қысқартуға шаралар қабылдануда. Мысалы, талаптар сәйкес келмеуі туралы қауіп төнсе, түптұлғалы жүйе құрастырылуы мүмкін.
3. *Құрастыру және шектілікті тексеру*. Тәуекелді бағалағаннан кейін жүйеге арналған құрастыру үлгісі таңдалады. Мысалы, қолданбалы интерфейс қаупі басым болса, түптұлғаның бос талдауы тиімді болады. Егер қауіпсіздік қауіптері негізгі қарастыруда болса, ресми түрлендіру негізіндегі құрастыру тиімді үдеріс болады және т.б. егер негізгі сәйкестендірілген тәуекел – бұл жүйе асты интеграциясы болса, сарқырама үлгі құрастырудың тиімді үлгісі болады.
4. *Жоспарлау*. Жоба қаралып, шиыршықты циклді әрі қарай жалғастыру не жалғастырмау туралы шешім қабылданады. Егер жалғастыру туралы шешім қабылданса, жобаның келесі фазасына жоспарлар құрылады.

Шиыршықты үлгі мен бағдарламалық үдерістің басқа үлгілері арасындағы айырмашылық уақтылы тәуекелді тануда. Шиыршықтың циклі өнімділік және функционалды сияқты мақсаттарды құрастырудан басталады. Бұл мақсаттарға жетудің баламалы тәсілдері мен шектеулер тізбеленеді. Әр балама ір мақсатқа қатысты бағаланады және жобаның тәуекел көздері сәйкестендіріледі.

Келесі қадам бұл тәуекелдерді ақпарат жинау жолымен шешеді, олар толыққанды талдау, түптұлға талдауы және үлгілеу. Тәуекелдер бағаланып, кейбір құрастырулар орындалған соң, үдерістің келесі фазасына әрекеттер жоспарланады. Бейресми түрде тәуекел бір әрекеттің бұрыс орындалуын білдіреді. Мысалы, егер мақсат – бағдарламалаудың жаңа тілін қолдану болса, оның тәуекелі – бұл қолжетімді компиляторлардың сенімсіз болуы немесе тиімді объектілі код өндірмеуі болады. Тәуекелдер бағдарламалық қамтамасыз етудің өзгеруі мен кесте және артық шығын сияқты жоба мәселелеріне әкеледі. Осылайша, тәуекелді азайту – жобаларды басқарудың маңызды әрекеті болып табылады. Жобаны басқарудың негізгі бөлімі, тәуекелді басқару 22-тарауда қарастырылған.

2.4. Рационалды біріктірілген үдеріс

Рационалды біріктірілген үдеріс (RUP) (Krutchen, 2003) – бұл UML мен бағдарламалық қамтамасыз етуді Құрастырудың Біріктірілген Үдерісі бойынша жұмыстардан алынған үдерістің заманауи үлгісінің мысалы болады (Rumbaugh, және бас., 1999; Arlow және Нойштадт, 2005). Бұл үдерістің гибридті үлгісінің жақсы мысалы болады. Ол үдерістердің барлық әмбебап үлгілерінің элементтерін біріктіреді (2.1-бөлім), жобадағы тәжірибені көрсетеді (2.2-бөлім) және үлгілеу мен инкрементті жеткізуді ұсынады (2.3-бөлім).

RUP әдетте үш көзқарас тұрғысынан сипатталады:

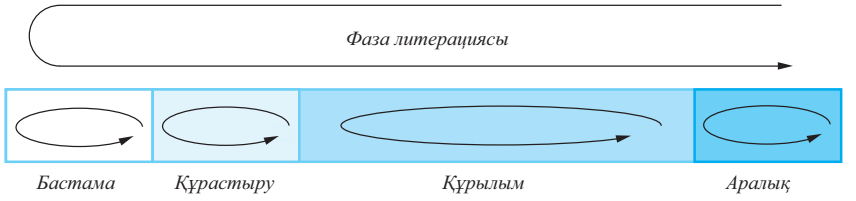
1. Ұзақ уақыт ішінде үлгі фазаларын көрсететін серпінді перспектива.
2. Үдерістің тағайындалған әрекеттерін көрсететін статикалық перспектива.
3. Үдеріс барысында жақсы әдістердің қолданылуын білдіретін тәжірибе перспективасы.

RUP көпшілік сипаттамалары статикалық және серіппелі перспективаларды ортақ сұлбаға біріктіруге тырысады (Krutchen, 2003). Менің ойымша бұл үдерісті түсінуге қиындатады, сондықтан мен әр перспективаның жеке сипаттамаларын қолданамын.

RUP – бағдарламалық үдерістегі төрт дискретті фазаны сәйкестендіретін сатылы үлгі. Алайда, фазалары үдеріс әрекеттеріне теңестірілген сарқырама үлгіден ерекшелігі, RUP фазалары көбінесе техникалық мәселелер емес, бизнеспен байланысты. 2.11-сурет RUP фазаларын көрсетеді. Олар:

1. *Бастама*. Бастама фазасының мақсаты жүйеге экономикалық үлгі орнатуда. Сіз жүйемен әркеттесетін және сол әрекеттестікті анықтайтын барлық сыртқы объектілерді сәйкестендіруіңіз қажет (адамдар мен жүйелер). Сіз бұл ақпаратты жүйе бизнеске салатын салымды бағалау үшін қолдана аласыз. Егер бұл салым мәнсіз болса, жобадан бас тартылуы мүмкін.
2. *Құрастыру*. Құрастыру фазасының мақсаттары проблемалық облысты түсінуден, жүйеге сәулеттік платформа орнатудан, жоба жоспарын құрудан және жобаның түйін тәуекелдерін сәйкестендіруден тұрады. Бұл фазаның аяғында Сізде жүйеге қойылған талаптар үлгісі болуы қажет, бұл UML қолдану мысалдарының қатары, сәулеттік мысал және салыстырмалы бағдарламалық қамтамасыз етуді құрастыру жоспары болуы мүмкін.
3. *Құрылым*. Құрылым фазасына жүйені жобалау, бағдарламалау және тестілеу жатады. Жүйе бөліктері қатарлас құрастырылып, осы фаза барысында интегралданған. Осы фазаның аяғында Сізде бағдарламалық қамтамасыз етудің жұмыс жүйесі және қолданушыға жеткізуге әзір құжаттама болуы қажет.
4. *Аралық*. RUP қорытынды фазасы шынайы ортада жұмыс істеуі үшін жүйенің құрастыру бірлестігінен тұтынушылық бірлестікке алмасуымен байланысты. Бұл бағдарламалық үдеріс үдгілерінде көп жағдайда еленбеген, бірақ іс жүзінде қымбат және кейде мәселелі әрекет болып табылады. Осы фазаның аяғында Сізде операциялық ортада дұрыс істейтін бағдарламалық қамтамасыз етудің құжатталған жүйесі болуы қажет.

RUP итерациясы екі тәсілмен сүйемелденеді. Әр фаза сатылы құрастырылған нәтижелері бар итеративті тәсілмен қабылданады. Сондай-ақ, азалар жиынтығы да сатылы қабылдануы мүмкін, бұл 2.12-суретте Алмасудан Басына өтетін циклдық орындау нұсқары арқылы көрсетілген.



2.12-сурет. Рационалды біріктірілген үдерістегі фазалар

RUP статикалық ұсынылуы құрастыру үдерісі барысында орын алатын әрекеттерде шоғырланады. Олар RUP сипаттамасында операциялар ағымы ретінде көрінеді. Үдерісте сәйкестендірілген базалық үдеріс операцияларының алты ағымы және қолдау операцияларының үш базалық ағым бар. RUP UML-мен үйлестіріле құрастырылған, осылайша операциялар ағымының сипаттамасы бірізділік үлгілері, объектілі үлгілер және т.б. тәрізді UML байланысқан үлгілер аймағына бағдарланған. Базалық құрастыру және қолдау операцияларының ағымдары 2.13-суретте сипатталған.

Серіппелі және статикалық ұсыныстардың артықшылығы құрастыру үдерісінің базалары операциялардың белгілі бір ағымдарымен байланыссыз болуында. Түптеп келгенде, аз дегенде RUP операцияларының барлық ағымдары үдерістің барлық сатыларында белсенді болады. Үдерістің ерте фазаларында күштің барлығы кейінгі фазалардағы, тестілеу мен орналастырудағы бизнес-үлгілеу тәрізді операциялар ағымына бағытталуы ықтимал.

RUP жөніндегі перспективалы көзқарас жүйелік құрастыруда қолдану үшін ұсынылатын бағдарламалық қамтамасыз етуді құрастырудың жақсы әдістерін сипаттайды. Әлдеқайда сәтті тәжірибенің алты негізгі әдісі ұсынылады:

1. *Бағдарламалық қамтамасыз етуді көп мәрте құрастырыңыз*. Жүйе инкременттерін тұтынушылық басымдылық негізінде жоспарлаңыз және жүйенің ең жоғары функциясын құрастыру үдерісі барысында жасаңыз.
2. *Талаптарды басқарыңыз*. Клиент талаптарын құжаттап, сол талаптардағы өзгерістерді бақылаңыз. Жүйедегі өзгерістерді қабылдамас бұрын, әсерін талдаңыз.
3. *Компонентті-бағдарлы сәулетті қолданыңыз*. Жүйе сәулетін жоғарыда талданғандай, құрамдастырыңыз.
4. *Көзбен бағдарламалық қамтамасыз етудің үлгісін жасаңыз*. Бағдарламалық қамтамасыз етудің статикалық және серіппелі ұғымдарын ұсыну үшін UML графикалық үлгілерін қолданыңыз.
5. *Бағдарламалық қамтамасыз етудің сапасын тексеріңіз*. Бағдарламалық қамтамасыз етудің сапаның ұйымдастырушылық стандарттарына сәйкес келетініне кепіл беріңіз.
6. *Бағдарламалық қамтамасыз етудің өзгерістерін басқарыңыз*. Жүйені басқару және конфигурацияны басқару тәртібін қолдана отырып, бағдарламалық қамтамасыз етудің өзгерістерін басқарыңыз.

Операциялар ағымы	Сипаттама
Бизнес-үлгілеу.	Бизнес-үрдістер бизнес-қолдану жағдайларын қолдану арқылы үлгіленген
	Жүйемен өзара әрекеттесетін талаптар агенттері сәйкестендірілген және жүйелік талаптарды үлгілеу үшін жағдайларды пайдаланады.
Талдау және жобалау.	Жоба үлгісі сәулеттік үлгіні, компоненттік үлгіні, объектіні үлгіні және бірізді үлгіні қолдана отырып жасалады және құжатталады
Жүзеге асыру.	Жүйедегі компоненттер жүйе астында жүзеге асырылып, құрастырылған. Автоматты код генерация бұл үрдісті жеделдетуге ықпал етеді
Тестілеу.	Тестілеу – жүзеге асырумен бірге орындалған итеративті үрдіс. Жүйені тестілеу жүзеге асыру әрекетінен кейін жалғасады
Өрістету	Өнім шығарылып, қолданушыға таратылады және жұмыс орнында орнатылады.
Конфигурацияны және өзгерістерді басқару.	Бұл операциялар ағымы жүйедегі өзгерістерді басқарады (25-тарауды қара)
Жобаларды басқару	Бұл қолдау операцияларының ағымы жүйенің құрастырылуын басқарады (22 және 23-тарауларды қара)
Қоршаған орта	Бұл операциялар ағымы бағдарламалық қамтамасыз етуді құрастыру командасына қолжетімді тиісті бағдарламалық қамтамасыз етуді құруға қатысты

2.13-сурет. Рационалды біріктірілген үдерістегі операциялардың статикалық ағымдары

RUP құрастырудың барлық типіне, мысалы, кіріктірме бағдарламалық қамтамасыз етуді құрастыруға жарамайды. Алайда бұл шын мәнінде 2.1-тарауда талқыланған үш әмбебап үдеріс үлгілерін қиыстыратын әдісті ұсынады. RUP ең маңызды инновациялар – бұл операцилар фазалары мен ағымдарын және тануды бөлу болып табылады. Қолданушы ортасында орнатылған бағдарламалық қамтамасыз ету – бұл үдерістің бір бөлігі. Фазалар серпінді және мақсаттары бар. Операциялар ағымдары статикалық және ортақ фазамен байланыссыз, бірақ әр фазаның мақсатына жету үшін құрастыру барысында қолданылатын техникалық әрекеттер болып табылады.

НЕГІЗГІ ТҰЖЫРЫМДАР

- Бағдарламалық қамтамасыз ету жүйелерін жасауға енгізілген әрекеттің бағдарламалық үдерістері. Бағдарламалық үдерістің үлгілері – бұл үдерістердің абстрактілі көріністері.
- Үдерістердің жалпы үлгілері бағдарламалық үдерістердің ұйымдастырылуын сипаттайды. Бұл жалпы үлгілердің мысалына сарқырама үлгі, сатылы құрастыру және қайта қолдануға бағытталған құрастыру жатады.
- Талаптарды құрастыру – бұл бағдарламалық қамтамасыз етуге талаптарды құрастыру үдерісі. Спецификациялар клинттің жүйелік қажеттіліктерін жүйелік құрастырушыларға тапсыруға арналған.
- Құрастыру және жүзеге асыру үдерістері бағдарламалық қамтамасыз етудің орындалатын жүйесіне спецификацияны түрлендіруге қатысты. Құрастырудың жүйелік әдістері осы түрленудің бір бөлігі ретінде қолданылады.
- бағдарламалық қамтамасыз етудің шектілігін тексеру – бұл жүйенің спецификацияға сәйкестігін және жүйе тұтынушыларының қажеттіліктерін қанағаттандыруын тексеру үдерісі.
- Бағдарламалық қамтамасыз етудің дамуы Сіз жаңа талаптарды қанағаттандыру үшін бағдарламалық қамтамасыз етудің қолданыстағы жүйелерін өзгерткенде орын алады. Өзгерістер толассыз, сондықтан пайдалы болып қала беру үшін бағдарламалық қамтамасыз ету дамуы қажет.
- Үдерістер өзгерістерді қабылдауы үшін әрекеттерден тұруы тиіс. Оған талаптар мен жобалар туралы кері шешімдерді болдырмауға көмектесетін түптұлғаны талдау фазасы жатады. Үдерістер енгізілген өзгерістер жүйені бұзбайтындай етіп құрамдастырылуы мүмкін.
- Рационалды Біріктірілген Үдеріс – бұл фазалардан тұратын (бастама, құрастыру, құрылым және алмасу), бірақ әрекеттерге бөлінетін (талаптар, талдау және жоба, т.б.) үдерістің заманауи әмбебап үлгісі.

ҚОСЫМША ӘДЕБИЕТТЕР

Managing Software Quality and Business Risk. This is primarily a book about software management but it includes an excellent chapter (Chapter 4) on process models. (M. Ould, John Wiley and Sons Ltd, 1999.)

Process Models in Software Engineering. This is an excellent overview of a wide range of software engineering process models that have been proposed. (W. Scacchi, *Encyclopaedia of Software Engineering,* ed. J.J. Marciniak, John Wiley and Sons, 2001.) http://www.ics.uci.edu/~wscacchi/Papers/SE-Encyc/Process-Models-SE-Encyc.pdf.

The Rational Unified Process — An Introduction (3rd edition). This is the most readable book available on the RUP at the time of this writing. Krutchen describes the process well, but I would like to have seen more on the practical difficulties of using the process. (P. Krutchen, Addison-Wesley, 2003.)

ЖАТТЫҒУЛАР

2.1. Құрастырылған жүйе типіне негізделген жауабыңызды дәлелдеңіз, келесі жүйелерді басқару негіздемесі болатын бағдарламалық үдерістің тиімді әмбебап үлгісін ұсыныңыз:

Автокөлікті тежейтін антиблоктау жүйесін басқаруға арналған жүйе

Бағдарламалық қамтамасыз етуді қолдауға арналған виртуалды шындық жүйесі

Қолданыстағы жүйені алмастыратын университеттік есеп жүйесі

Қолданушыларға қоршаған ортаға аз әсермен саяхаттар жоспарлауға көмектесетін саяхат жоспарлаудың интерактивті жүйесі.

2.2. Сатылы құрастырудың неге бизнес үшін бағдарламалық қамтамасыз ету жүйесінің дамуындағы ең тиімді тәсілі болатынын түсіндіріңіз. Неге бұл үлгі шынайы уақыт жүйесін құрастыруға тиімсіз?

2.3. 2.3-суретте көрсетілген қайта пайдалануға негізделген үдерісті қарастырыңыз. Неге үдерісте талаптарды құрастыруда екі жеке әрекеттің болғаны маңызды.

2.4. Қолданушы талаптарн құрастыру мен жүйелік талаптарды құрастыруда ажыратудың қажеттігін негіздеңіз.

2.5. Бағдарламалық қамтамасыз етуді жобалау үдерісіндегі және әрекет нәтижелеріндегі негізгі қызмет түрлерін сипаттаңыз. Сұлбаны қолдана отырып, сол әрекет нәтижелері арасындағы қатынастарды көрсетіңіз.

2.6. Неге күрделі жүйелерде өзгерістер болатынын түсіндіріңіз, өзгерістерді болжайтын бағдарламалық үдеріс әрекеттеріне мысал келтіріңіз (түптұлға талдауы мен инкрементті жеткізуден басқа), бағдарламалық қамтамасыз етуді өзгерістерге берік етіп жасаңыз.

2.7. Түптұлға ретінде жасалған жүйелер неге өндірістік жүйе ретінде қолданбайтынын түсіндіріңіз.

2.8. Боэмның шиыршықты үлгісі неге өзгерістер мен рұқсат әрекетін болдырмайтын және қолдайтын бейімделген үлгі болатынын түсіндіріңіз. Бұл неге болатынын болжап көріңіз.

2.9. Бағдарламалық үдерістің статиклық және серіппелі ұсыныстарын Рационалды Біріктіру Үдерісінде қамтамасыз ету артықшылқтары қандай?

2.10. Тарихи тұрғыда, технологияның енуі еңбек нарығына өзгерістер енгізіп, уақытша болса да адамдарды жұмыстан алшақтатты. Технологиялық үдерістердің автоматтандырлуы бағдарламалық қамтамасыз ету құрастырушыларының мұндай зардап шегетінін, не шекпейтінін талқылаңыз. Егер олай болмайды деп есептесеңіз, түсіндіріңіз.

ПАЙДАЛАНЫЛҒАН ӘДЕБИЕТТЕР

Arlow, J. and Neustadt, I. (2005). *UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design (2nd Edition)*. Boston: Addison-Wesley.

Boehm, B. and Turner, R. (2003). *Balancing Agility and Discipline: A Guide for the Perplexed*. Boston: Addison-Wesley.

Boehm, B. W. (1988). 'A Spiral Model of Software Development and Enhancement'. *IEEE Computer*, **21** (5), 61–72.

Budgen, D. (2003). *Software Design (2nd Edition)*. Harlow, UK.: Addison-Wesley.

Krutchen, P. (2003). *The Rational Unified Process — An Introduction (3rd Edition)*. Reading, MA: Addison-Wesley.

Massol, V. and Husted, T. (2003). *JUnit in Action*. Greenwich, Conn.: Manning Publications Co.

Rettig, M. (1994). 'Practical Programmer: Prototyping for Tiny Fingers'. *Comm. ACM*, **37** (4), 21–7.

Royce, W. W. (1970). 'Managing the Development of Large Software Systems: Concepts and Techniques'. IEEE WESTCON, Los Angeles CA: 1–9.

Rumbaugh, J., Jacobson, I. and Booch, G. (1999). *The Unified Software Development Process*. Reading, Mass.: Addison-Wesley.

Schmidt, D. C. (2006). 'Model-Driven Engineering'. *IEEE Computer*, **39** (2), 25–31.

Schneider, S. (2001). *The B Method*. Houndmills, UK: Palgrave Macmillan.

Wordsworth, J. (1996). *Software Engineering with B*. Wokingham: Addison-Wesley.

3.

Бағдарламалық жасақтаманың икемді әдістемелері

Мақсаттар

Осы тараудың негізгі мақсаты – оқырманды бағдарламалық қамсыздандыру дайындаудың икемді әдістемелерімен таныстыру. Аталмыш тарауды оқып болған соң, Сіз:

- икемді бағдарламалық қамсыздандырудың логикалық әдістемесін және жоспарланған жобалаудың ерекшелігін аңғара аласыз;
- төтенше бағдарламалаудың негізгі әдістемелерін әрі икемді әдістердің жалпы қағидаттарына қатынасын анықтай аласыз;
- Scrum әдістемесінің жобалардың икемді басқарылуын жүзеге асырудағы рөлін түсінетін боласыз;
- Ірі ауқымды бағдарламалық қамсыздандыру жүйелерін дайындау кезінде қолданылатын икемді әдістерді дұрыс бағалай білуді үйренесіз.

Мазмұны

Бүгінгі таңда бизнес жаһандық және шапшаң өзгеріп отыратын орта жағдайында дамып келеді. Ол жаңа мүмкіндіктердің пайда болуына және жаңа нарықтардың дамуына қарай әрекет етуі, сондай-ақ тез өзгере беретін экономикалық шарттарға және бәсекелес тауарлар мен қызметтердің пайда болуына бейімделе білуі тиіс. Бағдарламалық қамсыздандыру барлық дерлік іскерлік операциялардың бір бөлігі болып табылады, сондықтан жаңа мүмкіндіктерді жіберіп алмау әрі бәсекелестер тарапынан көрсетілетін қысымға жауап қайтара білу үшін жаңа бағдарламалық қамсыздандыру барынша қысқа мерзім ішінде дайындалады. Осылайша, қазіргі уақытта бағдарламалық қамсыздандыру жүйелеріне қойылатын ең маңызды талаптар оны жылдам дайындауды және іске қосуды қамтиды. Шындығында, көптеген кәсіпорындар бағдарламалық қамсыздандырудың сапасын құрбан етіп, қажетті бағдарламалық қамсыздандыруды тез іске қосу талабын қоюға дайын. Мұндай кәсіпорындар өзгермелі жағдайда қызмет жасайтындықтан, бағдарламалық қамсыздандыруға қойылатын тұрақты талаптар тізімін жасау мүмкін емес дерлік. Бастапқы талаптар шарасыз өзгертіледі, себебі тапсырыс берушілер жүйенің жұмыс тәжірибесіне қалайша ықпал ете алатынын, ол қалай басқа жүйелермен өзара әрекет ете алатынын және қандай тұтынушылық әрекеттер автоматтандырылған болуы тиіс екендігін алдын ала білу мүмкін емес деп есептейді. Шын мәнінде, қандай талаптар қойылу керек екендігін жүйе іске қосылып, тұтынушылар осы жүйемен нақты жұмыс тәжірибесін алғаннан кейін ғана белгілі бола алады. Тіпті, сол кезде осы қойылған талаптар нақты сыртқы факторлардың салдарынан тез әрі болжанбаған өзгеріске ұшырауы әбден мүмкін. Жүйені енгізу сәтінде бағдарламалық қамтамасыз ету ескіріп қалуы ықтимал.

Барлық талаптардың анықтамасын және содан кейін жүйені жобалау, құру және тестілеуді болжамдайтын бағдарламалық қамтамасыз етудің әзірлеу жұмысы жылдам үдеріс болып қарастырылмайды. Қойылған талаптардың бірі өзгерсе, әлде жүйені жобалау кезінде бір мәселе пайда болса, бүкіл жүйені құрастыру үдерісі қайтадан басталады. Сондықтан жобалау үдерісі ұзаққа созылып, тапсырыс берушіге бағдарламалық қамсыздандырудың соңғы нұсқасы анықталған уақыттан кеш жетеді.

Қауіпсіздікке негізделген бағдарламалық қамсыздандыру үшін жүйенің толықтай анализі өте маңызды болып табылады, осындай жағдайда жоспарлау арқылы басқарылатын тәсіл қолданылуы жөн. Бірақ шапшаң өзгеріп отыратын орта жағдайында бұл әдіс кейбір мәселелерді туғызуы мүмкін. Бағдарламалық қамсыздандыру пайдалануға дайын болған кезде, оның шын мәніндегі қолдануға арналған мақсаты әбден өзгеріп қалуы ықтимал. Сондықтан бағдарламалық қамсыздандырудың шапшаң дамуына негізделген жобалау үдерісін құру қажет, әсіресе, бизнес жүйелерінде мұндай әдістеме өте пайдалы.

Жүйені және өзгермелі талаптарды жүргізуін жүзеге асыра алатын бағдарламалық қамсыздандырудың шапшаң дамуына негізделген үдерістің қажеттілігі анықталды. 1980 жылы IBM кезеңдік даму әдісін ұсынған болатын (Mills et al., 1980). Осы жылы бағдарламалау тілінің төртінші буыны енгізілген, ол бағдарламалық қамсыздандыруды құрастыруын тездетуге бағытталған (Martin, 1981). Бірақ 1990 жылы бұл жаңадан енгізілген үдеріс DSDM (Stapleton, 1997),

Scrum (Schwaber and Beedle, 2001) және төтенше бағдарламалау (Beck, 1999; Beck, 2000) әдістерімен шектеліп кеткен.

Шапшаң бағдарламалық қамсыздандыру үдерісі тапсырыс берушіге бағдарламалық қамсыздандыруды тез әрі ыңғайлы түрде жеткізуге негізделген. Жүйе бір үлкен топтама негізінде қарастырылмай, бірнеше қадамдардан тұратын даму үдерісі негізінде құрылады. Осындай даму үдерісінің бірнеше ұстанатын талаптамалары бар:

1. Жобаны айрықшалау, жүзеге асыру және әрленімін жасау үдерістері алмасып өтеді. Жобалық құжаттау минимумге дейін жеткізілген әрі жүйенің толық сипаттамасы берілмейді. Тапсырыс берушінің талаптары сипатталған құжат жүйенің тек маңызды өзгешеліктерін атап өтеді.
2. Жүйе бірнеше нұсқалардан құрастырылуы мүмкін. Бұл нұсқалардың құрастырылуы кезінде жүйенің пайдаланушылары және мүдделі тараптар міндетті түрде қатысу қажет. Олар жүйеге қажетті өзгерістерді анықтап, ұсынуы мүмкін.
3. Жүйенің интерактивтік дамуы тура интерфейстің үстіне қажетті нәрселерді салу және орналастыру арқасында жүзеге асырылады. Осыдан кейін жүйе белгілі бір браузерге арналған веб-интерфейсті, әлде кез келген Microsoft Windows платформасына арналған интерфейсті құрастыра алады.

Икемді әдістемелер кезеңдік даму үдерісіне жатады және жүйені бірнеше қадамдардан тұратын бөліктерге бөле отырып құрастырады, сондықтан тапсырыс берушіге жүйенің құрастырылған бөліктері әрбір екі-үш апта сайын тапсырылады. Тапсырыс берушілер жүйенің құрастырылуына үлесін қоса отырып, шапшаң талаптардың өзгерістеріне жауап қайтара алатын мүмкіншілікке ие болады. Құжаттама үдерісі бейресми сұхбат жүзінде өткізіледі.

3.1. Икемді әдістемелер

1980 жылдан 1990 жылдың басына дейін бағдарламалық қамсыздандыруды құру үшін ұқыпты жоспарланған жоба, нысанға-бағытталған сапамен қамсыздандыру, анализ және әрленім әдістерін қолдану қажет деген ой-пікір кең дамыған болатын. Бұл көзқарас ғарышкерлік немесе үкіметтік жүйелер сияқты ауқымды жобаларды құрастырушылардан таралған.

Осы жүйе түрлі компаниялардан тұратын топпен құрастырылған еді. топтар географиялық орналасу жағынан түрлі жерде орналасып, бір жобаға бірталай уақыт жіберетін болған. Мұндай ауқымды жобалардың біріне бүгінгі әуе флотының басқару жүйесін жатқызуға болады, бұл жобаның толықтай құрастырылуына 10 жылдай уақыт кеткен. Осы жоспарлау арқылы басқарылатын әдістемелерде қолданылатын жоспарлау үдерісі, әрленім және құжаттаманы жүргізу жұмысы едәуір шығынға душар етеді екен. Бірақ бұл шығын басқа топтармен бірге

жұмыс жасаған кезде үлкен пайда тигізеді, өйткені жобаға байланысты жасалған құжаттама кейбір мәселелерді шешу үшін едәуір үлесін тигізеді.

Бірақ бұл кешенді жоспарланған жоба үдерісін шағын немесе орташа бизнес жүйелеріне қолданылса, шығатын шығын бүкіл жоба үдерісін тоқтатуы әбден мүмкін. Уақыттың жартысынан көбі жүйені тестілеу мен дамуына емес, болашақ жүйенің жұмыс үдерісін жоспарлауымен өтеді. Жүйелік талаптар өзгерген жағдайда, бүкіл жоба қайтадан қарастырылып, өзгертілуі тиіс.

Соншалықты қиын әдістемелерге қанағаттанбаған жүйе құрастырушылары жаңа «икемді әдістерді» ұсынған болатын. Бұл жаңа әдістемелер жүйенің құжаттамасына және әрленім үдерісіне қарамай, бүкіл назарын жүйенің дамуына аударған. Икемді әдістер кезеңдік даму үдерісіне негізделген. Осындай жобалық әдіс жүйені құрастыру барысында өзгеріп отыратын жобалармен жақсы үйлеседі екен. Осы үдерістің басты мақсаты іске арналған жүйені тапсырыс берушіге ұсынып, пайда болған жаңа жүйелік талаптарды қадаммен енгізу болып табылады. Сондықтан бұл жүйелік құрастыруда құжаттама үдерісі мүлдем алынып тасталған, өйткені бұл жұмыс көп уақыт алуы мүмкін.

Икемді әдістемелер философиясы бүкіл жетекші құрастырушылар келісіп жазылған икемді манифестіне негізделген. Бұл манифест бойынша:

> *Біз бағдарламалық жасақтаудың тиімді жолдарын таба отырып, құрастырамыз. Осындай жұмыстың жасалғанынан кейін бірнеше тұжырымға келеміз:*
>
> *жобалау барысында тұлғалық әрекеттесу үдеріс пен құралдан маңыздырақ;*
>
> *іске арналған бағдарламалық қамсыздандыру орны кешенді құжаттамадан жоғарырақ;*
>
> *табыс берушімен құрылған келісімшарттың орнына ынтымақтастық қарым-қатынас құрған жақсы;*
>
> *жоспарланған жолдың орнына пайда болған өзгерістерге орай әрекет еткен табыстырақ болады.*

Сонымен, оң жақтағы элементтердің құндылығы болса да, біз сол жақ элементтерінің құндылығына қарап бағалаймыз.

Төтенше бағдарламалау, мүмкін, икемді әдістің ең көп таралған түрі болар (Beck, 1999; Beck, 2000). Бұл әдісті осы тараудың барысында әлі де қарастыратын боламыз. Scrum (Cohn, 2009; Schwaber, 2004; Schwaber and Beedle, 2001), Crystal (Cockburn, 2001; Cockburn, 2004), Adaptive Software Development (Highsmith, 2000), DSDM (Stapleton, 1997; Stapleton, 2003) және Feature Driven Development (Palmer and Felsing, 2002) сияқты икемді әдістеменің басқа да амалдары болып табылады. Бұл әдістемелердің табыстылығы арқасында дәстүрлі даму үдерістеріне негізделген жүйені икемді модельдеу әдісі (Ambler and Jeffries, 2002) және икемді Rational Unified Process (Larman, 2002) үлгісі пайда болды.

Қағидат	Баяндама
Тұтынушыны қатыстыру	Әзірлеу үдерісіне тұтынушыларды қатыстырған жөн. Олардың жүйенің жаңа сұраныстарын және келесі итерацияларын анықтауда рөлдері жоғары болады.
Бөліп жеткізу	Бағдарлама бөлініп әзірленеді. Қатыстырылған тұтынушы әр бөлімнің жаңа сұраныстарын анықтайды.
Адамдар үдеріс емес	Әзірлеуші топтың дағдылары икемді ретелу керек. Топ мүшелері өздерінің тәсілдерін ұстануға рұқсатты болу керек.
Өзгерістерді қабылдау	Сұраныстардың өзгеруінің алдын алу үшін, жүйе өзгермелі болып әзірлену керек.
Жарамдылық, қарапайымдылық	Жүйе және әзірлеу үдерісі қарапайым болғаны абзал. Жүйедегі күрделіліктен құтылуға тырысу керек.

3.1-сурет. Икемді тәсілдердің қағидаттары

Бұл икемді әдістемелер түрлі үдерістерді қолдана отырып, кезеңдік даму және жеткізу моделін құрайды. Бірақ осы әдістер икемді манифесті жариялаған тұжырымдарымен сәйкес келеді. Манифестіде жазылған тұжырымдардың бәрі *3.1-суретте* көрсетілген. Түрлі икемді әдістемелер осы тұжырымдарға негізделген, бірақ бүкіл әдістемелерді сипаттау мүмкін емес. Осы кітапты оқу барысында Сіз осы икемді әдістеменің екі түрімен таныс боласыз. Олар: төтенше бағдарламалау (*3.3-тарау*) мен Scrum (*3.4-тарау*).

Осы икемді әдістемелерді қолдану нәтижесінде бірнеше табысты бағдарламалық қамсыздандыру жобалары:

1. Шағын немесе орташа өнім әзірлеу үдерісі;
2. Жүйелік құрастыру барысында тапсырыс берушілер жобалау үдерісіне белсенді қатысатын жобалар құрылған болатын;

Айтып кеткенімдей, бұл икемді әдістердің табыстылығы басқа да бағдарламалық қамсыздандыру жүйелерінің назарын аударған болатын. Бірақ бұл әдістердің тек қана шағын жобаларға арналғандығы ауқымды жобаларды құруға қиындық туғыздырады.

Екіталай инженерлік жүйелерге арналған бірнеше икемді эксперименттер өткізілген болатын (Drobna et al., 2004). Бірақ осындай жүйелерді құру үшін қауіпсіздік, түгелдік және сенімділікке байланысты тестілеудің өту маңызды, осы жерден икемді әдістемелердің әлсіздігі байқалып қалады.

Шын мәнінде, икемді әдістемелер негізделген тұжырымдарды іске асыру қиындық туғызады:

1. Табыс берушілерді бағдарламалық қамсыздандырудың құрылу үдерісіне енгізген жөн көрінеді, бірақ осы үдерісте өз уақытын өткізіп, дұрыс түсіне білетін адамды табудың өзі қиындыққа соғуы мүмкін. Негізінде, табыс берушілердің көбі осындай үдерістен бас тартады.
2. Бағдарламалық қамсыздандыру тобының кейбір мүшелері өз топ мүшелерімен тіл табыса алмай, бір топ ішінде түсініспеушілік пайда болуы мүмкін.
3. Кейбір жүйелік жобаларда пайда болатын өзгерістердің басымдылығын анықтау қиын, өйткені мүдделі тараптар жағынан түрлі көзқарастар шыға келуі мүмкін.
4. Жүйенің жеңілдігін қолдау қосымша уақыт талап етеді. Құрастырушылар жүйені толықтай жеткізу уақыт кестесі бойынша жұмыс жасап, осындай қосымша мүмкіндіктерге қолын жеткізе алмай қалуы мүмкін.
5. Ауқымды ұжымдардың көбі белгілі бір дәстүрлі бағдарламалық қамсыздандыруды құру әрі дамыту үдерісін қолданады. Сондықтан оларға жаңа үдерістермен жұмыс істеуге қиынға түседі.

Кезеңдік дамыту мен жеткізу үдерісі кезінде пайда болатын тағы бір мәселе, ол – жүйеге тапсырыс беруші адам жүйелік дамыту үдерісін басқа бір ұжым көмегімен жүзеге асырайтындығында. Жүйелік талаптар, негізінде, тапсырыс беруші мен тапсырысты орындаушы арасындағы келісімшарттың бір бөлігі болып келеді. Икемді әдістердің өзіндік айырмашылықтарына байланысты осындай бағдарламалық қамсыздандыруды дамыту үдерісіне келісім жасау қиынға түспек.

Сонымен, икемді әдістер бойынша құрылған келісімшартта тапсырыс берушілер тек белгіленген жүйелік талаптардың орындалуына емес, жүйені түгелімен құру мен дамытуға кеткен уақытқа төлеуі тиіс. Осы үдеріс сәтті өтіп жатса, тапсырыс беруші де тапсырысты орындаушы да табысқа жетеді. Бірақ белгілі бір мәселелер пайда болған жағдайда кім кінәлі екенін анықтау өте қиын.

Көптеген кітаптар мен зерттеулерде жаңа жүйелерді құру үшін икемді әдістерді және осы әдістерге негізделген эксперименттерді қолданып сипаттайды. Осы кітаптың 9-тарауында айтып кеткенімдей, бүкіл бағдарламалық қамсыздандыру бүгінгі жүйелерді жетілдіру мен дамуына арналған. Икемді әдістерді қолдана отырып, жүйені жетілдірген үдерістер өте аз (Poole and Huisman, 2001). Осы әдістерді қарастырған жағдайда:

1. Икемді әдісті қолдана отырып құрылған жүйелерге құжаттама жүргізу үдерісін минимумге жеткізіп жетілдіру мүмкін бе, жоқ па?
2. Икемді әдісті қолдана отырып жүйені пайда болған талаптық өзгерістерге сай жетілдіру мүмкін бе, жоқ па? – деген екі сұраққа жауап беру керек болады.

Ресми құжаттама жүйеге түгелдей сипаттама бере отырып, болашақ бағдарламаны қамсыздандыруды құрушылардың жұмысын жеңілтуге бағытталған.

Бірақ жасалған құжаттаманың басым көпшілігі жаңартылмай қалады. Сондықтан икемді әдістің ынтагерлері жақсы әрі сапалы жазылған кодтың рөлін осындай ресми құжаттамадан әлдеқайда жоғары бағалайды. Икемді әдістің маңызды бір ерекшелігі ол – сапалы жазылған әрі жетілдірілген код. Міне, осыған байланысты икемді әдіспен құрылған бағдарламалық қамсыздандыру жобасында ресми құжаттама жүргізудің қажеті де болмас.

Бірақ, менің тәжірибем көрсеткендей ең маңызды жобалық құжаттама – ол жүйелік талаптарды сипаттайтын құжат. Бұл құжаттың көмегімен жүйе құрастырушылары жобаға байланысты ақпарат алуға мүмкіндіктері бар. Осындай ақпаратсыз жүйенің өзгерістеріне жауап қайтару қиынға түседі. Икемді әдістердің көпшілігі осындай талаптық құжаттаманы жасамай, кейбір жүйелік талаптамаларды бейресми әрі кезеңді түрде жинақтайды. Осыған байланысты, икемді әдістемелер жүйені жетілдіру үшін көп қаражат әрі уақыт қажет етеді.

Жүйені дамыту барысында икемді амалды түгелдей қолданбай-ақ, тек жетілдіру үдерісінде икемді әдісті қолданса тиімді нәтижелерге жетуге болады. Кезеңмен дамыту, өзгерістерге байланысты әрленім құру және жеңілдікке жетілдіру – осының бәрі жүйенің өзгерістеріне сай негізделген. Негізінде, осы икемді әдістеме үдерісін бағдарламалық қамсыздандырудың жаңа бір қадамы ретінде қарастырған жөн.

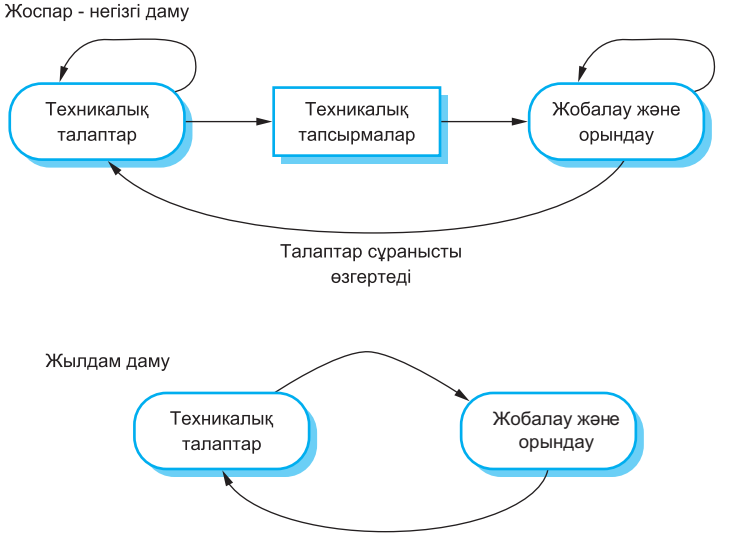
Алайда, бағдарламалық қамсыздандыруды тапсырыс берушіге жеткізгеннен кейін ең басты мәселе – ол осы жүйені құрастыру кезінде қатысқан пайдаланушыларды сақтап қалу. Бағдарламалық қамсыздандыруды дамыту үдерісі кезінде тапсырыс беруші жүйе пайдаланушыларының жұмыс күнін қадағалап негіздей алса, жүйені жетілдіру үдерісі кезінде бұл мүмкін болмайды. Тапсырыс беруші жақтың өкілдері құрылған жүйеге деген көңілі жоғалып қалуы ықтимал. Сондықтан *25-парауда* көрсетілген қосымша әдістерді жүзеге асырып, жаңа жүйелік талаптарды құру керек.

Бағдарламалық қамсыздандыруды дамыту барысында тағы бір пайда болып қалатын мәселе – ол құрастырушы тобының сабақтастығын сақтап қалу. Жүйені құрастырушылар икемді әдісті қолданғанда, жүйені құжаттамасыз түсініп білуі қажет. Егер осы топтың мүшелері жоғалып кетіп қалса, жаңадан келген құрастырушыларға осы жүйені түсініп құрастыру қиынға түседі.

Икемді әдістің ынтагерлері осы әдістің қолданылуын таратып, пайда болған жүйелік проблемаларды орағытып өткен. Осындай әрекет басқа құрастырушылар тарапынан икемді әдістің салдарынан шығатын мәселелерді тереңдетіп жібереді (Stephens and Rosenberg, 2003). Де Марко және Боэм сияқты дәйекті сыншылар икемді әдістердің жақсы мен жаман жақтарын дұрыс бағалай білді (DeMarco and Boehm, 2002). Олар жаңа гибридті ұсынған болатын, бұл әдіс икемді әдістеменің тұжырымдарын қолдана отырып, жоспарлау арқылы басқарылатын дамыту үдерісін қолданады.

3.2. Басқару жоспары және икемді өңдеу

Бағдарламалық қамсыздандыру дамыту кезінде жүйені әзірлеу және жетілдіру қадамдары ең маңызды болып табылады. Осы қадамдар жүйені жетілдіру кезінде жүйенің талаптарын анықтап, тестілеуден өткізеді. Икемді әдістемелерге қарағанда жоспарлау арқылы басқарылатын дамыту кезінде әрбір келесі қадам бұрын өтіп кеткен қадамның нәтижесіне тікелей байланысты. Өтіп кеткен қадамның үдерісі келесі қадамның үдерісін жоспарлап шығады. Икемді мен жоспарлау арқылы басқарылатын әдістемелер арасындағы айырмашылықтар *3.2-суретте* анық көрсетілген.



3.2-сурет. Басқару жоспарымен және икемді әзірлеудің сипаттамасы

Жоспарлау арқылы басқарылатын үдерісте әрбір қадам ресми құжаттамамен сипатталып бекітіледі және бұл құжаттама осы үдерісте өтетін қадамдар арасында қатынасты сақтауға бағытталған. Мысалға айтатын болсақ, жүйелік талаптар құрастырыла отырып, жаңа жүйелік сипаттама енгізіледі. Бұл жүйелік әзірлеу мен жетілдіру үшін бастапқы қадам болады. Икемді әдістеменің қадамдары белгілі бір әрекеттерге байланысты өтеді. Сондықтан жүйелік талаптар мен жүйелік әрленім бірге құрылуы қажет.

Жоспарлау арқылы басқарылатын үдеріс кезеңдік дамыту мен жетілдіру үдерісін қолдайды. Жүйелік талаптарды белгілеу, жүйені әзірлеу мен жетілдіру үдерістерін жеке қадамдар ретінде қарастырған жөн. Икемді үдеріс тек қана кодқа тоғыстырылмаған, ол жүйелік құжаттаманы жүргізуге де бейім. Икемді әдіспен, дамытумен айналасатын құрастырушылар тобы жүйенің жаңа нұсқасын ұсынбай, жүйелік құжаттаманы құрастырады.

Негізінде, бағдарламалық қамсыздандыруды дамыту жобалары икемді әдістер мен жоспарлау арқылы басқарылатын дамыту әдістерінің тәсілдерін қолданады.

Жоспарлау арқылы басқарылатын дамыту және икемді әдістер арасындағы тепе-теңдікті тауып алу үшін бірталай техникалық әрі ұйымдастырушылық сұрақтарға жауап беру керек болады:

1. Жүйенің жетілдіру үдерісіне өту үшін жүйенің толық сипаттамасы мен әрленім жасау қаншалықты маңызды? Егер бұл жүйелік құрастыру үшін маңызды болса, онда жоспарлау арқылы басқарылатын дамыту үдерісі қолданылуы тиіс.
2. Құрастырылып жатқан жүйеңізге кезеңдік дамыту үдерісін қолдануға бола ма, жоқ па? Егер осы дамыту үдерісін қолдануға мүмкіндік болса, онда икемді әдісті қолдануға болады.
3. Жүйеңіз қаншалықты ауқымды? Егер құрастырылып жатқан жоба ауқымды болса, онда жоспарлау арқылы басқарылатын дамыту әдісін қолданған жөн. Керісінше, құрастырылып жатқан жүйе жобасы шағын болса, онда икемді әдістемені қолданған дұрыс болады.
4. Құрастырылып жатқан жүйеңіз қандай? Толықтай келтірілген әрленімі бар жүйені құрастыру алдында едәуір анализ үдерісі жасалуы керек (осындай жүйелер қатарына кешенді уақытқа байланысты жүйелер жатады). Осындай жүйелерді дамыту үшін жоспарлау арқылы басқарылатын дамыту үдерісі қолданады.
5. Жүйенің болжалды қызмет мерзімі қандай? Ұзақ мерзімді жобалар үшін жасалатын ресми құжаттама мөлшері тым үлкен. Алайда, икемді әдістердің ынтагерлері жоба бойынша жасалатын құжаттаманы ешқашан жаңартпайды, сондықтан да мұндай жобалар үшін бұл әдіс қолайсыз болады.
6. Жүйе жұмысын қолдап отыру үшін қандай технологиялар қолданбақ? Икемді әдістер жүйе жұмысын қадағалау үшін жаңа технологияларды қолданады. Егер Сіз жүйенің дұрыс сипаттамасын және анализін бере алмайтын Біріктірілген Өңдеу Ортасын (IDE) қолдансаңыз, онда Сізге жобалық құжаттаманы әзірлеуге тура келеді.
7. Құрастырушылар тобы қалай ұйымдастырылған? Егер құрастырушы топ бірнеше шағын әрі бір-бірінен алыс орналасқан топтардан тұратын болса, онда осы топтар арасында қарым-қатынасты сақтау үшін жобалық құжаттама құрастырылуы керек. Бұл үшін Сізге жақсы жоспар құрастыру қажет болады.
8. Қандай да бір мәдени сұрақтар пайда болуы мүмкін бе? Дәстүрлі бағдарламалық қамсыздандыру үдерісі белгілі бір жоспарға байланысты орындалады. Осы үдеріс икемді әдісте қолданылатын мағлұматты қажет етпей, қосымша құжаттамаға негізделеді.
9. Құрастырушы тобындағы жүйе әрленімін даярлаушы мен бағдарлама жасаушылары қандай? Икемді әдістемемен жұмыс жасайтын құрастырушылар жоспарлау арқылы басқарылатын әдістегі құрастырушыларға қарағанда іскерлік қабілеттері жоғарырақ. Өйткені жоспарлау арқылы басқарылатын жобалауда құрастырушылар жүйенің бүкіл сипаттамасын код түрінде жазып шығару керек болады. Егер Сіздің тобыңызда іскерлік қабілеттері төмен

құрастырушылар болса, онда Сізге жүйенің әрленімін құрастырып алу үшін шынайы іскерлерді тауып алуға қажет болады.

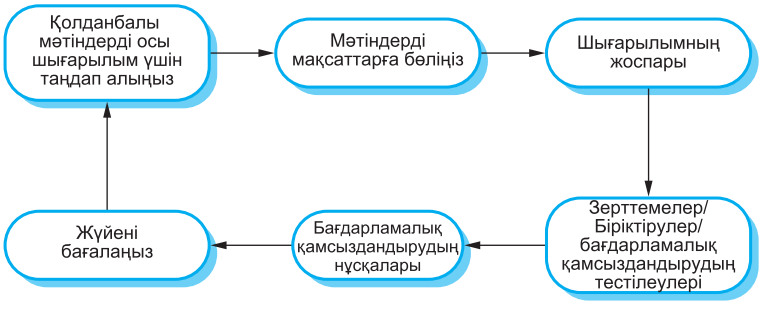
10. Жүйеңіздің сыртқы реттеуге ұшырау ықтималдығы қандай? Егер жүйеңіз сыртқы реттеуге ұшырайтын болса, онда жүйеңіздің толық құжаттамасы жасалуы керек (Федералдық авиация басқармасы авиация үшін жасалатын жүйелерді тексерістен өткізеді).

Негізінде, жүйені икемді немесе жоспарлау арқылы басқарылатын әдістерімен құру мәселесі ондай маңызды емес. Жүйені жасау кезінде осы бағдарламалық қамсыздандыруды пайдаланушылар қалай қолданатыны әрі осы жүйенің пайдалылығын қалай бағалайтыны маңызды. Бүгінгі таңда көптеген компаниялар икемді әдістің тұжырымдарын қолдана отырып жоспарлау арқылы басқарылатын әдістемені жүйелік құрастыруда іске қосады.

3.3. Төтенше бағдарламалау

Икемді әдістердің ішінен ең танымалы әрі көп қолданылатыны, ол – төтенше бағдарламалау. Бұл әдіс атауын Бэк ойлап тапқан, өйткені кезеңдік дамыту үдерісі төтенше деңгейге дейін жеткізілген болатын. Мысалға айтатын болсақ, құрастырылып жатқан бір жүйенің бірнеше нұсқаулары бір күн ішінде түрлі құрастырушылар жасап, тестілеуден өткізген.

Төтенше бағдарламалауда жүйелік талаптар белгілі бір сценарий ретінде (пайдаланушылар оқиғасы деп аталады) қарастырылып, бірнеше амал-тәсілдер арқылы жүзеге асырылады. Бағдарламаны құрастырушылар екі-екіден әрбір амал-тәсілге анализ жасап болғаннан кейін ғана кодты жазуға көшеді екен. Жаңа жазылған бағдарлама коды бүкіл тестілеуден өту керек. Жүйелік өнімдердің шығарылуы арасында біршама аралық болады. Төтенше бағдарламалау бойынша дамытылып жатқан жүйенің жетілдірілуі 3.3-суретте көрсетілген.



3.3-сурет. Төтенше бағдарламалау шығарылымдарының айналымы

Тәжірибе	Баяндама
Бөліп жобалау	Сұраныстардың барлығы тарихи карталарға жазылады. Әзірлеушілер бұл карталарды тапсырма ретінде қарастырады.
Кішігірім шығарылымдар	Пайдалы функцияларының негізгі минимумы әзірленіп, өнім шығарылады.
Қарапайым әрленім	Ағымдағы сұраныстарды қанағаттандыратын әрленім әзірленеді.
Алдын ала тестілеу әзірленімі	Функионалдықты әзірлеудің алдында оны тестілейтін автоматтандырылған тестілер әзірленеді.
Анықтылық	Барлық бағдарламашылар кодты анық жазуы керек. Бұл кодты қарапайым әрі жарамды етеді.
Жұппен бағдарламалау	Бағдарламалаушылар жұппен жұмыс жасайды. Бұл бір-бірінің қателерін дұрыстап, жақсы код әзірлеуге әкеледі.
Топтық меншік	Бағдарлама толығымен топтың меншігінде болады. Сондықтан әр адам әр нәрсені өзгерте алады және барлық бағдарламаға барлық адамдар жауапты болады.
Жалғасатын интеграция	Бір тапсырма орындалған соң, ол толық жүйеге енгізіледі. Осындай енгізуден кейін, жүйе барлық тестілерден өтіп, қайта сыналады.
Тұрақты екпін	Бағдарлама сапасының төмендеуіне байланысты шамадан тыс жұмыс істеу қабылданбайды
Тұтынушының жетімділігі	Әзірленіп жатқан жүйені толығымен пайдаланатын кем дегенде бір тұтынушы әрдайым қолжетімді болу керек.

3.4-сурет. Экстремалдық бағдарламалау тәжірибелері

Төтенше бағдарламалаудың тұжырымдары икемді әдістердің ұстанымдарын қамтып көрсетеді, осы тұжырымдардың сипаттамасы 3.4-суретте айқын көрсетілген:

1. Кезеңдік даму үдерісі жүйенің шағын әрі жиі шығарылымына тікелей негізделген. Жүйелік талаптар тапсырыс берушілердің тілектеріне және көріністеріне сай жасалады.
2. Пайдаланушыны жобалық дамытуға еліктіру үшін осы адамды жүйені жетілдіру барысына үнемі қатыстыру керек. Тапсырыс берушілер жүйені дамыту үдерісіне қатысып, жасалған тестілеудің нәтижелеріне жауапты болады.

3. Жұптық бағдарламалауда ұжымдық меншік, тұрақты жұмыс кестесі және адам арқылы басқарылу үдерісі басты болып саналады.
4. Жүйенің жаңа функционалың үздіксіз ықпалдасуын және құлдырауын болдырмау мақсатында жүйенің тестілеуін жүйелі түрде өткізу керек және осы тестілеудің негізінде құрастыруды жалғастыру маңызды .
5. Жүйенің жеңілдігі тұрақты түрде кодты өзгерту әрі жүйенің қарапайым әрленімін құрастыру арқылы жүзеге асырылады.

Төтенше бағдарламалау кезінде пайдаланушы жүйелік талаптарды анықтау мен тіркеу үдерісіне қатысуға міндетті. Осы талаптар жүйеге қажетті талаптамалар ретінде берілмеуі тиіс. Жүйені пайдаланушы адам осы жүйеге тапсырыс берушіден гөрі құрастырушы тобының мүшесі ретінде қарастырылады. Бірге олар жүйенің «тарихи картасын» құрастырып, тапсырыскерлердің тілектерін жүзеге асырмақ. Құрастырушы топ мүшелері осы жүйе сценарийін келесі шығарылымға дейін жүзеге асыру міндетті. Адамның денсаулық сақтау карточкасының толтыру жүйесі *3.5-суретте* көрсетілген. Бұл дәрімен емдеу үдерісінің қысқа сипаттамасы болып табылады.

«Тарихи карталар» төтенше бағдарламалауды жоспарлау үдерісінде немесе «жоспарлама ойынында» басты қызмет атқарады. Осы карталар пайда болғаннан кейін құрастырушы тобының мүшелері бірнеше шағын группаларға бөлініп, әрбіреуі жүйенің бір-бір мәселесін қарастырып шығарады. Осындай амал жүйенің тапсырыскерлері арасында талқылауды туғызуға және жүйелік талаптарды жақсартуға бағытталған. Осыдан кейін пайдаланушылар керекті талаптарды таңдай отырып бизнесте оң нәтижеге жетуге тырысады. Бұл амалдың негізгі мақсаты – жүйені екі аптаның ішінде толықтай өңдеп пайдаланушыға шығару.

Әрине, жүйелік талаптардағы өзгеріс кейбір қамсыздандырылмаған жүйелік сценарийлерді жою мүмкін. Егер шығарылымға түсіп кеткен жүйелер үшін жаңа талаптар тіркелген болса, онда пайдаланушы жаңадан құрастырылған жүйелік карталардың артықшылығын бағалау керек.

Кейде жоспарлама ойыны кезінде кейбір жауап беруге қиынға түсетін сұрақтар пайда болып, осы сұрақтарға жауап іздеу үшін қосымша жұмыс керек болып қалады. Құрастырушы топ осы мәселелерді шешу мақсатында жүйенің әрленімін жасап, дамыту үдерісінің сынағын өткізіледі. Төтенше бағдарламалауда бұл қадам бағдарламалауды қажет етпейтін кезеңге жатады. Бұл уақытта жүйенің архитектурасы құрастырылып, әрленімі мен құжаттамасы жасалуы тиіс.

Төтенше бағдарламалау кезеңдік дамытуды төтенше нұсқаға дейін жеткізіп жүзеге асырады. Жүйенің жаңа нұсқалары әр күн сайын ұсынылып, әрбір екі апта сайын шығарылымға түседі. Шығарылым мерзімі ешқашан өз мерзімінен таймайды, егер дамыту барысында кейбір мәселелер пайда болса, онда пайдаланушымен талқылау жүргізіліп, кейбір жүйелік қызметтер құрастырудан шығарылады.

Құрастырушы жүйенің жаңа бір нұсқасын шығару үшін бүкіл тестілеу үдерісінен өткізіп алып, жаңа жүйелік қызметтерді ұсыну қажет. Жүйенің жаңа нұсқасы бүкіл тестілеуден сәтті өткеннен кейін ғана шығарылымға түседі. Бұл жүйелік нұсқа келесі кезеңдік дамыту үшін негіз ретінде алынады.

Дәстүрлі бағдарламалық қамсыздандыруды дамыту үдерісінің басты мақсаты – ол жүйелік өзгерістерге дайын болу. Жүйеде болып қалатын болашақ өзгерістерді есепке ала отырып, өз жүйеңіздің сипаттамасын мен талаптарын дұрыс құрастыра білу қажет. Бірақ төтенше бағдарламалау әдісін қолданатын құрастырушылар осындай жолмен бағдарламаны құрастыру үшін көп уақыт кететіндіктен осы тұжырымнан бас тартқан болатын. Пайда болатын өзгерістерге дұрыс жауап қайтара алу үшін бүкіл жүйенің тұтастылығын құру түкке тұрмайтын үдеріс. Жүйеде пайда болған өзгерістерді іске асырғанмен соңында болжалды нәтиже алу анық емес. Төтенше бағдарламалау осындай өзгерістердің пайда болу ықтималдығын мойындайды және осы өзгерістер анықталғаннан кейін жүйені өзгертуге бейім.

Кезеңдік дамыту барысында пайда болатын мәселелердің ең бастысы – ол бағдарламалық қамсыздандыру құрылымын бұзу үдерісі болып табылады, сондықтан да жүйеде пайда болатын өзгерістерге жауап қайтару одан бетер қиын болып кетеді. Жүйенің дамыту үдерісі пайда болатын жүйелік мәселелерді жаңа кодты енгізу арқылы орағытып өтеді, бірақ жүйенің кейбір бөлімдері дұрыс қолданылмай қалса, жүйелік құрылым бұзу үдерісіне ұшырайды.

Төтенше бағдарламалау үдерісі бұл мәселені жүйеге тұрақты өзгерістер жасау арқылы шешкен болатын. Жүйені жақсарту мақсатында құрастырушы топ бүкіл мүмкіншіліктерді қарастыра отырып, оларды жүзеге асыруға тырысады. Жүйені құрастырушы топ мүшесі жетілдіруге қолайлы код бөлшегін көрген жағдайда оны ешбір қажеттіліксіз жетілдіре береді. Осындай өзгерістер қатарына кодтың класстық құрылымын қайта жасау, дубликаттарды жою және атрибуттар мен әдістердің аттарын өзгерту құбылыстары жатқызылады. Eclipse (Carlson, 2005) сияқты бағдарламаны жасақтау ортасында осындай өзгерістерді жүзеге асыру үшін кодта жаһандық өзгерістер жасау үшін және әртүрлі код бөлшектері арасындағы тәуелділікті барлау үдерісін оңайлату үшін арнайы құралдар енгізілген.

Негізінде, құрастырылып жатқан бағдарламалық қамсыздандыру әрдайым түсінуге оңай әрі пайда болған өзгерістерге сай жетілдіруге мүмкіндік беруі қажет. Бірақ бұл тұжырым міндетті емес. Кейде жүйенің өзгерістерін жүзеге асыру үдерісінің тоқтатылуы жаңа жүйелік амал-тәсілдердің пайда болуымен сипатталады. Кейбір жаңа жүйелік амал-тәсілдердің жүйенің бағдарламалық жолымен жетілдірілуі мүмкін емес, сондықтан жүйенің құрылымы өзгеріске ұшырауы мүмкін.

Бүгінгі таңда көптеген компаниялар төтенше бағдарламалаудың *3.4-суретте* келтірілген тұжырымдарын бәрін қолдана бермейді. Осы тұжырымдар жергілікті жұмыс істеу принциптеріне сай таңдалады. Мысалы, кейбір компаниялар жұптық бағдарламалау әдісін жөн көреді, ал басқа компаниялар жеке бағдарламалау әдісіне негізделген. Бағдарламаны құрастырушылар өзі жасамаған код бөлшегін өзгерту жұмыстарын жасамайды және бұл жағдайда әдеттегі жүйелік талаптар қойылуы қажет. Бірақ, төтенше бағдарламалау әдісін қолданған көптеген компаниялар шағын шығарылымдарды, тестілеу негізінде жасалған дамытуды және үздіксіз ықпалдасу үдерістерін қолданған болатын.

3.3.1. Төтенше бағдарламалауды тестілеу

Осы тараудың басында айтылып өткендей жоспарлау арқылы басқарылатын және кезеңдік дамытулардың арасында басты айырмашылық – ол тестілеуден өткізу үдерісі. Кезеңдік дамыту кезінде сырттан келген тестілеу тобы жүйеге анализ жасау үшін белгіленген жүйелік сипаттаманы қолданбайды. Сондықтан жасалған тестілеу үдерісі бейресми сипаттама алады. Керісінше, жоспарлау арқылы басқарылатын дамыту кезінде бұл ресми құжаттама жүзінде бекітеледі.

Жүйеге жасалған анализ бен мақұлдау үдерісі кезінде кейбір мәселелерді орағытып өту мақсатымен бағдарлама тестілеуі өткізілуі шарт. Төтенше бағдарламалау ағымдағы жүйеде қателіктердің пайда болмауы үшін тестілеуге арналған тұжырымдарын ұсынған болатын.

Осы тұжырымдардың ерекшеліктері:

1. Тестілеу негізінде дамыту үдерісі;
2. Сценарийге негізделген кезеңдік дамыту;
3. Жүйені дамыту мен тестілеу үдерісіне пайдаланушыларды қатыстыру;
4. Автоматтандырылған тестілеу құрылымдарын пайдалану.

Тестілеу негізінде дамыту – ол төтенше бағдарламалаудың ең маңызды ашылуы болып табылады. Жүйенің бағдарламалық кодынан кейін тестілеу үдерісін жазу орнына бағдарламалық құрастырушылар алдымен жүйені тестілеуден өткізіп, содан соң бағдарламалық кодты жазуға кіріседі. Бұл дамыту үдерісінде пайда болатын қателіктерді тауып алып, реттеуге бағытталған. Тестілеу үдерісі, ең алдымен, жүйенің интерфейсі мен функционалдық қасиетін анықтауға көмектеседі. Жүйені дамыту кезінде пайда болатын түсінбеушілік пен жүйелік талаптарға байланысты мәселелер шешіледі. Бұл үдеріс әрбір жүйелік талаптар мен осы талаптарды бағдарламалау үдерісі арасындағы қарым-қатынас бар жүйеде жүзеге асырылады. Төтенше бағдарламалауда бұл қарым-қатынас айқын байқалады, өйткені жүйенің тарихи карталары бірнеше амалдарға бөліне отырып, әрбір амал ол толық жүйенің жетілдіру үдерісіне әсер етеді. Төтенше бағдарламалауда тестілеу негізінде дамыту әдісі пайда болуы жаңа әдістердің құрылуына негіз болған (Astels, 2003). Бұл әдістер кітаптың 8-тарауында сипатталған.

Осы тестілеу негізінде дамыту үдерісінде бағдарламаны жасақтаушылар жүйелік сипаттаманы толығымен түсініп білуі қажет. Сондықтан жүйелік сипаттаманы құрастыру барысында әрбір тұрақсыздық пен қателіктер шешілуі міндет. Бұдан басқа бұл жүйенің тестілеуден артта қалушылығын шешеді. Жүйенің тестілеуден артта қалушылығы бағдарламалық жасақтаушы тестілеушілерден шапшаңырақ жұмыс жасағанымен түсіндіріледі. Жүйені жетілдіру үдерісі әрі қарай өте беріп тестілеу қадамын өтіп кетуі ықтимал, сондықтан жүйенің жобалық кестесі дұрыс жасалуы қажет.

Емделудің ресми өкілі

Кэйт - клиникаға келетін науқасқа ем белгілегісі келетін дәрігер. Емделушілер жайлы есептеме оның компьютерінде көрсетілген, сөйтіп, ол емдеу тұсына басады да 'ағымдағы ем', 'жаңа ем' немесе 'формуляр' керектісін таңдайды.

Егер ол 'ағымдағы ем' таңдаса, жүйе одан мөлшерді тексеруді сұрайды. Егер ол мөлшерді өзгерткісі келсе, ол керекті мөлшерді енгізіп орындау туралы нұсқаны растайды.

Егер ол 'жаңа ем'-ді таңдаса, жүйе оның қандай емді белгілеу керектігін біледі деп болжайды. Ол алғашқы бірнеше препарат атауының әріптерін басады. Жүйе сол әріптерден басталатын, мүмкін препарат тізімін көрсетеді. Ол керекті емді таңдайды, және жүйе одан таңдалған емнің дұрыстығын тексеруін сұрайды. Ол мөлшерді енгізіп, бұйрықты растайды.

Егер ол 'формуляр' таңдаса, жүйе іздеу терезесін мақұлданған формуляр үшін көрсетеді. Сонда ол қажетті препаратты іздей алады. Ол препаратты таңдайды және одан емнің дұрыстығын тексеруді сұрайды. Ол мөлшерді енгізіп, бұйрықты растайды.

Жүйе қашанда мөлшердің мақұлданған диапазоны бойынша тексереді. Егер олай болмаса, Кейттен мөлшерді өзгертуді сұрайды.

Кейт бұйрықты растағаннан кейін, ол тексеріліс үшін көрсетіледі. Ол керектісін 'Растау немесе Өзгерту' басады. Егер ол 'Растау' болса, бұйрық бақылау дерекқорында тіркеледі.

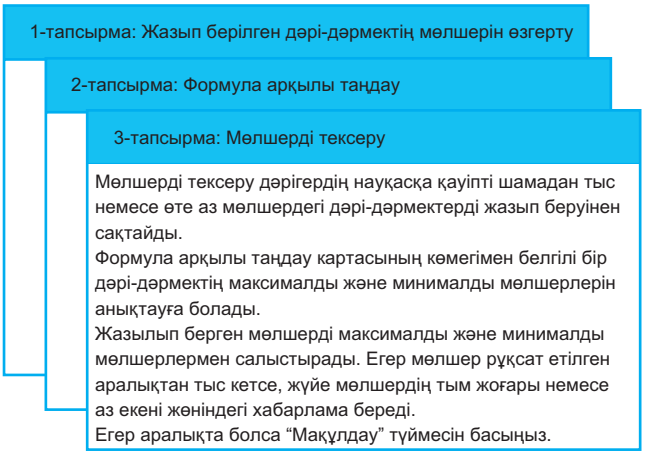
3.5-сурет. Медициналық дәрі-дәрмекті жазып беру оқиғасы

Төтенше бағдарламалауда пайдаланушы талаптары белгілі бір сценарий ретінде сипатталып жетілдіріледі. Әрбір бағдарламаны жасақтаушы топ осы сценарийлерді бірнеше қадамдарға бөле отырып дамытады. Мысалы, кейбір медициналық карточкалар бойынша (*3.5-сурет*) жасалған дәрімен емдеу жүйесі *3.6-суретте* көрсетілген. Әрбір жасалған қадамда тестілеу жүргізіледі. Дәрімен емдеуге берілген дәрілердің қауіпсіздік мөлшері қысқаша сипатталған (*3.7-сурет*).

Тестілеу кезіндегі пайдаланушының міндеті – ол жүйенің тестілеу үдерісіне қатыса отырып, келесі қадамдар үшін тестілеу үдерісін анықтау. *8-тарауда* айтылғандай жүйенің тестілеу үдерісі, ең алдымен, пайдаланушы талаптары мен тілектерінің қаншалықты орындалғандығын тексереді.

Төтенше бағдарламалауда тестілеу үдерісі де қадамдық жолмен іске асырылады. Пайдаланушы жасақтау тобының мүшесі ретінде дамыту жүргізіліп жатқан мезетте тестілеу үдерісін сипаттап дайындайды. Бүкіл жаңадан жазылған бағдарламалық код пайдаланушының талаптарының сәйкестігіне тексеріледі. *3.5-суретте* көрсетілген диаграммада дәрімен емдеу үдерісінің тестілеу үдерісі көрсетілген, мұнда (а) дәрі мөлшері өзгертілген жағдай, (ә) жаңа дәрі белгіленген жағдай, (б) фармакологиялық анықтамалық қолданған жағдай. Негізінде, бірнеше тестілеу үдерістері бір үлкен тестілеуден маңыздырақ болады.

Пайдаланушың тестілеу үдерісінде қатысуы төтенше бағдарламалаудың ең қиын әрі кешенді қадамы ретінде саналады. Өйткені жасақтау тобымен толықтай жұмыс жасалмай, тағайындалған уақыттың көбі пайдаланушылармен жұмыс жасаумен кетеді. Бірақ кейбір пайдаланушылар жүйеге деген талаптарын білдіргеннен кейін әрі қарай осы жүйенің жасақтау үдерісіне қатысуға көңілдерін бөлмейді.



3.6-сурет. Дәрі-дәрмекті жазып беру карталарының мысалдары

Автоматтандырылған тестілеу осы тестілеу негізінде дамыту үдерісі үшін өте маңызды болып келеді. Жүйелік тестілер амалды құрастыру қадамына дейінгі уақытта өткізіліп, шағын орындалатын әмір ретінде жазылуы тиіс. Осы тестілеу компоненттері дербес түрде құрастырылып, соңында жүйелік талаптарға сәйкестігіне тексереді. Автоматтандырылған тестілеу жүйесі – ол тестілеуді орындайтын компоненттері арқылы жүйені тексерістен өткізеді. Junit (Massol and Husted, 2003) – бұл аты шулы автоматтандырылған тестілеу жүйесінің бір түрі.

Тестілеу автоматтандырылғаннан кейін тестілеу үдерісі тест жиынтығынан құрастырылып жүзеге асырылады. Жүйеге қандай да бір жаңа амалдар қосылғаннан кейін әрбір жаңадан енгізілген функция осы тест жиынтықтары арқылы тексеріліп қателіктер анықталуы мүмкін.

Тестілеу негізінде дамыту және автоматтандырылған тестілеу үдерістері негізінде тестілеудің бірнеше түрлері жасалып орындалады. Бірақ осындай әдіс бағдарламалық тестілеудің ұқыптылығына кепіл болмайды. Бұған үш түрлі себеп бар:

1. Бағдарламаны жасақтаушы тестілеуден гөрі бағдарламаны жасақтау үдерісіне көп мән білдіреді. Мысалға, олар толық тесті жасамай, кейбір қателіктерді байқамай қалуы мүмкін.
2. Кейбір тестерді қадамдық жолмен жасаған қиын. Мысалы, кешенді пайдаланушы интерфейстің жұмыс үдерістерін көрсету қиынға түседі.
3. Тест жиынтығының толықтығын бағалау ауырға түседі. Жүйені тестілеуге арналған тест жиынтығында көптеген тестердің болғанымен, ол жүйені толықтай қамти алмайды. Жүйенің маңызды бөліктері тестілеуден өтпей қалуы мүмкін.

4-тапсырма: Мөлшерді тексеру

Енгізілетін ақпарат:
1. Бір рет қабылданатын дәрі-дәрмектің мөлшерін анықтайтын мг саны
2. Дәрі-дәрмектің бір күнде қабылданатын саны

Тестілер:
1. Енгізілген ақпараттағы дәрі-дәрмектің мөлшерін дұрыс енгізіп, ал қабылдайтын жиілігін тым жоғары ету
2. Енгізілген ақпараттағы дәрі-дәрмектің мөлшерін тым жоғары немесе тым аз ету.
3. Енгізілген ақпараттағы дәрі-дәрмектің мөлшерінің қабылдау жиілігіне көбейтіндісін тым жоғары немесе тым аз ету
4. Енгізілген ақпараттағы дәрі-дәрмектің мөлшерінің қабылдау жиілігіне көбейтіндісін рұқсат етілген аралықта енгізу.

Шығатын ақпарат:
Дұрыс (OK) немесе қате (error). Соңғысы енгізілген мөлшерлердің аралықтан тыс болған жағдайында көрсетіледі.

3.7-сурет. Мөлшерді тексеретін тесттік сипаттама

Сондықтан кейбір тестілеу жинақтары жүйенің толықтығы мен дұрыстығын көрсете отырып, оның кейбір қателіктерін көрмей қалуы әбден мүмкін. Егер осы тест жиынтығы әрі қарай қолдана берсе, онда бағдарламалық жүйе қателіктерге толы болып шығарылады.

3.3.2. Жұптық бағдарламалау

Төтенше бағдарламалауда тағы бір маңызды әдістердің бірі – жұптық бағдарламалау, ол бағдарлама жасақтаушылардың екі-екіден жұмыс жасағанына негізделген. Олар бірге бір компьютер алдында отырып, бір жүйені жасақтаумен айналысады. Бірақ бір жұп бағдарламалық кодты бірге құрастырмайды. Бұл жұптар динамикалық түрде таңдалады, сондықтан бүкіл топ мүшелері бір-бірімен жұмыс істеп шығады.

Осындай бағдарламалық жасақтау әдісінің төмендегі ерекшеліктері бар:

1. Ұжымдық меншік пен жауапкершілік тұжырымы қолданады. Бұл Вейнбергтің тұжырымына (1971) қарсы шығады, ол бағдарламалық жасақтама бүкіл топтың меншігі болып есептеледі, сондықтан жеке бір топ мүшесі белгілі бір жүйенің бөліміне жауап етпейді. Бүкіл топ мүшелері қандай да жүйелік қателіктерге бірге жауап беруі қажет.
2. Бағдарламалық код екі адам көмегімен қарастырылатындықтан, бұл үдеріс бейресми сипаттама алады. Осы кодтың жиі және шұғыл тексерісі (*жиырма төртінші тарауда көрсетілген*) жүйелік қателіктерді сәттілікпен анықтауға көмектеседі. Бірақ бұл үдерістер әжептәуір уақытты алатындықтан бағдарламалық жасақтаудың дамыту үдерісін тежейді. Жұптық бағдарламалау бейресми үдеріске жататындықтан, бұл әдісті жүзеге асыру арзанға түседі.

3. Жүйелік жақсарту үдерісіне жататын жүйені өзгерту құбылысын жүзеге асырады. Бірақ бұл әдістеме қалыпты дамыту жүйесінде едәуір уақытты талап етеді. Жүйені жасақтаушы топ ішінде бағдарламаны жасақтаушы адам осы бағдарлама талаптарының өзгерістерін жүзеге асыратын адамнан жоғары бағаланады екен. Ұжымдық меншік пен жұптық бағдарламалау бар жер осы жүйелік өзгерістерге жауап қайтару үдерісін қолдайды.

Сіз жеке бағдарламалық жасақтауды жұптық бағдарламалаудан пайдалы деп ойлап қалуыңыз мүмкін. Берілген уақыт ішінде жасақтаушылардың бір жұбы жеке жасақтаушыларға қарағанда жартыдан көп жұмысты орындап шығады. Осы бағдарламалау әдісі жан-жақтан қарастырылып зерттелген болатын. Уилиамс пен оның қызметкерлері (Cockburn and Williams, 2001; Williams et al., 2000) студенттерді түрлі эксперименттерге қатыстыра отырып, жұптық бағдарламалау әдісін жеке бағдарламалаумен салыстыруға келетіндігіне көзі жеткен болатын. Жұптық бағдарламалауда жасалатын жұмысты алдын ала талқылаудан өткізіп, пайда болатын өзгерістер мен қателіктерді орағытып өткеннен кейін ғана жұмысты бастайды. Осыған қоса осындай бейресми жұптық талқылаудан кейін анықталған қателіктер мөлшері тестілеу үдерісі кезінде табылған қателіктерге қарағанда орасан зор.

Бірақ тәжірибелі жасақтаушылармен жасалған эксперименттер осындай нәтижелерге келмеген болатын (Arisholm et al., 2007; Parrish et al., 2004). Олар жұптық бағдарламалау нәтижесінде жүйенің бағдарламалық жасақтау өнімділігі төмендегенін байқады. Осындай бағдарламалық әдістеменің жақсы жақтары да аңғарылған, бірақ олар толықтай жобаны жетілдірмеген болатын. Бірақ жұптық бағдарламалау кезінде өтетін мағлұмат айырбасы өте пайдалы, өйткені кейбір топ мүшелері жұмыстан кеткеннен кейін қалған топ мүшелері үшін жүйені түгелімен танып біліп алу қиындыққа түспейді. Осыған байланысты жұптық бағдарламалау өте пайдалы болып шығады.

3.4. Жобаны икемді түрде басқару

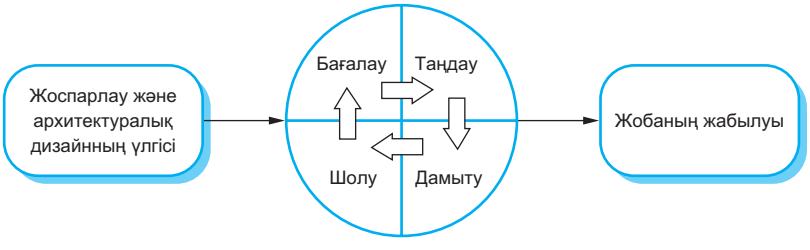
Бағдарламалық жасақтау жүйе басқарушысының басты міндеті – ол осы жобаны белгіленген қаржы мен мерзімде тапсырыс беруші қолына жеткізу. Олар бағдарламалық жасақтаушылардың жұмысын тексере отырып, бағдарламалық жасақтаманың дамыту үдерісінің өту қалпын қадағалайды.

Жобаны басқару үдерісі жоспарлау арқылы басқарылатын дамыту үдерісіне тікелей негізделген. *23-парауда* айтып кеткенімдей жобаны басқарушы адамдар жобаны ұсыну алдында жүйенің жасалу мерзімін, қандай бағдарламалық жасақтама жасалатынын және осы жобаны құрастыратын адамдарды ұсынады. Жоспарлау арқылы басқарылатын әдісінде басқарушы адамның болуы міндет, өйткені осы адам бүкіл жобаға шолу жасайды. Бірақ икемді әдістемеде жүйелік талаптар қадамдық үлгімен жүзеге асырылып, бағдарламалық жасақтама шағын

уақыт ішінде шығарылымға ұсынылып және жүйелік талаптардағы өзгерістердің пайда болуына үйреншікті болғандықтан бұл әдісті қолдануға болмайды.

Көптеген бағдарламалық қамсыздандыру дамыту үдерістері икемді әдісті қолдана отырып, жүйенің шығарылым уақытын және қаржысын белгілеп алуы қажет. Кезеңдік даму үдерісін және икемді әдістің кейбір артықшылықтарын қамтитын жаңа жобаны дамытатын әдіс қажеттілігі артты.

Scrum әдістемесі (Schwaber, 2004; Schwaber and Beedle, 2001) – бұл икемді әдістердің бір түрі болып келеді, бірақ бұл әдіс кезеңдік дамыту үдерісіне негізделген әрі икемді әдістің кейбір тұжырымдарын қолдайды. Осы әдістің басқару жүйесі 3.8-суретте көрсетілген. Scrum әдістемесі жұптық бағдарламалау мен тестілеу негізінде жасалатын дамыту жүйесі сияқты бағдарламалық кодты қолданбайды. Сондықтан төтенше бағдарламалау сияқты техникалық икемді әдісті қолдана отырып, жобалық дамытудың негізін құрастыруға болады.



3.8-сурет. Scrum әдістемесі

Scrum әдістемесі үш кезеңнен тұрады. Барлық жүйелік талаптар анықталып жүйе құрылымы жоспарлау қадамында құрастырылады. Осыдан кейін жүйе шапшаң үдерістер арқылы дамытылады. Ең соңында жүйенің толық сипаттамалық құжаттамасы жасалады.

Scrum әдісінде қолданатын шапшаң өтетін үдерістер осы әдістің ең танымал ашылуы ретінде қарастырылады. Осы шапшаң үдерісі барысында жүйені жетілдіру қадамының жоспарлауы өткізіледі, осыдан кейін жүйені жасақтау кезеңі басталады. Үдерістің соңына қарай жүйені құрастыру аяқталып, тапсырыс берушіге тапсырылады. Осы үдерістің басты ерекшеліктері:

1. Шапшаң үдерістер екі немесе төрт аптаға дейін созылады. Осындай дамыту уақыт аралығы төтенше бағдарламалау арқылы дамытумен бірдей.
2. Жоспарлау қадамы аяқталмаған жұмыс тізімінен басталып, болашақ жұмыс тәртібін анықталады. Жүйені бағалау барысында жүйенің кейбір қауіпқатерлері анықтала отырып, жаңа талаптар орнатылады. Осы үдерістер барысына пайдаланушыларды қатыстыру маңызды.
3. Ал таңдау кезеңі барысында пайдаланушылардың тілектерін мен талаптарын есепке ала отырып, жүйенің жаңа амал-тәсілдері енгізілуі мүмкін.

4. Осының бәрі дайын болғаннан кейін ғана жасақтау тобы бағдарламаны жасақтау үдерісіне кіріседі. Күнделікті қысқа кездесулер кезінде жүйенің даму үдерісі талқыланып керекті өзгерістер енгізілуі мүмкін. Бұр үдеріс барысында жасақтаушы топ пайдаланушы мен компаниядан алшақтанып, тек қана Scrum-мастер арқылы қарым-қатынаста бола алады. Scrum-мастердің негізгі мақсаты – ол жасақтаушы топты сыртқы аландатқыш факторлардан сақтау. Жұмыстың атқарылу жолы тікелей пайда болған мәселе мен топқа байланысты. Төтенше бағдарламалауға қарағанда, Scrum әдісі талаптарды қалай анықтау және құжаттаманы қалай жүргізу керектігіне кеңестерін бермейді екен. Бірақ төтенше бағдарламалаудың кейбір амалдары қолданылуы да мүмкін.
5. Бір шапшаң үдеріс аяқталғаннан кейін орындалған жұмысты тапсырыс беруші қолына тапсырылып, жаңа үдеріске көшу басталады.

Scrum әдісі жасақтаушы топ ішінде бір жобалық басқарушыны таңдамай, бүкіл топ жобаны дамыту үдерісін қалай басқару керектігіне үйреніп алуы қажет. Күнделікті кездесулерді ұйымдастыратын, жасалатын жұмыс тізімін құрастыратын, шешімдерді қабылдайтын, жұмыс тізіміндегі прогрессті анықтайтын әрі сыртқы ортамен қарым-қатынасты орындайтын адам, ол – Scrum-мастер болып табылады.

Осы күнделікті шағын кездесулерге бүкіл топ қатысу міндетті. Кездесу барысында әрбір топ мүшесі орындалған жұмыс прогресін, туған қиыншылықтарын және келесі күнге деген жоспарын сипаттайды. Осындай кездесулерден кейін әрбір топ мүшелері өтіп жатқан жұмыстардың барысын танып біледі. Қысқа уақытқа жасалатын жоспарлауда топтың әрбір мүшесі қатысады.

Интернет жүйесі осы Scrum әдісі қолданылған сәтті эксперименттер туралы ақпаратқа толы. Ризин мен Янов (2000) телебайланыс ортасына арналған бағдарламалық жасақтама құру кезінде осы әдісті қолданған кезіндегі ерекшеліктерін сипаттаған болатын:

1. Өнім бірнеше шағын әрі түсінікті бөліктерге бөлініп басқарылады.
2. Жүйенің жетілдіруіне өзгермелі талаптар кедергі келтірмейді.
3. Бүкіл топ барлық жақтардан көріне отырып, топ ішіндегі қарым-қатынас жақсарады.
4. Тапсырыс берушілер жұмыстың орындалып жатқан үдерісін көре отырып, тапсырысты уақытында алады.
5. Жүйені жасақтаушы мен тапсырыс беруші арасында сенімділік орнатылып, жобаның сәттілігіне жол ашылады.

Scrum әдісі, негізінде, бір-біріне географиялық орналасу жағынан жақын және күнделікті кездесулерге қатыса алатын топтарға арналған. Бірақ бүгінгі күні бағдарламалық жасақтау жобалары географиялық жағынан бір-бірінен алыс орналасқан топтарды құрып жұмыс жасайды. Сондықтан Scrum әдістемесінің осындай жұмыс істеу принципіне негізделген амалдарды ұсынады (Smits and Pshigoda, 2007; Sutherland et al., 2007).

3.5. Икемді әдістемелерді ауқымдау

Икемді әдіс кішкентай бөлме ішінде жұмыс жасап бір-бірімен бейресми түрде қарым-қатынас жасай алатын шағын топқа арналған. Сондықтан бұл әдіс шағын және орташа жобаларда қолданылады. Әрине, ауқымды жобалар үшін өнімнің шапшаң шығарылымы мен тапсырыс берушілердің талаптарын орындайтыны қажет. Міне, осыған байланысты икемді әдістерді үлкен жобаларға қолдану мақсатында ауқымдау үдерісі басталды.

Деннинг пен басқалар (2008) бағдарламалық жасақтауды дамыту кезінде пайда болатын тапсырыс берушінің талаптарын орындамау мен қаражаттың тым көп кету мәселелерін икемді әдістердің ауқымды жобаларға қолдану жолдарын тауып алғаннан кейін шешуге болатынына көз жеткізді. Левингвелл (2007) икемді әдістердің қайсысы ауқымды жобаларға келетіні туралы сұхбат жасаған. Мур мен Спенс (2008) түрлі географиялық жерлерде орналасқан 300 бағдарлама жасақтаушы қатысқан ауқымды медициналық жүйенің қалай құрастырылғаны туралы хабар жасаған.

Ауқымды жүйелердің шағын жүйелерден айырмашылықтары:

1. Ауқымды жүйелер бірнеше шағын бөлімдерден құрастырыла отырып, әрбір бөлім белгілі бір жасақтаушыға тапсырылады. Көбінесе осы жасақтаушы топ мүшелері бір-бірінен алыс жерлерде орналасқан. Осыған байланысты бүкіл жүйенің толықтай түсініктемесін алу қиыншылыққа соғады. Әрбір жасақтаушы топ мүшесінің міндеті – ол жүйенің өз бөлімін аяқтап бітіру.
2. Ауқымды жобалар, негізінен, «күтімсіз қалып қалған жүйелер» (Hopkins and Jenkins, 2008) деп аталады, сондықтан олар іске қосылған жүйелермен қосылып құрастырылады. Осындай әрекеттесу көптеген жүйелік талаптарды қажет етеді, сондықтан жүйенің икемділігі мен кезеңмен даму үдерісі жойылады. Көбінесе, осы мәселелерді шешу мақсатында іске қосылған жүйелерді өзгертуге тура келеді. Бірақ осы үдерісті жүзеге асыру үшін жүйе басқарушысымен келіссөздер жүргізу керек болады.
3. Егер бірнеше жүйелер бірігіп, жаңа бір жүйені құрастыратын болса, онда дамыту үдерісінің көп уақыты осы жүйелердің бағдарламалық кодын жазбай реттеуге кетеді. Осындай үдеріс кезеңдік даму мен жүйемен жиі ықпалдасу үдерістерімен сай келмейді.
4. Ауқымды жүйелерді дамыту барысында сырттан көңіл бөлініп, белгілі бір ережелерге сай болуы шарт болып қалады, сондықтан жүйелік құжаттаманы жасау міндетті қадамға айналады.
5. Ауқымды жобаларды жетілдіру мен дамыту көп уақытты талап етеді. Осындай ұзақ уақытқа созылатын жобаларды құрастырушылар осы жұмыстан басқа жерлерге кететіндіктен бүкіл жобаның жұмыс барысын сақтау қиынға түседі.
6. Ауқымды жобалар алуан түрлі мүдделі тараптар жиынын құрайды. Мысалы, медициналық жүйеде мейірбике мен администраторлар осы жүйенің пайдаланушылары болып саналады, бірақ департамент басқарушылары мен жүйе

жобасының басқарушылары да осы жүйенің мүдделі тараптар қатарына жатады. Осы адамдардың бәрін жүйенің дамыту үдерісіне қарастыру мүмкін емес.

Икемді әдістерді ауқымдаудың екі жолы бар:

1. «Кеңейту» үдерісі тек қана ауқымды жүйелерді құрастыру үшін қолданылады және шағын жасақтаушы топ арқылы құрастырылмайды.
2. «Ауқымдау» үдерісі икемді әдістердің үлкен әрі тәжірибелі компанияларда қалай қолданылатының сипаттайды.

Икемді әдістерді ауқымды жүйелерді жасақтауға қолданылатындай жетілдіру керек. Левингвелл (2007) икемді әдісте қолданылатын икемді жоспарлау кезеңін, жиі өнім шығарылымын, тестілеу негізінде дамытуын және топ арасында жақсы қарым-қатынастың жетілдіруін маңызды деп сипаттаған. Осы әдісті жетілдіру барысында мына өзгерістер енгізілуі қажет:

1. Ауқымды жүйелерді дамыту барысында тек қана бағдарламалық кодқа негізденбеу керек. Сіз жүйенің әрленімін жасап құжаттамалық жұмысты жүргізуіңіз керек. Бағдарламалық жасақтама жүйесінің құрылымы жасалып дерекқор сызба мен диаграммаларын сипаттайтын құжаттама енгізілуі қажет.
2. Топ аралық қарым-қатынасты орнықтыратын амал-тәсілдер енгізілу керек. Ол үшін күнделікті кездесулер, телефон және бейне конференцияларды ұйымдастыру қажет. Жасақтау топ мүшелері арасындағы қарым-қатынас электрондық пошта, жылдам хабарлама жүйесі және әлеуметтік торап жүйесі арқылы жүргізілуі қажет.
3. Бірнеше шағын жүйелерді бір жүйеге ықпалдастыру үдерісі кезінде жүйедегі талаптар өзгерістеріне жауап қайтару қиын. Сондықтан жүйені жиі жетілдіріп шығарылым мерзімдерінің аралықтарын қысқарту қажет. Осыған байланысты бірнеше топтар жұмысын қолдайтын басқару үдерісі жасалуы қажет.

Икемді әдістерді жиі қолданылатындарға шағын бағдарламалық жасақтау компаниялары жатады. Осы компаниялар қандай да бір ережелерге негізделмеген әрі жаңа идеялар мен талаптарға ашық. Әрине, ауқымды компаниялардың көбі жобаларды дамыту барысында икемді әдістерді қолданып, эксперименттер жасаған болатын, бірақ бұл үдерістерді бүкіл компания ішінде ауқымдау қиынға түспек. Линдвалл және басқалары (2004) төрт ауқымды технологиялық компаниялар ішінде икемді әдістерді ауқымдау үдерісін жүргізгенде пайда болған мәселелерді сипаттап берген.

Икемді әдістерді ауқымды компанияларға енгізудің келесі қиыншылықтары туады:

1. Икемді әдіспен жұмыс істеу тәжірибесі жоқ жобалық басқарушылар осы әдіспен жұмыс жасауға тез келіспейді, өйткені осы үдеріс барысында пайда болатын мәселелерді шешу жолдарын білмей қалады.
2. Ауқымды компаниялардың көбі кейбір өнімдерді жасақтауға байланысты өзіндік ережелері мен стандарттары бар, осыған байланысты икемді әдістер осы ережелер мен стандарттарға сәйкес келмей қалуы да мүмкін. Кейде икемді әдістер кейбір бағдарлама жасақтау құралдарды қажет етеді.
3. Бағдарлама жасақтаушы тобы тәжірибелері жоғары әрі өз ісін білетін адамдардан тұрса, онда икемді әдістің нәтижелері сәттілікке әкеледі. Бірақ үлкен компаниялар ішінде әртүрлі адамдар жұмыс істейді және тәжірибелері төмендеу жасақтаушы топтар да құрылып қалуы мүмкін.
4. Кейбір ауқымды компаниялар ішінде дәстүрлі дамыту үдерістері бар, сондықтан икемді әдісті қолданған кезде түсініспеушілік пайда болуы мүмкін.

Өзгерістерді басқару және тестілеу үдерістері – бұл икемді әдістемелерді қолдануға келмейтін мысалдарға жатады. Өзгерістерді басқару үдерісі пайда болатын өзгерістерді реттейді, сондықтан осы өзгерістерді алдын ала болжап шешуге болады. Жүйеде пайда болатын өзгерістерді жүйеге енгізу алдында тексерістен өткізілуі керек, осы үдеріс жүйені өзгерту құбылысына қарсы шығады. Төтенше бағдарламалауда жасақтаушы ешбір сырттай тексеріссіз барлық бағдарламалық кодты жетілдіріп өзгерте алады. Ауқымды жүйелер де өзіндік тестілеу үдерісін жүргізіп сыртқы тестілеу тобына тексеріске жібереді. Осындай үдерістер төтенше бағдарламалаудың тестілеу негізінде дамыту әрі жиі тестілеу әдістеріне қарсы шығып қалады.

Икемді әдісті ауқымды компаниялар назарына таныстыру мен енгізу үдерісі мәдени өзгеріске әкеледі. Бұл үдеріс көп уақытты қажет етеді, сонымен қатар компанияның ұйымдастыруына өзгерістер енгізеді. Осы икемді әдісті енгізу үшін сәйкес ынтагерлерді тауып алып, осы әдісті компания жұмыскерлері ішінде тарату қажет. Икемді әдістерді компания жұмысына енгізу үшін сәйкесті материалдар қолданылуы қажет. Кітапты жазу барысында кейбір ауқымды компаниялар икемді әдістің даму үдерісіне көшкен болатын.

НЕГІЗГІ ТҰЖЫРЫМДАР

- Икемді әдістер шапшаң дамыту үдерісіне, жиі шығарылым мерзімдеріне, жүйенің жұмыс істеуін жеңілдететіне және жоғары сапалы кодтың жазылуына негізделген кезеңдік дамыту әдістерінің бір түрі болып саналады. Осы әдістер пайдаланушыны дамыту үдерісіне тікелей қатыстыруға тырысады.
- Икемді әдісті немесе жоспарлау арқылы басқарылатын дамыту әдісін қолдану туралы шешім құрастырылып жатқан бағдарламалық жасақтаманың

түріне, компанияның дәстүрлі дамыту үдерісіне және жасақтаушы топтың мүмкіншіліктеріне негізделеді.

- Төтенше бағдарламалау – икемді әдістердің ішінен ең танымал түрі. Ол жүйенің жиі шығарылымына, жүйені жетілдіру үдерісі ұзаққа созылатынына және тапсырыс берушінің жүйені құрастыруына қатысатыны осы жүйенің негізгі ерекшеліктері болып саналады.
- Төтенше бағдарламалаудың бір ерекшелігі – ол автоматтандырылған тестілеу үдерісін бағдарламаны жазғанға дейін өткізеді. Келесі жасақтама қадамына көшу үшін бүкіл тестілеу үдерістері сәтті өтуі қажет.
- Scrum әдісі бағдарламалық жасақтаманың жобалық даму үдерісінің бір әдісі болып табылады. Бұл әдіс бірнеше шапшаң үдерістерден тұрады әрі осы үдерістерден әрқайсысы белгілі бір уақытқа созылады.
- Жоспарлау кезеңі барысында жобаның жұмыс барысы анықталады.
- Үлкен жобаларда икемді әдістердің ауқымдау үдерісі қиын өтеді. Ауқымды жүйелер алдын-ала әрленім мен сипаттамалық құжаттаманы қажет етеді. Бірнеше жасақтаушы топтар бірге жұмыс жасағаннан жүйелік ықпалдасу ұзаққа созылып кетеді.

ҚОСЫМША ӘДЕБИЕТТЕР

Extreme Programming Explained. This was the first book on XP and is still, perhaps, the most readable. It explains the approach from the perspective of one of its inventors and his enthusiasm comes through very clearly in the book. (Kent Beck, Addison-Wesley, 2000.)

'Get Ready for Agile Methods, With Care'. A thoughtful critique of agile methods that discusses their strengths and weaknesses, written by a vastly experienced software engineer (B. Boehm, *IEEE Computer*, January 2002.) http://doi.ieeecomputersociety.org/10.1109/2.976920.

Scaling Software Agility: Best Practices for Large Enterprises. Although focused on issues of scaling agile development, this book also includes a summary of the principal agile methods such as XP, Scrum, and Crystal (D. Leffingwell, Addison-Wesley, 2007.)

Running an Agile Software Development Project. Most books on agile methods focus on a specific method but this book takes a different approach and discusses how to put XP into practice in a project. Good, practical advice (M. Holcombe, John Wiley and Sons, 2008.)

ЖАТТЫҒУЛАР

3.1. Бизнес ортасы үшін жүйенің толық құжаттамасына қарағанда осы жүйені тапсырыс берушіге ерте жеткізу мен орналастыру маңызды. Неге?

3.2. Икемді әдістерді түсіндіретін тұжырымдамалар осы жүйенің дамытылу мен орналастыру үдерістерін қалайша жеделдетінін түсіндіріңіз.

3.3. Қай уақытта бағдарламалық жасақтама құру үдерісінде икемді әдістерді қолданбаған жөн?

3.4. Төтенше бағдарламалау пайдаланушы талаптарын тарихи кестеге енгізіп құрастырады. Осындай үдерістің жақсы мен жаман жақтарын талқылаңыз.

3.5. Неге бағдарлама жасақтаушы тестілеу негізінде жасалатын дамыту әдісі көмегімен жүйе құрылымын толығырақ түсінеді? Осы әдіс барысында қандай қиыншылықтар пайда болуы мүмкін? Жұптық бағдарламалау үдерісі жеке жасақтауға қарағанда екі есе көп сәтті нәтижелерге әкелетініне төрт себеп келтіріңіз.

3.6. *Жиырма үшінші тарауда* көрсетілген жоспарлау арқылы басқарылатын дамыту әдісі мен Scrum әдісінің тұжырымдарын салыстыра отырып сипаттаңыз. Салыстыру екі әдістің жоспарлау үдерісін, қаражатты тағайындау, топ аралық жұмыс пен қарым-қатынас амалдарын қамту қажет.

3.7.Сіз авиациялық басқару жүйесін дайындайтын компанияның бағдарлама жасақтаушысы болып саналасыз. Сіз жүйе талаптарын осы жүйенің толықтай сипаттайтын құжатқа аударатын бағдарламалық жасақтаманың жетілдіруіне жауаптысыз. Осы дамыту үдерісінің жақсы мен жаман жақтарын түсіндіріңіз.

   а. Жоспарлау арқылы басқарылатын дамыту әдісін қолдана отырып, бағдарлама жасақтаушы мен мүдделі тараптар көмегімен жүйе талаптарын құрастырыңыз.

   ә. Жүйе түптұлғасын Ruby немесе Python сияқты сценарийлер тілінің біреуін қолдана отырып құрастырыңыз. Осыдан кейін осы әрленімді бағдарламалық жасақтаушы және мүдделі тараптар көмегімен бағалап жүйелік талаптарды қайтадан қарап шығыңыз. Жүйенің ақырғы нұсқасын Java тілі көмегімен қайтадан жоспарлап шығыңыз.

   б. Java бағдарлама тілін қолданып жүйені құрастырыңыз. Осы бағдарламалық жасақтау үдерісіне пайдаланушыны қатыстыруға тырысыңыз.

3.9. Пайдаланушылардың жасақтаушы топ жұмысына қатысуы пайдаланушының өзі топ талаптарына елігіп қалып өз талаптарын елемейтініне әкеледі екен. Осы мәселені қалай шешер едіңіз және осы әдістің жақсы мен жаман жақтарын талқылап шығыңыз.

ПАЙДАЛАНЫЛҒАН ӘДЕБИЕТТЕР

Ambler, S. W. and Jeffries, R. (2002). *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process*. New York: John Wiley & Sons.

Arisholm, E., Gallis, H., Dyba, T. and Sjoberg, D. I. K. (2007). 'Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise'. *IEEE Trans. on Software Eng.,* **33** (2), 65–86.

Astels, D. (2003). *Test Driven Development: A Practical Guide*. Upper Saddle River, NJ: Prentice Hall.

Beck, K. (1999). 'Embracing Change with Extreme Programming'. *IEEE Computer,* **32** (10), 70–8.

Beck, K. (2000). *extreme Programming explained*. Reading, Mass.: Addison-Wesley.

Carlson, D. (2005). *Eclipse Distilled*. Boston: Addison-Wesley.

Cockburn, A. (2001). *Agile Software Development*. Reading, Mass.: Addison-Wesley.

Cockburn, A. (2004). *Crystal Clear: A Human-Powered Methodology for Small Teams*. Boston: Addison-Wesley.

Cockburn, A. and Williams, L. (2001). 'The costs and benefits of pair programming'. *In Extreme programming examined. (ed.)*. Boston: Addison-Wesley.

Cohn, M. (2009). *Succeeding with Agile: Software Development Using Scrum*. Boston: Addison-Wesley.

Demarco, T. and Boehm, B. (2002). 'The Agile Methods Fray'. *IEEE Computer,* **35** (6), 90–2.

Denning, P. J., Gunderson, C. and Hayes-Roth, R. (2008). 'Evolutionary System Development'. *Comm. ACM,* **51** (12), 29–31.

Drobna, J., Noftz, D. and Raghu, R. (2004). 'Piloting XP on Four Mission-Critical Projects'. *IEEE Software,* **21** (6), 70–5.

Highsmith, J. A. (2000). *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. New York: Dorset House.

Hopkins, R. and Jenkins, K. (2008). *Eating the IT Elephant: Moving from Greenfield Development to Brownfield. Boston,* Mass.: IBM Press.

Larman, C. (2002). *Applying UML and Patterns: An Introduction to Object-oriented Analysis and Design and the Unified Process*. Englewood Cliff, NJ: Prentice Hall.

Leffingwell, D. (2007). *Scaling Software Agility: Best Practices for Large Enterprises*. Boston: Addison-Wesley.

Lindvall, M., Muthig, D., Dagnino, A., Wallin, C., Stupperich, M., Kiefer, D., May, J. and Kahkonen, T. (2004). 'Agile Software Development in Large Organizations'. *IEEE Computer,* **37** (12), 26–34.

Martin, J. (1981). *Application Development Without Programmers*. Englewood Cliffs, NJ: Prentice-Hall.

Massol, V. and Husted, T. (2003). *JUnit in Action*. Greenwich, Conn.: Manning Publications Co.

Mills, H. D., O'Neill, D., Linger, R. C., Dyer, M. and Quinnan, R. E. (1980). 'The Management of Software Engineering'. *IBM Systems*. J., **19** (4), 414–77.

Moore, E. and Spens, J. (2008). 'Scaling Agile: Finding your Agile Tribe'. *Proc. Agile 2008 Conference,* Toronto: IEEE Computer Society. 121–124.

Palmer, S. R. and Felsing, J. M. (2002). *A Practical Guide to Feature-Driven Development*. Englewood Cliffs, NJ: Prentice Hall.

Parrish, A., Smith, R., Hale, D. and Hale, J. (2004). 'A Field Study of Developer Pairs: Productivity Impacts and Implications'. *IEEE Software,* **21** (5), 76–9.

Poole, C. and Huisman, J. W. (2001). 'Using Extreme Programming in a Maintenance Environment'. *IEEE Software,* **18** (6), 42–50.

Rising, L. and Janoff, N. S. (2000). 'The Scrum Software Development Process for Small Teams'. *IEEE Software,* **17** (4), 26–32.

Schwaber, K. (2004). *Agile Project Management with Scrum*. Seattle: Microsoft Press.

Schwaber, K. and Beedle, M. (2001). *Agile Software Development with Scrum*. Englewood Cliffs, NJ: Prentice Hall.

Smits, H. and Pshigoda, G. (2007). 'Implementing Scrum in a Distributed Software Development Organization'. Agile 2007, Washington, DC: IEEE Computer Society.

Stapleton, J. (1997). *DSDM Dynamic Systems Development Method*. Harlow, UK: Addison-Wesley.

Stapleton, J. (2003). *DSDM: Business Focused Development, 2nd ed*. Harlow, UK: Pearson Education.

Stephens, M. and Rosenberg, D. (2003). *Extreme Programming Refactored*. Berkley, Calif.: Apress.

Sutherland, J., Viktorov, A., Blount, J. and Puntikov, N. (2007). 'Distributed Scrum: Agile Project Management with Outsourced Development Teams'. 40th Hawaii Int. Conf. on System Sciences, Hawaii: IEEE Computer Society.

Weinberg, G. (1971). *The Psychology of Computer Programming*. New York: Van Nostrand.

Williams, L., Kessler, R. R., Cunningham, W. and Jeffries, R. (2000). 'Strengthening the Case for Pair Programming'. *IEEE Software,* **17** (4), 19–25.

4

Өндіріс талаптары

Мақсаттары

Төртінші тараудың мақсаты – бағдарламалық жасақтаманың талаптарын таныстыру және талаптардың ашылымы мен құжаттамасы барысындағы үдерістерді талқылау. Осы тараудың соңына жеткенде сіз:

- пайдаланушы мен жүйе талаптарын және осы талаптардың неге түрлі амалмен жазылу себебінің тұжырымдамаларын;
- функциялық және функциялық емес бағдарламалық жасақтама талаптарының арасындағы айырмашылықтын;
- бағдарламалық жасақтама талаптарының құжаттауында талаптардың қалай ұйымдастарылатынын;
- өндіріс талаптарының басты анықтау, талдау және қабылдау қызметтерін, осы қызметтердің өзара әрекеттерін;
- талаптарды басқару неге маңызды екенін және басқа да өндіріс талаптарының қызметін қалай қолдайтынын білетін боласыздар.

Мазмұны

Көрсетілген қызметтер мен оның шектеу амалдары – жүйеге арналған талаптар, жүйенің не жасау керектігінің анықтамасы. Бұл талаптар пайдаланушылардың жабдықты бақылау, тапсырыс беру немесе ақпарат іздеу сияқты белгілі мақсаты бар жүйеге тұтынушылығын көрсетеді. Анықтау, талдау, құжаттау және қызметтер мен шектеулерді тексеру үдерісі өндіріс талаптары (ӨТ) деп аталады.

«Талап» термині бағдарламалық жасақтама өнеркәсібінде дәйекті түрде қолданылмайды. Қызметті абстрактілі баяндау, жүйені қамтамасыз ету немесе жүйені шектеу сияқты талаптар кей жағдайларда тіпті, жоғарғы деңгейде. Бір себептен, бұл егжей-тегжейлі жүйе функциясының формалды анықтамасы. Дэвис (1993) бұл айырмашылықтардың болуын былай түсіндіреді:

> *Егер де компания бағдарламалық жасақтама дамуы бойынша үлкен жобаға келісімшарт жасағысы келсе, шешімі дайын болмағандықтан өз тұтынуларын барынша абстрактілі қарастыруы қажет. Талаптар бірнеше контракторлар жоба үшін саудаласып ұсыныс жасайтындай, мүмкіндігінше ұйым клиентінің сұрауларын қанағаттандыратын түрлі амалдар қарастыратындай жазылуы керек. Жобаға марапат беруге ұйғарса, контрактор клиентке жүйе анықтамаларын толықтай, бағдарламалық жасақтаманың мүмкіндігін түсініп қабылдайтындай жазуы тиіс. Осы құжаттың екеуін де жүйеге талапты құжаттау деп атауға болады.*

Өндіріс талаптары үдерісі барысында пайда болған кейбір күрделі мәселелер осы анықтамалардың түрлі деңгейлер араларын айқын бөлудің сәтсіз қорытындысы болып табылады. Мен бұларды мына терминдер арқылы ажыратамын: жоғары деңгейдегі абстрактілі талаптарды «пайдаланушылар талаптары», жүйенің не істеу керектігінің толық сипаттамасын «жүйе талаптары» қарастырады. Пайдаланушылар мен жүйе талаптары келесі тәсілдермен анықталады:

1. Пайдаланушылар талаптары мәлімдемесі жүйе қызметін табиғи тілде және диаграммаларда, күткендей, жүйе пайдаланушылары жұмыс жасайтын шекараларды қамтамасыз етеді.
2. Жүйе талаптары басқармалық жасақтама функциясы, қызметі мен эксплуатациялық шегерулерінің толықтай сипаттамасы. Жүйе талаптарын құжаттау (кейде функциялық сипаттау деп аталады) нақты ненің құрастырылуы жөн екенін анықтайды. Бұл жүйе сатып алушы мен бағдарламалық жасақтама құрушылар арасындағы келісім болуы мүмкін.

Неше түрлі деңгейдегі талаптар тиімді болып саналады, себебі олар жүйе жайындағы ақпаратты әртүрлі тұрпатты оқырмандармен байланыстырады. *4.1-сурет* пайдаланушы мен жүйе талаптары арасындағы айырмашылықты көрсетеді. Бұл мысал емделушінің психикалық денсаулығының жүйелік басқаруы (ЕПДЖБ), пайдаланушы талаптары бірнеше жүйе талаптарына бөлінетінін көрсетеді.

4.1-сурет сіз пайдаланушы талабы барынша жалпы екенін көре аласыз. Жүйе талаптары – құрастырылуға тиісті жүйе қызметтері мен функцияларын нақты

мағлұматпен қамтамасыз ету. Сіз талаптарды бірнеше деңгейде талдап жазуыңыз тиіс, себебі әр оқырман оны әр бағытта қолданады. 4.2-сурет пайданушылар мен жүйе талаптарының мүмкін болатын оқырмандарын көрсетеді. Әдетте, пайдаланушы талаптарының оқырмандары жүйенің қалай құрастырылатынын және жүйе объектінде толық қызықпайтын менеджерлер болатынын ойламайды. Жүйе талаптарының оқырмандары жүйенің не жасайтындығын нақты білгендері жөн, себебі олар бизнес үдерістерін қалай қолдайтындығына араласады немесе жүйенің жүзеге асуына атсалысады.

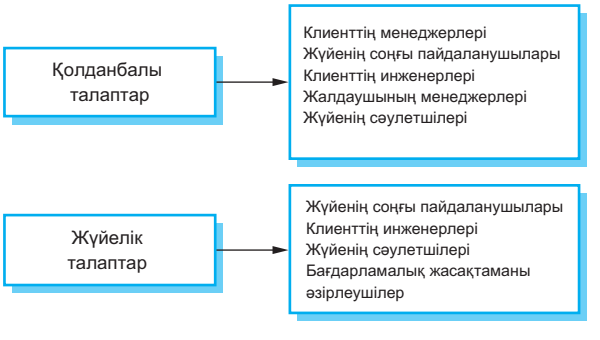
Қолданбалы талаптың ұйғарымы

1. MHC-PMS сол ай аралығында , әрбір клиникаға бөлінген дәрі-дәрмектің бағасына , ай сайын басқарушылық есептеме жасау керек.

Жүйелі талаптың айрықшалануы.

1.1 Әр айдың соңғы жұмыс күнінде жазылып берілген дәрі-дәрмектің қысқа мазмұны, олардың бағасы және белгілеген клиникалардың тізімі шығуы керек.
1.2 Жүйе автоматты түрде есептемені , айдың соңғы жұмыс күнінде 17.30 дан кейін жүргізеді.
1.3 Есептеме әр клиникаға жасалуы міндетті және препараттардың жеке атауларын, белгіленген дәрі- дәрмектердің жалпы құнын атап өту міндетті.
1.4 Егер препараттар түрлі мөлшердің өлшеміне ашық болса (мысалы 10 мг, 20 мг), онда жеке баяндамалар әрбір мөлшердің бірлігіне жеке дайындалуы керек.
1.5 Барлық есептемелердің құны рұқсат алудың тексеріс тізіміне аударылған, тіркелген пайдаланушыға жабық болуы керек.

4.1-сурет. Пайдаланушы және жүйенің талаптары



4.2-сурет. Әртүрлі талаптар сипаттамасының оқырмандары

Бұл тарауда мен талаптардың икемді үдерістерінен бұрын «дәстүрлі» түрін қарастырамын. Көбіне көлемді жүйелерге бұрынғыдай айқын талқыланған өндіріс талаптарының кезеңі жүйе басталымының жүзеге асырылуынан бұрын тұрады. Жүйе дамуы келісімшартының бөлігі бола алатын талаптардың құжаттамасы

қорытынды нәтиже болады. Әрине, кезектегі талаптардағы өзгерістер және пайдаланушылардың талаптары бұдан да нақты айқындалған жүйе талаптарына кеңею мүмкін. Бірақ талаптарды икемді ықпалда бір уақытта анықтау ірі жүйе дамуына қолданылатын құрастырылған жүйе секілді.

4.1. Функциялық және функциялық емес талаптар

Бағдарламалық жасақтама жүйесінің талаптары көбіне, функциялық немесе функциялық емес деп сұрыпталады:

1. *Функциялық талаптар.* Бұл жүйе қызметі жайындағы пікір жүйенің жеке қолдануына қалай әрекет ететінін және жүйенің белгілі жағдайларда қалай амалданатынын қамтамасыз етеді. Кей жағдайларда функциялық талаптар жүйенің не жасамауға тиістілі екенін нақты анықтай алады.
2. *Функциялық емес талаптар*. Бұл жүйе қарамағынан көрсетілген қызметтер мен функциялардағы шегерулер. Оған уақытқа, даму үдерісіне және стандарт міндеттеріне қойылған шегерулер жатады. Функциялық емес талаптар көбінесе, жүйенің жеке функциялары мен қызметтеріне қарағанда жалпылай жүйенің өзіне қолданылады.

Шынында, талаптардың бірнеше түрлеріндегі өзгешеліктер осы жай анықтамалардағыдай, сондай айқын емес. Тіркелген пайдаланушыларға рұқсат жоқ деген мәлімдеме секілді, қамтамасыздыққа байланысты пайдаланушы талаптары функциялық емес талап болып қалуы мүмкін. Әйткенмен, тым егжей-тегжейіне дейін дамығанда, осы талаптар бұдан басқа, жүйеде пайдаланушы тіркелімінің мүмкіндіктерін қажет ететін сияқты анық талаптар шығарады.

Бұл талаптардың тәуелді еместігін және бір талап әдетте, жаңасын қарастыра алатындығын немесе оның басқасын шегеретіндігін көрсетеді. Демек, жүйе талаптары керекті жүйе қызметтері мен функцияларын ғана қамтамасыз етіп қоймай, осы қызметтер/функциялар дұрыс жеткізілуінің қажетті функцияларын қамтиды.

4.1.1. Функциялық талаптар

Жүйеге бағытталған функциялық талаптар жүйе не жасау керек екендігін анықтайды. Бұл талаптар құралып жатқан бағдарламалық жасақтама түрлеріне, пайдаланушылардың бағдарламалық жасақтама болжамына және талаптардың жазылу барысындағы ұйым қабылдаған жалпы ықпалдарға тәуелді. Пайдаланушылар талаптары функциялық талаптар айтылымында әдетте, абстрактілі түрде жүйе пайдаланушыларына түсінікті болатындай анықталады. Солай болса да, көбіне функциялық жүйе талаптары жүйенің функциясын, оның кірісі мен шығысын, ерекшеліктерін, т.б. толық суреттейді.

Функциялық жүйе талаптары жұмыстың тұрақты әдістерімен немесе ұйымның бар жүйелерінде болатын ерекше талаптар үшін жүйенің не жасау керектігін қамтамасыз ететін жалпы талаптардан бастап түрленеді.

Мысалға, мына жерде психикалық зақымдалған денсаулығына ем қабылдаушы аурулар жайлы ақпараттарды қолдайтын *емделушінің психикалық денсаулығының жүйелік басқаруы* (ЕПДЖБ) жүйесінің функциялық талаптарына үлгілер берілген:

1. Пайдаланушы кездесу тізімдерін барлық емханада қарауына мүмкіндігі болуы керек.
2. Жүйе белгілі күні кездесуге барам деген емделуші тізімдерін әр күні, әр емханада шығаруы керек.
3. Әрбір қызметкерде жүйені қолдану үшін өзінің ерекше жеке сегіз сандық қыметкер нөмірі болуы керек.
4. Бұл пайдаланушының функциялық талаптары жүйеде қарастырылатын нақты мүмкіншіліктерін анықтайды. Бұлар пайдаланушы талаптары құжаттамасынан алынған және функциялық талаптардың әртүрлі тәптіштеу деңгейінде жазуға болатындығын көрсетеді (1 мен 3 қарама-қарсы талаптар).
5. Талаптардың сипаттамасындағы олқылық бағдарламалық жасақтама өндірісінде көптеген мәселенің тууына себеп болады. Құрастырушының әлдеқандай талапты орындалуы оңай болатын амалмен түсіндіруі табиғи жағдай. Бірақ, көбіне, бұл клиенттің қалауы емес. Жаңа талаптар орнатылып, жүйеде өзгертілуі тиіс. Әрине, осы тапсырыс кідірістері шығынды көбейтеді.
6. Мысалы, емделушінің психикалық денсаулығының жүйелік басқаруына (ЕПДЖБ) қойылған бірінші үлгі талабы – пайдаланушыға тағайымдалған тізімдерді барлық емханада қарауына мүмкіндік беру. Бұл талапқа негіздеме кейде емделушінің психикалық денсаулығының проблемасын шатастыру. Олар бір емхананың орнына басқасына баруы мүмкін. Егер олар тағайындалған тізімде болса, онда емханада бұрын тіркелген науқастай емделеді.
7. Осыған сүйене отырып, емхана қызметкерлері емделушінің аты-жөнін «іздеу»-ге енгізгенде жүйе емделуші аты-жөні арқылы оның барлық емханадағы тағайындалған тізімдерін іздейді. Соған қарамастан бұл талаптар да нақты болып есептелмейді. Жүйе құрастырушылары талаптарды әрқалай түсіндіре алады және іздеуге кіріспеген бұрын пайдаланушы алдымен емхананы таңдайтындай етіп іске асыра алады. Сірә, бұл пайдаланушылардың кірісін көп шақырады және уақыты жағынан ұзағырақ болады.
8. Негізінде, жүйенің функциялық талап сипаттамасы толық және ретті болады. Толықтық пайдаланушы қажет еткен барлық қызметтің айқындалуын, яғни реттілік талаптардың қарама-қайшы анықтаулары болмайтынын білдіреді. Іс-тәжірибеде, ірі және күрделі жүйелерде толық және ретті талаптарды жүзеге асыру мүмкін емес дерлік. Бұның бір себебі – күрделі жүйелерге

сипаттама жазу барысында қателіктер мен абайламаушылықтардың көп болуы. Келесі себеп, ірі жүйелерде қызығушылықпен қарайтын адамдардың көп болуы. Мүдделі тұлғалар немесе олардың жүйедегі маңызы қайсыбір дәрежеде жүйеге тәуелді. Әртүрлі және көбіне дәйексіз тұтынушылықтар мүдделі тұлғалардікі. Талаптар алдымен орындалып болғанда дәйексіз талаптар сипаттамаға қосылса, осы дәйексіздіктердің ақиқат болуы мүмкін емес. Бұл мәселелер терең қарастырылған талдаудан соң немесе жүйе тапсырыс берушіге берілгеннен кейін шығуы мүмкін.

4.1.2. Функциялық емес талаптар

Функциялық емес талаптар, аты айтылғандай, жүйенің пайдаланушыларына көрсетілетін нақты қызметтермен тура араласпайтын талаптар болып табылады. Олар жүйенің сенімділік, жауап қайтару уақыты, орынды сақтау секілді жаңа қасиеттеріне жатады. Кезекте, жүйе құрастырылымына кіру/шығу құралдарының мүмкіншілігі немесе басқа да жүйе интерфейсінде қолданылатын ақпарат көрсетілімдері дегендей шегерулер тағайындайды.

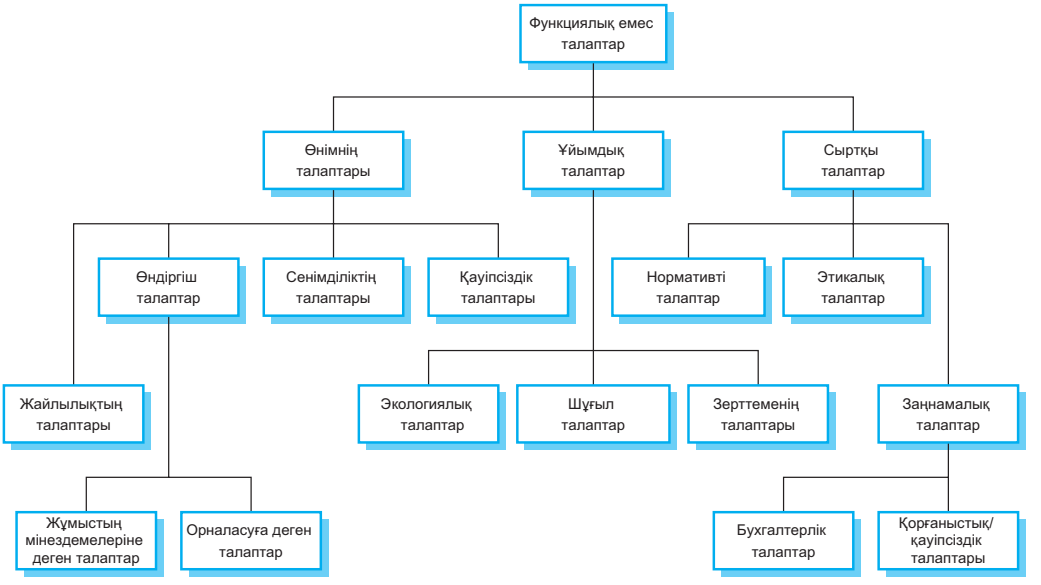
Өндірімділік, қауіпсіздік немесе қолжетімділік секілді функциялық емес талаптар, әдетте, жалпы жүйенің сипаттамасына шегерулерді көрсетеді. Жеке функциялық талаптарға қарағанда функциялық емес талаптар көбіне қиын болады. Әшейінде, жүйе пайдаланушылары тұтынуларына жауап бермейтін жүйелерден айналып өтіп, жұмыс жасаудың амалдарын табады. Соған қарамастан, сәтсіз болған функциялық емес талаптар бүкіл жүйенің қолдануға жарамсыз екенін білдіреді. Мысалы, егер ұшақ жүйесі сенімділік талабын қамтымаса, бұл сенімді операция болып сертификатталмайды; егер кіріктірме жүйе басқаруы өндірімділік талаптарына сай болмаса, басқару функциялары дұрыс жұмыс істемейді.

Дегенмен, қандай жүйе құрамдары нақты функциялық талаптарды құрастыратынын (мысалы, есеп беру талаптарын орындайтын форматты құрамдар болуы мүмкін) жиі анықтауға болады, бұл құрамдарды функциялық емес талаптарға байланыстыру өте қиын. Бұл талаптардың іске асуы бүкіл жүйе бойынша таралуы мүмкін. Бұған екі себеп бар:

1. Функциялық емес талаптар жүйенің жеке құрамдарынан бұрын жалпы архитектурасына әрекет етуі мүмкін. Мысалы, талаптардың қолданылуына сенімді болу үшін, сізге құрамдар арасындағы қатынасты ықшамдайтын жүйе қарастыру қажет.
2. Қауіпсіздік талабы секілді жалғыз функциялық емес талап керекті жаңа жүйе қызметтерін ойластыратын қатар функциялық талаптар байланысын құрайды. Одан басқа, бар талаптарды шегеретін талаптарды жүзеге асырады.

Функциялық емес талаптар пайдаланушылардың тұтынулары барысында пайда болады, бұл қаржы шегерімі, ұйым саясаты, басқа да бағдарламалық

және жабдықтамалық жасақтама жүйелерімен қатынасу мұқтаждығы немесе қауіпсіздікті реттейтін не құпиялық заңнама секілді ішкі факторлар себебінен болады. 4.3-сурет функциялық емес талаптардың сарапануы.



4.3-сурет. Функционалды емес талаптардың үлгілері

Осы диаграммадан сіздер функциялық емес талаптардың сұраныстағы бағдарламалық жасақтаманың (өнім талаптары) сипаттамасынан, ұйым құрастырған бағдарламалық жасақтамадан (ұйым талаптары) немесе ішкі қордан болатынын көре аласыздар:

1. *Өнім талаптары.* Бұл талаптар бағдарламалық жасақтаманың тәртібін анықтайды немесе шегереді. Оған жүйенің қалай тез орындалуы және қанша сақтау желісін сұрайтыны, интенсивті қарсылық, қауіпсіздік және ыңғайлылық талаптарын түсіндіретін сенімділік талаптары кіреді.
2. *Өнім талаптары.* Бұл талаптар тапсырыс беруші мен құрастырушылар ұйымының саясаты мен рәсімінен шыққан жалпы жүйе талаптары. Мысалы, жүйенің қалай қолданатынын анықтайтын, бағдарламалық тілді айқындайтын даму үдеріс талаптары, даму ортасын немесе үдеріс үлгілерін қолданатын және жүйенің ортада әрекет етуін анықтайтын қоршаған ортаға бағытталған талаптар іс жүзіндегі үдеріс талаптарына жатады.
3. *Сыртқы талаптар.* Бұл тақырып жүйеге сыртқы ықпалдары бар және олардың даму үдерісінен шыққан барлық талаптарды қамтиды. Бұл жүйеге орталық банк секілді реттегіш қолдануға рұқсат алу үшін не жасалу керектігін; жүйе заңмен атқарылатынын айқындайтын заңды талаптар;

жүйенің пайдаланушылары мен жалпы халыққа ашық болуын анықтайтын этикалық талаптар секілді қалыпты талаптар жатады.

ӨНІМ СҰРАНЫСЫ
MHC-PMS стандартты Дүйсенбіден Жұмаға дейін сағат 08.30-дан 17.30-ға дейін жұмыс істейтін барлық клиникаларда болу керек.

ҰЙЫМ СҰРАНЫСЫ
MHC-PMS-тің пайдаланушылары медициналық карталарын пайдаланып, жүйеде анықталуы керек.

СЫРТТАЙ СҰРАНЫС
Жүйе пациенттердің жеке ақпараттарын HStan-03-20006-priv-те айтылғандай қауіпсіз әрі сенімді сақтау керек.

4.4-сурет. MHC-PMS-тегі функционалдық емес сұраныстардың мысалдары

4.4-сурет өнім үлгілерін, пайдаланушы талаптары *4.1.1-бөлімде* айтылған ЕПДЖБ-нан алынған ұйымдық және сыртқы талаптарды көрсетеді. Өнім талаптары – жүйенің қай кезде ашық және күннің қай уақытында пайдалануға балатынын анықтайтын талаптардың бар болуы. Мұның ЕПДЖБ функциясына және нақты анықталған жүйе дизайнері қарастыратын шегерулерге қатысы жоқ.

Ұйым талаптары пайдаланушының өз еркімен қалай жүйеге тіркелетінін дәйектейді. Денсаулық сақтау мекемелері бүкіл бағдарламалық жасақтамаға арналған стандарттық аутентификация рәсімімен жұмыс барысында пайдаланушының тіркелген атының орнына оны тану үшін жеке куәлік карточкасын оқитын құрал арқылы өткізеді. Жүйенің құпиялық заңдармен келісуі сыртқы талаптарға жатады. Әлбетте, құпиялық денсаулық сақтау жүйелерінде өте маңызды және талаптарда жүйе ұлттық құпиялық стандартқа сай болып құрастырылуы керек екендігі көрсетілген.

Функциялық емес талаптардың жалпы проблемасы, әдетте, пайдаланушылар мен тапсырыс берушілер үшін бұл талаптарды қолданудың оңайлылығы, жүйенің сәтсіздіктен түзелу қабілеттілігі немесе пайдаланушының жауап қайтаруы сынды жалпы мақсат ретінде ұсынады. Мақсаттар жақсы ниетпен қойылған, бірақ жүйе құрастырушыларына проблема туғызады, себебі олар жүйе жеткізілген соң кезектегі дауға жол ашады. Мысалы, келесі жүйе мақсаты әдеттегідей менеджердің қолайлылық талаптарын қалай қоятынын көрсетуі:

Жүйе емхана қызметкерлерінің қолдануына оңай болуы керек және пайдаланушылардың қателік жіберулерін барынша азайтатындай құрастырылуы керек.

Мен бұл мақсаттың қалай «тексерілетін» функциялық емес талап болатынын білдіру үшін мұны қайтара жаздым. Жүйе мақсатын айқын анықтау мүмкін емес, бірақ келесі сипаттамаларда, не болғанда, сіз жүйені тестілеу кезінде жасалған қателіктерді санайтын бағдарламалық жасақтама құрылғысын қоса аласыз.

> *Емханалық қызметкерлер төрт сағат оқыту сабағынан кейін жүйе функцияларына рұқсаты болуы қажет. Осы дайындықтан кейін, тәжірибиелі пайдаланушылардың орташа жіберген қателігі сағатына екі қатеден артық болмауы тиіс.*

Функциялық емес талаптарды әр мүмкін сәті түскен сайын оны дұрыс тексерілетіндей қылып, жасау керек. *4.5-суретте* жүйенің функциялық емес қасиетін анықтайтын метрикалар көрсетілген. Сіз осы сипаттамаларды сынақтан өткізерде функциялық емес талаптардың болу не болмауын байқай аласыз.

Қасиет	Өлшембірлік
Жылдамдық	Орындалған тарнзакциялар / секунд Пайдаланушы / жауап уақыты Монитордың жаңғырту уақыты
Көлем	Мегабайттар ROM чиптарының саны
Пайдаланудың жеңілділігі	Жаттығу уақыты Көмекші кескіндердің саны
Сенімділік	Сәтсіздіктердің орташа саны Жетімсіздіктің ықтималдығы Жетімділік
Төзімділік	Сәтсіздіктен кейінгі қайта жүктелу уақыты Сәтсіздікке әкелетін жағдайлардың проценті Сәтсіздіктің нәтижесіндегі ақпараттың зақымдану ықтималдылығы
Ықшамдылық	Құрылғыға тәуелді тұжырымдардың проценті Құрылғылардың саны

4.5-сурет. Функционалдық емес сипаттардың метрикалары

Іс-тәжірибеде клиенттер жүйеде өз мақсаттарын өлшенерлік талаптарға ауыстыруды қиын деп тапты. Жөндеуге жарамдылық сияқты кейбір мақсаттарға қолдануға болатын метрикалар жоқ. Басқа жағдайда, кейбір сипаттамаларына рұқсат болса да, клиенттер өз тұтынуларын осы сипаттамалармен байланыстыра алмайды. Олар кейбір анықталған сандардың сенімділікті сұрайтыны олардың күнделікті тәжірибеде компьютер жүйесімен байланысын білдіретінін түсінбейді. Бұдан басқа, олар тексерілу бағасының шын өлшенбелі функциялық емес талаптардың қымбат және клиенттер жүйеге төлегені ақталады деп ойламайды.

Көп жағдайда функциялық емес талаптар басқа функциялық және функциялық емес талаптарға қарама-қайшы әрекет етеді. Мысалы, *4.4-суретте* түпнұсқалық тексеріс талаптары, сірә, жүйеге қосылған әрбір компьютерде карточканың оқылуын сұрайды. Бірақ дәрігерлер мен медбикелердің компьютерлерінен жүйеге мобильдік рұқсат алуын сұрайтын тағы бір талап болуы мүмкін. Бұлар карточка оқығыштармен толық қамтамасыздандырылмаған, сол себепті түпнұсқалық тексеріс әдістерінің баламалы жолдарын қолдануға рұқсат береді.

Талаптар құжатының стандарттары

АҚШ қорғаныс департаменті және Электрлік және электрондық инженерлер институты (IEEE) сияқты көптеген ірі мекемелер талап құжатына қатысты стандарттарды анықтап алды. Әдетте олар біртекті болып келеді, бірақ әйткенмен анағұрлым толыққанды ұжымдық стандарттардың даму негізі ретінде пайдалы болып табылады. АҚШ Электрлік және электрондық инженерлер институты (IEEE) мекемесі стандарттармен қамдаушылардың арасындағы ең танымал мекемелердің бірі және олар талап құжаттарының құрылымына стандарт жасақтап шығарды. Бұл стандарт әскери команда мен басқару жүйелері сияқты жүйелерге сай келеді. Онда жүйелердің қолданылу мерзімі ұзақ келеді және әдетте олар топтық мекемелермен жобаланады..

http://www.SoftwareEngineering-9.com/Web/Requirements/IEEE-standard.html

Іс-тәжірибеде, талаптарды құжаттауда функциялық және функциялық емес талаптарды ажырату өте қиын. Егер де функциялық емес талаптар функциялық талаптардан ажыратылса, онда бұлардың арасындағы байланысты түсіну одан да қиындайды. Соған қарамастан, сіздер орындалу немесе сенімділік секілді жүйе қасиеттеріне нақты байланысқан талаптарды айқын бөлулеріңіз жөн. Сіздер мұны талаптарды құжаттаудың бөлек бөліміне салу арқылы немесе басқа жүйе талаптарынан ажырату арқылы жасауларыңызға болады.

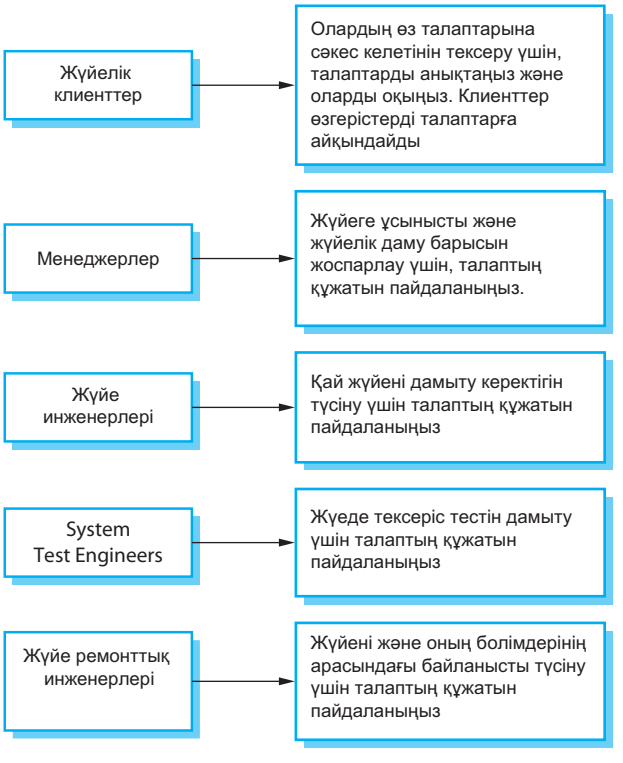
Сенімділік, қауіпсіздік және құпиялық талаптар сияқты функциялық емес талаптар көптеген қажетті жүйелерге ерекше маңызды. Бұл талаптарды *12-тарауда* толығырақ сенімділік және қауіпсіздік талаптарын анықтайтын нақты әдістері ретінде қарастырамын.

4.2. Бағдарламалық жасақтама талаптарын құжаттау

Бағдарламалық жасақтама талаптарын құжаттау (кейде бағдарламалық жасақтама сипаттамасы немесе БЖС деп аталады) – құрастырушылардың жүйеде нені іске асыруы туралы ресми мәлімдеме. Ол жүйеге және толық сипаттамаға арналған пайдаланушылардың талаптарын қамтуы керек. Кейде пайдаланушылар мен жүйе талаптары жеке сипаттамаға ықпалданған. Басқа жағдайларда пай-

даланушы талаптары бағдарламалық жасақтама сипаттамасының кіріспесінде анықталады. Егер талап сандары көп болса, онда жүйенің талаптары толығырақ бөлек құжаттамада таныстырылады.

Талаптарды құжаттау бағдарламалық жасақтама жүйесін бөтен мекеме құрастыратын кезде қажет. Бірақ үдеріс кезіндегі даму әдістерің өзгеруінен, құжаттау ескіріп қалып, жазылған талаптар шұғыл өзгереді, сол себепті бар салынған күш текке кетеді. Бұған қарағанда, біртіндеп пайдаланушы талаптарын жинайтын және оларды картада пайдаланушылар тарихы ретінде жазылған Экстремалдық Бағдарламалау (Extreme Programming, Beck, 1999) сияқты ресми құжаттама амалдары жақсырақ, содан пайдаланушы жүйенің келесі бөлімінің дамуына талап басымдылықтарын береді.



4.6-сурет. Талаптар құжаттамасының пайдаланушылары

Талаптары тұрақты емес кәсіптік жүйелерге осындай амалдар ыңғайлы. Бірақ бизнес пен жүйе талаптарын анықтайтын қысқа жұмыс барысын сипаттайтын құжаттама пайдалы деп ойлаймын; бұл талаптар жүйенің жалпы функциялық талаптарының орындалуына көзделгенде оңай ұмытылып кетеді.

Талаптарды құжаттау бағдарламалық жасақтаманың дамуына жауапты жүйе инженерлеріне ақы төлейтінін ұйымның бас басқарушысынан бастап неше түрлі пайдаланушыларды қамтиды. *4.6-суретте* Джеральд Котонья екеуіміздің

кітабымыздан (Kotonya and Sommerville, 1998) алынған құжаттауда мүмкін болатын пайдаланушылар мен оларды қалай пайдалануын суреттеп еді.

Мүмкін болатын пайдаланушылардың әртүрлілігі талаптарды құжаттауда клиентермен әрекеттесетін талаптар арасында құраушылар мен тексерушілер талаптары, оның ішінде жүйенің мүмкін эволюциясы жайлы ақпарат, толықтай түсіндірілген келісімшарт болуы керектігін білдіреді. Жорамалданған өзгерістер ақпарат жүйесінің әрленімін даярлаушысына шектелген шешімдерінен сақ болуына және жүйені жаңа талаптарға бейімдендіруіне көмектеседі.

Тарау	Баяндама
Алғысөз	Осында құжаттың оқырмандары және оның тарихи шығарылымдары анықталады. Жаңа шығарылымның алдыңғы шығарылымнан айырмашылықтары анықталады.
Кіріспе	Бұл жүйенің керектігін суреттейді. Жүйенің функиялары және оның басқа жүйелермен қалай жұмыс істейтіні суреттеледі. Және де жүйенің даярлаушы ұжымның стратегиялық тұжырымдарымен келісуі анықталады.
Пайдаланушы сұраныстарының анықталуы	Бұл жерде пайдаланушыға арналған қызметтер суреттеледі. Функционалды емес сұраныстар да осы бөлімде суреттелу керек. Бұл анықтамалар қарапайым тілде жазылып, тұтынушыға түсінікті диаграммалар, кестелер көрсетіледі.
Жүйе сәулеті	Бұл тарауда функиялардың жүйедегі модульдермен байланысын көрсететін жоғарғы деңгейдегі жүйенің сәулеті анықталады. Қайта қолданған сәулеттік компоненттер ерекше көрсетіледі.
Жүйе сұраныстарының сипаттамасы	Осында функционалды және функционалды емес сұраныстар толықтай талқыланады. Керек болса, функционалды емес сұраныстарға да қосымша талқылау қосылады. Пайдаланушы жүйелерінің интерфейсін де талқылауға болады.
Жүйе модельдері	Осында жүйе компоненттерінің, жүйе мен қоршаған ортаның қарым қатынасы графикалық түрде анықталады.
Жүйе эволюциясы	Бұл жүйенің негізгі ой-пікірлерін және әзірлену эволюциясындағы өзгесістерін суреттейді.
Аппендикс	Бұл тарауда әзірленіп жатқан жүйенің нақты сипаттамасы беріледі. Мысал ретінде, қамтамасыз техника немесе дерекқорлар жинағын келтіруге болады. Қамтамасыз техника сипаттамасы жүйенің минималды конфигурациясын суреттейді. Дерекқорлар сипаттамасы деректердің жүйесін және өзара байланысын суреттейді.

Индекс	Кейбір құжаттардың индекстері қосылады. Әдеттегі әліпби индекстермен қатар, диаграммалардың және функциялардың индекстері қосылады.

4.7-сурет. Сұраныстар құжатының құрылымы

Талаптарды құжаттаудағы ақпараттардың деңгейі құрастырылатын және құрастыру үдерісінде қолданылатын жүйе түрлеріне тәуелді.

Өте қажетті жүйелер нақты айқындалған талаптарды керек етеді, себебі сенімділік пен қауіпсіздік толық талдануы тиіс. Жүйе әртүрлі компанияларда құрастырылса (мыс., сыртқы ресурстар арқылы), жүйе сипаттамасы толық және нақты болуы керек. Егер үйде қайталанбалы құру үдерісі құрылса, талаптарды құжаттау аз ақпараттандырылады және әрбір екі жақты ойлар жүйенің даму барысында шешіледі.

4.7-суретте белгілі бір ұйымның талаптар құжаты IEEE стандартында (IEEE, 1998) бекітілген талаптар құжаттауын көрсетеді. Бұл стандарт – маңызды қолдануларға арналған жалпы стандарт. Бұл жағдайда, мен болжамалы жүйе бағалары жайлы ақпарат қосу арқылы стандартты кеңейтемін. Бұл ақпарат жүйе атқосшысына көмектеседі және дизайнерлерге жүйенің келешек функцияларына қолдау көрсетуіне рұқсат етеді.

Әрине, талаптар құжаттауына қосылған ақпарат құрастырылған бағдарламалық жасақтаманың түрлеріне және дамуға қолданылатын ықпалға тәуелді. Егер бағдарламалық жасақтама өніміне (айтарлық) эволюциялық ықпал қабылданса, жоғарыда ұсынылған тараудан толықтай талаптарды құжаттау алынып тасталады. Басты көңіл пайдаланушылардың талаптарын анықтауға және жоғары деңгейге, жүйенің функциялық емес талаптарына аударылады. Бұл жағдайда, дизайнерлер мен бағдарлама жасаушылар пайдаланушылардың жүйеге деген талаптарын қалай шешетін жоспары секілді өз ой-пікірлерін қолданады.

Бірақ, бағдарламалық жасақтама жабдықтамалық жасақтама мен бағдарламалық жасақтама жүйелерін қамтитын ірі жобаның бір бөлігі болып келген кезде, әдетте, тұтынушының талаптарын анықтау өте маңызды. Бұл талаптардың құжаттауы ұзақ болып келуін білдіреді және де *4.7-суретте* көрсетілгендей, тараудың негізгі мазмұнын қосады. Ұзақ құжаттауларға оқырмандар өзі іздеген ақпараттарын табу үшін мазмұнның комплексті кестелері мен құжаттау индекстерін қосу өте маңызды.

4.3. Талаптардың сипаттамасы

Талаптардың сипаттамасы – талаптардың құжаттауында пайдаланушылар мен жүйе талаптарын жазу үдерісі. Пайдаланушылар мен жүйе талаптары айқын, анық, оңай түсінетін, толық және жүйелі болып келсе, тіпті жақсы. Іс-тәжірибеде

бұған тән шиеленістер мен талаптардағы сәйкессіздік талаптарды әртүрлі амалмен анықтайтын мүдделі тұлға ретінде қол жеткізу өте қиын.

Жүйеге арналған пайдаланушылар талаптары функциялық және функциялық емес талаптарды толық техникалық білімі жоқ жүйе пайдаланушылары түсінетіндей сипаттау қажет. Негізінде, бұлар жүйенің тек сыртқы тәртібін ғана көрсетуі керек. Демек, егер сіз пайдаланушы талаптарын жазсаңыз, сіз бағдарламалық жасақтаманың жаргондарын, құрылымды және формальді жазылымдарын қолданбауларыңыз керек. Сіздер пайдаланушылардың талаптарын жай кестелер, формалар мен интуитивтік диаграммалармен бірге табиғи тілде жазғандарыңыз жөн.

Талаптар спецификациясына табиғи тілді пайдалану мәселелері

Спецификация үшін де пайдалы болып келетін табиғи тілдің икемділігі жиі жағдайларда бірқатар мәселелер тудырып келеді. Мұнда түсініксіз жазылған талаптар ауқымы салдарынан оқырмандар (жобалаушылар) талаптарды қате ұғуы мүмкін, себебі олар пайдаланушыға түрлі түсіндірмелер береді. Бірнеше талап мазмұнын бір сөйлемге біріктіру оңай, ал табиғи тіл талаптарын құру қиынға соғуы мүмкін.

http://www.SoftwareEngineering-9.com/Web/Requirements/NL-problems.html

Жүйе талаптары – жүйе дизайнын бастау нүктесі ретінде қарастыратын бағдарламалық жасақтама құраушылары қолданатын пайдаланушы талаптарының кеңейген түрі. Олар пайдаланушы талаптары жүйе арқылы қалай қамтамасыз ететінін толық түсіндіреді. Олар жүйе құрылымы келісімінің бір бөлігі ретінде қолданыла алады, сондықтан жалпы жүйенің сипаттамасы толық және анықталған болуы керек.

Негізінде, жүйе талаптары жүйенің сыртқы тәртібін және оның эксплуатациялық шегерулерін жай сипаттауы керек. Олар жүйенің қалай құралатыны мен жүзеге асыруы жайлы мазаланбауы керек. Бірақ талдап тексеру деңгейінде күрделі бағдарламалық жасақтама жүйесін нақты қарауын талап етеді, бұл даму жайында бар ақпаратты жою мүмкін емес. Бұған бірнеше себеп бар:

1. Мүмкін сіздерге талаптардың сипаттамасын құруға көмектесу үшін жүйенің негізгі архитектурасын құрауға тура келер. Жүйе талаптары неше түрлі жүйе құрамына кіретін жүйе бөліктері арқылы құрастырылады. *6*-және *18-тарауларда* айтылғандай, егер сіз жүйені құру барысында бағдарламалық жасақтама құрамдарын қайта қолданғыңыз келсе, архитектуралық анықтаулар өте маңызды.
2. Көп жағдайларда жүйе дамуды қолдайтын және жаңа жүйеде талаптарды қарастыратын бар жүйелермен қатынасуы керек.

3. Функциялық талаптарды (*13-тарауда* айтылған сенімділікке қол жеткізетін N-түрлі бағдарламалар) қанағаттандыру үшін белгілі архитектураны қолдану маңызды болуы мүмкін.

Пайдаланушы талаптары барлық жағдайда дерлік талаптар құжаттамасында сай диаграммалар мен кестелермен қамтамасыздандырылған табиғи тілде жазылады. Және де жүйе талаптары табиғи тілде жазыла алады, бірақ форма, графикалық жүйе модельдері немесе математикалық жүйе модельдеріне негізделген басқа белгілеулері қолданылады. *4.8-суретте* жүйе талаптарын жазуда қолданатын мүмкін белгілеулер көрсетілген.

Шарт	Баяндама
Қарапайым тілдегі сөйлемдер	Талаптар қарапайым тілде жазылады. Әр сөйлем бір ғана талапты суреттейді.
Қарапайым тілдегі құрылымдар	Талаптар қарапайым тілде алдын ала дайындалған үлгілер бойынша жазылады. Әр алаңда талаптың аспектілері суреттеледі.
Әрленім тілдері	Бұл шарт бағдарламалау тілі, операциялық модельдер сияқты талаптарды суреттейді. Қазіргі кезде бұл шарт сирек қолданылады.
Графикалық модельдер	Графикалық модельдер мәтінді түсініктемелер арқылы жүйенің функционалды талаптарын суреттеуге қолданылады. Мысал ретінде, UML және тізбек диаграммаларын қарастыруға болады.
Математикалық сипаттама	Бұл шарттар жиындар сияқты математикалық тұжырымда негізделеді. Бұндай сипаттамалар тұтынушының келісім шартты түсінуін қиындатады.

4.8-сурет. Жүйе талаптарының сипаттамасын жазу әдістері

Жағдайдың қалай өзгеруін көрсетуде немесе әрекет реттерін анықтауда графикалық модельдер ерекше маңызды. *5-тарауда* сипатталған UML реттік және қалыпты диаграммалар белгілі мәмілеге немесе оқиғаға жауап беру кезінде пайда болатын әрекеттердің реттілігін көрсетеді. Формальді математикалық сипаттамалар кейде талаптарды қауіпсіздікке немесе қауіпсіздігі өте қажетті жүйелерді анықтауда, бірақ көбіне басқа жағдайларда қолданылады. Мен бұл амалдарды *12-тараудағы* сипаттамаларды жазу үшін түсіндіріп жатырмын.

4.3.1. Табиғи тіл сипаттамалары

Табиғи тіл бағдарламалық жасақтама өндірісінің басынан бастап бағдарламалық жасақтамаға талаптар жазу үшін қолданылды. Бұл айқын, интуитивті және

универсальді болып келеді. Сондай-ақ бұлдыр ықтималдылығы бір қатарлы емес және оның мағынасы оқырманның фонына тәуелді. Нәтижесінде, талаптарды жазуда бірнеше баламалы амалдар ұсынысы болды. Бірақ олардың ешқайсысы толық қабылданбады. Табиғи тіл жүйе және бағдарламалық жасақтама талаптарын анықтап жазуда ең көп қолданылатын тіл болып қалды.

Талаптарды табиғи тілмен жазу барысындағы қайшылықтарды азайту үшін мен келесі мінездемелерді ұсынамын:

1. Барлық талап анықтамалары осы форматты ұстанатындай стандартты формат ойластырыңыз және оны қамтамасыз етіңіз. Форматты стандарттау қателіктерді азайтады және талаптардың тексерілуін жеңілдетеді. Мен талаптарды бір сөйлемде сипаттайтын форматта қолданамын. Мен мәлімелерді пайдаланушылардың әр талабын, оның не үшін ұсынылғанын түсіндіретін дәлелдеулермен байланыстырамын. Сондай-ақ сіздер талаптар өзгергенде кімге бару керектігіңізді білу үшін сол дәлелдеулер талапты (талаптар көзі) кім ұсынғаны жайлы ақпаратты толықтай қамтиды.
2. Міндетті және қажет талаптардың айырмашылығын табу үшін әрдайым тілді қолдан. Міндетті талаптар жүйені – қолдайтын және көбіне «міндетті» деген жазуды қолданатын талаптар. Қажет талаптар – елеулі емес және «міндетті» деген жазуды қолданатын талаптар.
3. Талаптардың маңызды бөліктерін таңдау үшін текстті бөліңіз (қаралау, курсивті немесе түсті).
4. Бағдарламалық жасақтама дамуының техникалық тілі әрқашанда түсінікті бола бермейді. Осылайша сіздер жаргонды қолдануларыңызды, қысқарған сөздер мен аббревиатураларды шегергендеріңіз жөн.
5. Мүмкін болған жағдайда, сіздер дәлелдеулерді әрбір пайдаланушы талаптарымен байланыстырып көрулеріңіз керек. Дәлелдеулер талаптардың не себепті қосылғанын анықтайды. Қандай өзгерістер қолайсыз екенін шешуге көмектесетін сияқты талаптар өзгерсе, өте пайдалы болады.

3.2 *Жүйе қан құрамының қант дәрежесін есептеп инсулинді тасымалдау керек.*

3.6 *Жүйе минут сайын өзін-өзі тексеріп, алдын ала анықталған шарттары бойынша тестілеу керек*

4.9-сурет. Инсулин сорғышы бағдарламасының талаптарының мысалдары

4.9-сурет осы принциптерді қалай қолдану керегін көрсетеді. Бұл *1-тарауда* таныстырылған, кіріктірме бағдарламалық жасақтамаға арналған автоматтандырылған инсулинді сорғының екі түрлі талабын қамтиды. Сіздер кітаптың веб-беттерінен толық инсулинді сорғы талаптарының сипаттамасын жүктей аласыздар.

4.3.2. Құрылымды сипаттамалар

Құрылымды сипаттамалар – талаптардың еркін жазылуы шектелген кезде және барлық талаптар стандартты түрде жазылуы керек кездегі жүйе талаптарының жазылу жолы болып есептеледі. Осындай ықпал көбіне, табиғи тілдің айқындылығы мен түсініктілігін қолдайды, бірақ кейбір біркелкіліктер сипаттамада қамтамасыз етіледі. Жүйелік талаптарды анықтауда құрылымды сипаттама тілінің белгілері дайын үлгілерді қолданады. Сипаттама баламалар мен итерацияларды көрсету үшін және көлеңкелеу мен түрлі қаріптерді пайдалану арқылы негізгі элементтерді көрсетуде бағдарламалық тілдің конструкциясын қолдана алады.

Робертсондар (Robertson and Robertson, 1999) өздерінің кітабында қарастырған VOLERE техникалық әдісінде талаптардың басынан бастап әр талаптың бір карточкаға жазуын ұсынады. Олар әр картада бірнеше аймақ болуын ұсынады, мысалы, дәлелдеу талаптары басқа талаптарға тәуелді, талаптардың дерекқорлары ақпараттарды қолдайды, т.б. Бұл *4.10-суретте* көрсетілген құрылымды сипаттама мысалына ұқсайды.

Жүйе талаптарын анықтауда стандартты үлгілерді құрылымды формада көрсету керек. Талаптар жүйеге әрекет ететін объект айналасында, жүйемен жасалатын немесе жүйемен өңделетін оқиға функцияларында құрылуы мүмкін. Мысалға алсақ, сипаттама негізіндегі форма, бұл жағдайда, инсулин дозасын қалай есептелуінің анықталуы қандағы қант деңгейі қауіпсіз группада болуы, бұл *4.10-суретте* көрсетілген.

Инсулин сорғышы/Бақылау Бағдарламасы/SRS/3.3.2

Функция	Инсулин мөлшерін бақылау: Қант мөлшерін сақтау.
Баяндама	Қант мөлшерінің қауіпсіз деңгейдегі 3-7-нің көлеміндегі тасымалдайтын инсулин мөлшерін анықтау.
Кіретін ақпарат	Қант мөлшері жайындағы байқаулар (r2), алдынғы екі байқау (r0 және r1).
Ақпарат көзі	Қант байқаулары сенсор арқылы оқылады. Алдынғы байқаулар жүйенің жадынан алынады.
Шығатын ақпарат	Тасымалданатын инсулиннің мөлшері.
Мұрат	Негізгі бақылау айналымы.
Әрекет	Қант мөлшерінің тұрақты болған жағдайында тасымалданатын инсулиннің мөлшері нөл болады.
Талаптар	Алдынғы екі бақылау. Өйткені, қант мөлшерінің өзгерісін анықтау қажет.
Алдынғы шарт	Инсулин қоймасы кемінде тасымалдауға болатын инсулиннің ең үлкен мөлшерін сақтап отырады.

Кейінгі шарт	r0 r1-мен алмасады, ал r1 r2-мен алмасады.
Эффекттер	Ешқандай

4.10-сурет. Инсулин сорғышының құрылымды сипаттамасы

Стандартты формалар функциялық талаптарды анықтаған кезде:

1. Функцияның немесе заңды тұлғаның сипаттамасы.
2. Кірістер мен олардың қайдан келетінінің сипаттамасы.
3. Шығыстар мен олардың қайда баратынының сипаттамасы.
4. Есептеуге немесе жүйеде қолданатын (‘сұраулар’ бөлігі) басқа субъектілерге керекті ақпараттар жайлы мәлімдеме.
5. Міндетті түрде қолдану керек әрекеттердің сипаттамасы.
6. Егер функциялық ықпал қолданылса, функция аталмай жатып ненің рас екенін (бастапқы) шарт, функция шақырылғаннан кейінгі растық (соңғы шарт) деп түсіндіріледі.
7. Операцияның қосалқы әсерлерінің сипаттамасы сияқты ақпараттарды қамтуы қажет.

Құрылымды сипаттардың қолданылуы табиғи тіл сипаттамаларының кей проблемаларын жояды. Сипаттаманың өзгермелілігі азаяды және талаптар нәтижелі ұйымдастырылады. Соған қарамастан, кейде әлі күнге дейін талаптарды анық және мағыналы етіп жазу, әсіресе, күрделі есептеулер (мыс., инсулиннің дозасын есептеу) анықталатын кезінде қиындықтар тудырады.

Бұл мәселені шешу үшін сіздер табиғи тіл талаптарына, мысалға, кестелер мен жүйенің графикалық моделін қолдану арқылы қосымша ақпараттар қосуларыңыз керек. Олар есептеулердің қалай жалғасып жатқанын, жүйенің күйі қалай өзгеретінін, пайдаланушылардың жүйемен қалай байланысатынын және әрекеттер реті қалай орындалатынын көрсетеді.

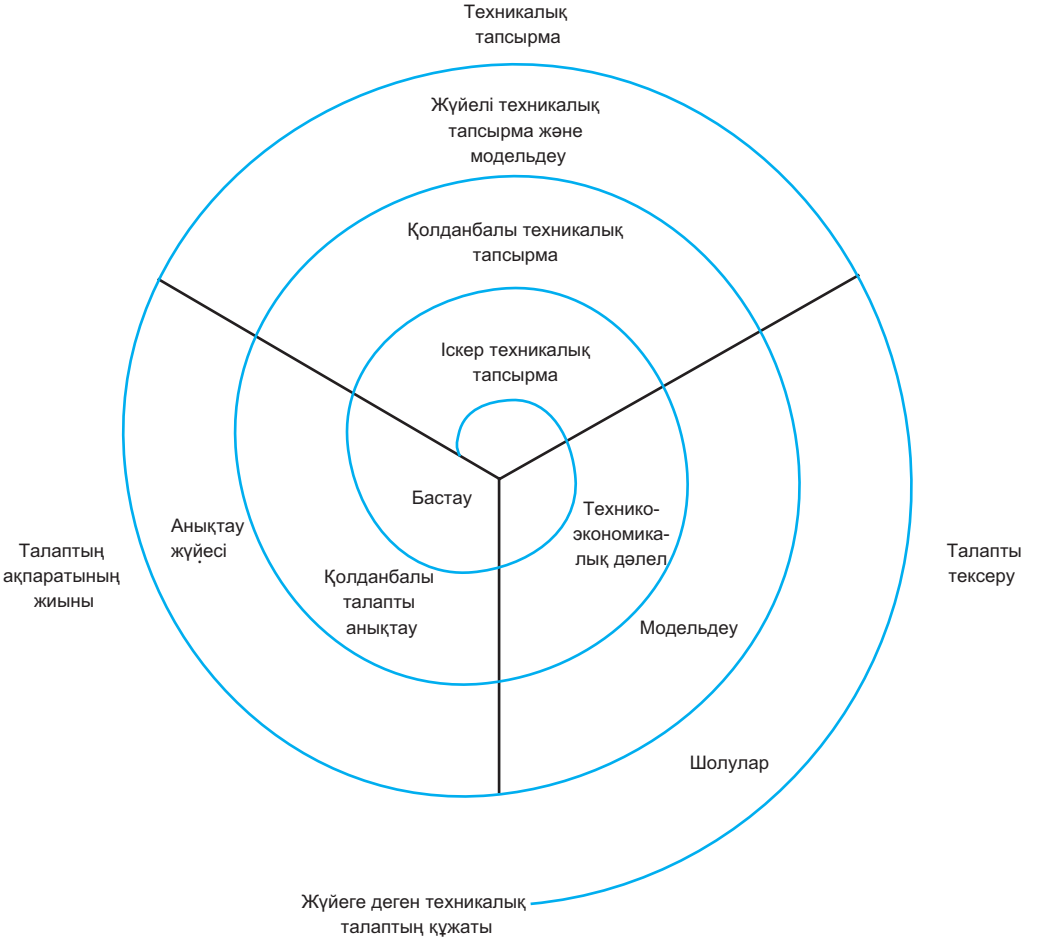
Шарт	Әрекет
Қант мөлшері азаюда (r2 < r1)	Мөлшер = 0
Қант мөлшері тұрақты (r2 = r1)	Мөлшер = 0
Қант мөлшері көбеюде және көбею дәрежесі азаюда ((r2-r1) < (r1-r0))	Мөлшер = 0
Қант мөлшері көбеюде және көбею дәрежесі тұрақты немесе көбеюде ((r2-r1) ⩾ (r1-r0))	Мөлшер = жуық шамамен ((r2 - r1)/4) Егер жуықталған нәтиже нөлге тең болса, онда мөлшер инсулиннің минималды мөлшеріне тең

4.11-сурет. Инсулин сорғышы есептеулерінің сипаттамасы

Кестелер бір қатар мүмкін баламалы жағдайлар бар кезде және олардың әрқайсысына керек әрекеттерді сипаттау кезінде өте пайдалы. Инсулин қанда қант деңгейінің өзгеру жылдамдығы талаптарының есептемелеріне негізделеді. Деңгейдің өзгеруі қазіргі және оның алдындағы көрсетілімдермен есептеледі. *4.11-суретте* қандағы қант деңгейінің қалай өзгеруінің сипаттамасы жеткізілетін инсулин мөлшерінің есептемесіне қолданылатын кестелік сипаттама берілген.

4.4. Өндіріс талаптарының үдерісі

Мен 2-тарауда талдағандай, өндіріс талаптарының үдерісі төрт жоғары деңгейдегі қызметтерді қамтиды. Егер жүйе бизнеске пайдалы болса (техникалық-экономикалық негіздеме), талаптарды аша отырып (анықтау және талдау), осы талаптарды бір стандартты формаға (сипаттама) айналдыру және клиент қалайтын



4.12-сурет. Талаптарды әзірлеу үдерісінің спиральді көрінісі

(қабылдау) жүйеде талаптарды шынында анықтайтын тексерістер зерттеуге көңіл тоқтатады. 2.6-суретте мұны реттік үдерістер ретінде көрсеттім. Бірақ істәжірибеде талаптардың құрастырылуы – қайталанбалы қызметтері алмасып келетін үдеріс.

4.12-суретте бұл алмасу екенін көрсетеді. Қызметтер жүйе талаптары шығыспен құжатталған спираль айналасындағы қайталанбалы үдеріс ретінде ұйымдастырылады. Уақыт пен ынта түсірілген әр қызметтің әр итерациясы үдерістің жалпы кезеңі мен жүйе түрлерінің құрылуына тәуелді. Үдерістің ең басында бар ынта дерлік жоғары деңгей бизнесі мен функциялық емес талаптарды және жүйе талаптарын түсіну үшін кетті. Кейінірек осы үдерісте, спиральдің сыртқы сақиналарында көбіне ынта толық жүйе талаптарын анықтау мен түсіну үшін қолданылады.

Бұл спиральді модель талаптардың әртүрлі деңгейде құрастырылып дамуына ықпал етеді. Спиральдің айналым саны өзгере алады, сондықтан спираль барлық немесе кейбір пайдаланушы талаптары анықталса тоқтауы мүмкін. Икемді даму кестенің орнына қолданылуы мүмкін, сондықтан талаптар мен жүйе талаптары бірге құрастырылады.

Кейбір адамдар өндірім талаптары нысанға-бағытталған анализ секілді құрылымдық әдіс анализінің қолдану үдерісі деп ойлайды (Ларман, 2002). Бұл өзіне жүйенің талдауын және жүйе сипаттамасы секілді қызмет ететін қолдануға болатын графикалық жүйе модель жинақтамаларының дамуын шақырады. Көп модель жинақтамалары деп, жүйе тәртібін және қосыша ақпарат сипаттамасымен белгіленуін, мысалы, жүйенің талап еткен өндірімділігін немесе сенімділігін айтады.

Жобаның жүзеге асырылуына қатысты зерттеулер

Жобаның жүзеге асырылуына қатысты зерттеу жұмысы ұзақ емес және мақсатты зерттеу үдерісінің басында орын алуы тиіс. Бұл негізгі үш сұраққа жауап беруі қажет: a) жүйе өз қызметін барлық тапсырмаларына өз үлесін келтіреді ме? b) жүйе өз қызметін ағымдағы технологияны пайдалану арқылы кесте мен бюджет бойынша орындайды ма? және c) Жүйені қолданыс үстіндегі басқа да жүйелермен біріктіруге болады ма?

Егер осы үш сұрақтың кез келгеніне жоқ деген жауап болса, жобаңызды бастаусыз қалдыруыңыз мүмкін.

http://www.SoftwareEngineering-9.com/Web/Requirements/FeasibilityStudy.html

Дегенмен, құрылымды методтар өндірім талаптарының үдерісінде белгілі рөл атқарады. Атап айтқанда, талаптарды анықтау адамдарға бағдарланған қызметтер мен қатал жүйе модельдеріне қойылған шегерулердің халыққа ұнамауы.

Барлық мүмкін жүйелерде талаптар өзгереді. Адамдар бағдарламалық жасақтаманың не жасайтынының қалауын жақсылап түсінулерінің дамуы-

на; жүйе өзгерістерін сатып алуын ұйымдастыруына; жүйенің жабдықтамалық жасақтамасы, бағдарламалық жасақтамасы мен орта ұйымдарын өзгертуде шықырылады. Өзгерген талаптардың басқару үдерісі *4.7-бөлімде* қарастырылған, ол талаптарды басқару деп аталады.

4.5. Талаптарды анықтау және талдау

Бастапқы техникалық-экономикалық негіздемеден кейінгі келесі кезең – өндірім талаптарының үдерісі талаптарды анықтау және талдау. Бұл жаттығуда бағдарламалық жасақтама құраушылары қосымша домендерді, қандай жүйе қызметтерін қолданатынын, керекті жүйенің, жабдықтамалық жасақтама шегерулерін және т.б. білу үшін клиенттер мен жүйенің түпкілікті тұтынушыларымен жұмыс жасайды.

Талаптарды анықтау және талдау ұйымда әртүрлі адамдардың бар болуын білдіреді. Мүдделі тұлғалар – жүйесі жүйе талаптарына тура немесе жанама әсер ететін кез келген адамдар. Мүдделі тұлғаларға жүйемен және ұйымдағы барлық адамдармен байланысатын адамдарды немесе тұтынушыларды да жатқыза аламыз. Басқа мүдделі тұлғалар өзге жүйелерді құрайтын бизнес менеджерлер, заттар саласы эксперттер, кәсіподақ таныстырушылары құрастырушылар бола алады.

Анықтау және талдау үдерістерінің модель үдерісі *4.13-суретте* көрсетілген. Әр ұйым қызметкер біліктілігі, құрастырылатын жүйе түрлері, қолданылатын стандарттар, т.б. сияқты жергілікті факторларға тәуелді жалпы модельдің жеке нұсқасы немесе данасы болады.

Үдеріс қызметтері:

1. *Талаптарды анықтау.* Бұл жүйенің мүдделі тұлғаларының талаптарын анықтау үдерісі. Осы бөлімде талаптарды анықтауда қолданатын бірнеше қосымша амалдар талқыланады.
2. *Талаптардың классификациялануы мен ұйымдастырылуы.* Бұл қызметтер құрылымды емес талаптар жинағын, топ талаптарын және олардың когерентті кластерлерінің ұйымдастырылуын өткізеді. Іс-тәжірибеде құрастырушылар талаптары мен архитектуралық дизайн қызметтерге толықтай тәуелді бола алмайды.
3. *Талаптардың басымдылығы мен келіссөздері*. Көптеген мүдделі тұлғалар шақырылғанда талаптарда шиеленіс шараларының болмауы мүмкін емес. Бұл қызмет талап басымдылығы, табылуы және талап шараларының шешілуі келіссөздерге байланысты. Әдетте, мүдделі тұлғалар кездесіп, келіспеушіліктерді шешіп және ымыралы талаптарға келісулері керек.
4. *Талаптар сипаттамасы*. Талаптар құжатталған және спиральдің келесі айналымына енгізілген. Формальды және формальды емес талаптарды құжаттау 4.3-бөлімде сипатталғандай алынуы мүмкін.

4.13-сурет. Талаптарды әзірлеу және анализдеу үдерісі

4.13-суретте әр қызметтің басқа қызметке тұрақты кері байланысының қайталанбалы үдеріс болуы талаптардың анықтауы мен талдауын көрсетеді. Жүйе үдерістері талаптардың ашылуынан басталып, талаптардың құжаттауымен аяқталады. Талаптар талдаудың түсініктілігі циклдің әр раундынан жақсарады. Талаптар толық құжатталған бойда жүйе аяқталады.

Мүдделі тұлғалар жүйесінен талаптардың анықталуы мен түсіндірілуі бірнеше жағдайлар бойынша қиын үдеріс болып келеді:

1. Мүдделі тұлғалар кейбір маңызды аталымдарды қоспағанда өздерінің компьютер жүйесінен нені қалайтынын көбіне білмейді; олар жүйеден не қалайтынын тұжырымдау қиын деп табады; олар қандай негіздемелердің бар жоғын білмегендіктен, мүмкін емес талаптарды сұрауы мүмкін.
2. Мүдделі тұлғалар жүйедегі табиғи талаптарды өздерінің терминдері мен күңгірт білімдерімен білдіреді. Құрастырушылардың талаптары клиенттер доменінде жұмыс тәжірибесінсіз бұл талаптарды түсінбеуі мүмкін.
3. Көптеген мүдделі тұлғалардың әртүрлі талаптары бар, олар талаптарын әртүрлі түсіндіруі мүмкін. Құрастырушылардың талаптары әр потенциалды талап көздерін және жалпы кескіндері мен шараларын ашуы керек.
4. Саясат факторлары жүйе талаптарына ықпал етуі мүмкін. Менеджерлер жүйеден нақты талаптарды сұрауы мүмкін, себебі олар ұйымдағы өз ықпалдарын нығайтуға рұқсат береді.
5. Экономикалық және бизнес орталарында болатын талдаулар динамикалық болып келеді. Бұл талдау үдерісінде сөзсіз ауысады. Жаңа талаптар басында кеңес өтпеген мүдделі тұлғалардан шығуы мүмкін.

Әрине, көптеген мүдделі тұлғалар басымдылық талаптарға неше түрлі көзқарастары бар және кейде олар қарама-қайшы болып келеді. Үдеріс барысында сіздер мүдделі тұлғалардың келіссөздерін тұрақты етіп ымыралы шарттар жүзеге асатындай қалыптастырғандарыңыз жөн. Әрбір мүдделі тұлғалардың толық

қамтамасыз ету мүмкін емес, бірақ егер мүдделі тұлғалар олардың көзқарастарын дұрыстап қарастырмаса, олар өндірістік талап үдерістерін бұзып тастауы мүмкін.

Талаптарды сипаттау кезеңінде бұрында құжатталып анықталған талаптар кей жағдайларда талаптарды ашуда көмектеседі. Қазіргі кезеңде ертеде болған жүйе талаптарының құжаттауы бос бөлімдер мен толық емес талаптар болып қарастырылуы мүмкін. Басқа жақтан қарағанда, талаптар түрлі амалдармен (мыс., электронды кестелер мен карталарда) толық құжатталуы мүмкін. Талаптардың карточкада жазылуы мүдделі тұлғаларға талап өзгерткенге және жұмыс ұйымдастырғанға өте қолайлы.

Көзқарастар

Көзқарас қызығушылықтары ұқсас ортақ мүдделі бірнеше тұлғалардан алынған талаптар жиынын жинақтаудың және ұйымдастырудың жолы болып табылады. Сондықтан, әрбір көзқарас жүйе талаптарының жиынын қамтиды. Көзқарастар соңғы тұтынушыдан, менеджерлерден және т.б. тұлғалардан болуы мүмкін. Олар өздерінің талаптары мен талдау талаптарын құруға қатысты ақпарат беретін адамдарды анықтауға көмектеседі.

http://www.SoftwareEngineering-9.com/Web/Requirements/Viewpoints.html

4.5.1. Талаптардың ашылуы

Талаптардың ашылуы – (кейде талаптарды анықтау деп аталады) сұралған жүйе және бар жүйелер жайында ақпараттар жинау және осы ақпарат негізінде пайдаланушылар мен жүйенің талаптарын айыру үдерісі. Ақпарат көздері талаптардың ашылу фазасында жүйенің мүдделі тұлғалары мен бірнеше жүйенің сипаттамалары құжаттамасын қарастырады. Сіздер мүдделі тұлғалармен сұхбат пен бақылау барысында және де сценарилер мен прототиптерді қолдануында мүдделі тұлғаларға жүйенің қандай болатынын түсіндіргенде араласасыздар.

Мүдделі тұлғалар жүйенің тұтынушы тұлғаларынан бастап сыртқы мүдделі тұлғалар менеджерлері арқылы тұрақты болатын жүйенің жарамдылығын шыңдайтындар диапазонында болады. Мысалы, жүйенің мүдделі тұлғалары емделушілердің психикалық денсаулығының жүйе ақпараттарына:

1. Емделушінің ахуалы мен емделуіне жауапкершілік атқаратын дәрігерлері.
2. Дәрігерлерден консультация алуды және кейбір емделу түрлерін басқаратын медбикелері.
3. Емшілердің тағайындауларын басқаратындар емхана басқарушылары.
4. Жүйенің жүктелуі мен қызмет етуіне жауапты ақпараттық технологиялар мамандары.
5. Ақпараттары жүйеде жазылған емделушілері.

6. Емдеушілерді қарауда жаңа этикалық жүйелеріне емханалық этикасына жауапты менеджерлері.
7. Жүйе ақпаратын басқара алатын денсаулық сақтау менеджерлері.
8. Қызметкерлердің емханалық жазбалары жазба процедураларының сақталып, нақты орындалатын жүйе ақпаратының қамтамасыздандыруына жауапты қызметкерлері туралы мәліметтерді толтырады.

Талаптар қосымша аймақтардан және жүйелермен қатынасын анықтайтын басқа жүйелерден шығатынын білеміз, бұл жүйеге мүдделі тұлғаларға қосымша болып есептеледі. Бұлардың барлығы талаптарды қабылдау үдерісінде маңызды болып есептеледі.

Осы әртүрлі талаптардың көздері (мүдделі тұлғалар, жүйе домені), әр көзқарас жүйеге желі бөлімдерінің талаптарын жүйе көзқарастары секілді көрсетеді. Проблемалардағы түрлі көзқарастар шараларды әртүрлі көреді. Соған қарамастан, олардың үміті толық тәуелді болып есептелмейді, бірақ әдетте, олар жалпы талаптар бар деп қайта анықталмай қалады. Сіздер бұл көзқарастардан жүйе талаптарының анықтау мен құжаттауды құрастыруға қолдана аласыздар.

4.5.2. Сауалнама

Жүйенің мүдделі тұлғаларымен болған формальді және формальді емес сауалнамалар – өндірім талаптары үдерісінің ең бір маңызды бөлігі. Бұл сауалнамаларда талаптарды құрастыратын топ мүдделі тұлғаларға олар қазір қолданып жүрген және құрастырылатын жүйе жайында сұрақтар қояды. Талаптар осы сұрақтардың жауабынан шығады. Сауалнама екі түрлі болады:

1. Мүдделі тұлғалардың алдынала дайындалған сұрақтарға жауап қайтаруы – жабық сауалнама.
2. Алдын ала қойылған ешқандай тақырыбы жоқ – ашық сауалнама.

Іс-тәжірибеде мүдделі тұлғалармен жүргізілген сауалнамалар – екеуінің де қоспасы. Мүмкін, сіздерге кейбір сұрақтарға жауап алуға тура келер, бірақ бұлар аз құрамдалған түрдегі жаңа сұрақтарды туындырады. Толықтай ашық пікірталастар жақсы жұмысқа әкеле қоймайды. Әдетте, жұмысты бастап сауалнаманы құрастырылатын жүйеге көңілін қаратып отыратын бірнеше сұрақ қою керек.

Сауалнамалар – мүдделі тұлғалардың не жасайтынын, жаңа жүйемен қалай байланысатынын және ағымдағы жүйеде кездесетін қиындықтарды жалпы түсіну. Адамдар өздерінің жұмыстарын жақсы көретіні жайлы айтады, сондықтан олар сауалнамаларға атсалысқанына қуанады. Бірақ заттар саласында қолданылатын талаптарды түсіну үшін сауалнамалар аса маңызды емес.

Заттар саласындағы білімді анықтау қиындығы сауалнама барысында екі себептен болады:

1. Барлық құрылыс мамандары заттар саласына тән терминология мен жаргон сөздерді қолданады. Заттар саласының талаптарын терминологиясыз талдау – олар үшін мүмкін емес зат. Әдетте, олар терминологияны нақты және таңдамалы құрастырушылардың түсінбеуіне өте оңай жолдарда қолданады.
2. Мүдделі тұлғалар кейбір заттар саласынан хабардар, сондықтан олар мұны түсіндіру қиын деп табады немесе олар мұны ең негізгі деп ойлағаннан ол жайлы айтудың қажеті де жоқ дейді.

Сұхбат алушылар, сондай-ақ ұйым талаптары мен шегерімдері жайлы білімді анықтауда тиімді метод емес, себебі мұнда ұйымдағы әртүрлі адамдардың арасында жіңішке өктем қарым-қатынас бар. Ұйымдық құрылымдарды жариялау ұйымдағы шындығында қабылданған мәселелермен келіспейді, бірақ распондентер теориялық құрылымына қарағанда нақтысын таныс емес адамдарға ашқысы келмеуі мүмкін. Жалпы айқанда, көп адамдар ереже бойынша талаптарға әсер тигізетін саяси және ұжыми сұрақтарды талқыламайды.

Нәтижелі сұхбат алушылардың екі сипаттамасы бар:

1. Ашық – талаптарға алдын ала идеялар ойластырылмаған және мүдделі тұлғаларды тыңдайтындар. Егер де мүдделі тұлға қызықтырарлықтай талаптар ойластырса, олар жүйе жайында өз ойларын ауыстыруға дайын.
2. Талдауда талаптарға ұсынылған трамплиндегі сұрақтарды қолдану үшін немесе жүйенің прототипімен бірге жұмыс жасай отырып, олар сұхбат берушіні сұрайды.

Сұхбат алушылардағы ақпараттар жүйенің бизнес үдерістерін құжаттауын немесе бар жүйелер мен бақылауларды мәлімдеулермен толтырады. Кейде, жүйе құжаттауындағы ақпараттардан бұрын сауалнама ақпараттары жүйе талаптарының жалғыз ақпарат көзі болуы мүмкін. Солай болса да, сауалнама маңызды ақпараттарды жарияламауына жауапты, сондықтан бұл басқа да талаптарды анықтау әдістерімен бірге қолданылуы жөн.

4.5.3. Сценарий

Әдетте, адамдар өмірден алынған мысалдарға абстрактілі сипаттамаларға қарағанда жақсы қарайды. Олар өздерінің бағдарламалық жасақтама жүйесімен қалай әрекет ететіндігінің сценарийін түсініп, оны сынай алады. Талаптарды құрастырушылар осы талқылаудан алынған ақпараттарды нақты жүйе талаптарын тұжырымдауда қолдана алады.

Талаптардың жобасын толықтап сипаттауда сценарий өте маңызды. Бұл мысалға әрекеттесу сессиясының сипаттамасы. Әр сценарий әдетте, бір немесе аз ғана мүмкін әрекеттесулерді қамтиды. Сценарийдің бірнеше түрі қарастырылады және олар жүйе жайындағы әртүрлі деңгейдегі тәптіштеулердегі ақпараттың түрлі

түрлерін қамтамасыз етеді. Тарихи төтенше бағдарламалауда қолданылады, бұл жайлы *3-тарауда* сценарий талаптарында айтылады.

Дейінгі пікірлер:
Науқас алдын ала қабылдаушымен кездесті. Оның аты, мекен-жайы, жасы туралы жеке ақпарат жүйеге енгізілді. Медбике жүйеге кіріп медициналық тарихты жинауда.

Қарапайым:
Медбике науқасты тегі арқылы іздейді. Егер жүйеде тектес науқастар кездессе керектісі аты немесе туған күні арқылы анықталады.
Медбике медициналық тарихты қосу үшін керекті мәзірді таңдайды.
Науқастың денсаулығының жағдайы (медбике жүйемен ұсынылған жағдайлардың біреуін таңтайды), алдын ала өткен кеңестері, қабылдап жүрген дәрі-дәрмектері (ұсынылған дәрілердің біреуін таңдайды), аллергия (еркін мәтін), өмір салты (қалып бойынша) жөніндегі ақпаратты енгізу үшін, медбике жүйемен ұсынылған көптеген мәзірлерден өтеді.

Ықтималды сәтсіздіктер:
Науқас жөніндегі ақпарат жоқ немесе табылмайды. Онда медбике жеке ақпаратты енгізіп, жаңа жазылым жасайды.
Науқастың жағдайы немесе дәрілерінің мәзірде жоқ болуы. Онда медбике “басқа” деген мәзірді таңдап сол ақапаратты мәтін ретінде енгізеді.
Науқас өзінің медициналық тарихы жөніндегі ақпаратты бере алмайды. Онда медбике оның себебін еркін мәтін ретінде енгізеді. Жүйе ақпараттың жеткіліксіз болғаны себебінен науқасқа кейбір функционалдың шектеулі жөніндегі хабарламаны қағазға басып шығарады. Осыған қол қойылып, бұл науқасқа беріледі.

Басқа әрекеттер:
Жазылымдар енгізіліп жатқан кезінде басқа қызметкерлермен талқыланады, бірақ олар оны өзгерте алмайды.

Жүйенің жұмысын аяқтағаннан кейінгі күйі:
Пайдаланушы енгізіліп тұрады. Науқастың жазылымы медициналық тарихымен бірге дерекқорға жазылады. Жазылым ақпаратты енгізген мебике және енгізілген күнін қамти отырып, жүйенің жадында толықтай сақталады.

4.14-сурет. MHC-PMS-тегі медициналық тарихты жинау мысалы

Сценарий жоспармен әрекеттесуден басталады. Үдерістерді анықтау барысында осы әрекеттің сипаттамасын толықтап құру үшін нақтылаулар қосылады. Сценарий өзіне:

1. Сценарий басталғанда жүйе мен пайдаланушылар күтетінінің сипаттамасын;

2. Сценарийдегі дұрыс ағын оқиғаларының сипаттамасын;
3. Дұрыс болмай қалуы және қалай жүзеге асыратынының сипаттамасын;
4. Бір уақытта болатын басқа шаралар жайлы ақпаратты;
5. Сценарий біту кезіндегі жүйе күйін сипаттауды қосады;

Сценарий анықтау негізінде мүдделі тұлғаларды жұмысқа шақырып, сценарийді анықтау мен осы сценарий қабылдайтын тәптіштеулерді қамтуы. Сценарий диаграммалар, экран суреттерімен қамтылған текст түрінде жазылуы мүмкін, сондай-ақ, бұдан құрылымды ықпал оқиға сценарийлері немесе қолданылған жағдайлар қолдануы мүмкін.

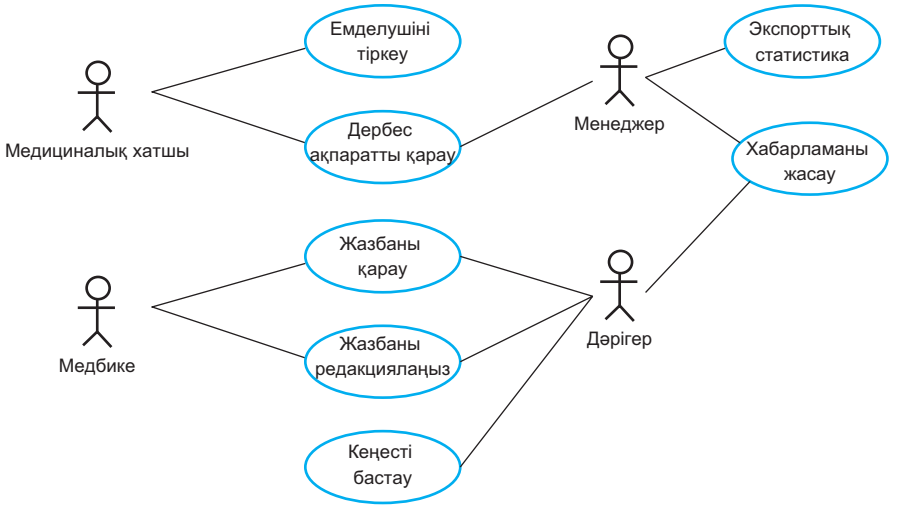
Мысал ретінде оңай сценарий тексті, емделушінің психикалық денсаулығының жүйелік басқаруы жаңа емделушіге деректерді енгізуде қалай қолданылады (4.14-сурет)? Жаңа емделуші емханаға келгенде, жаңа жазба емханалық тіркеушімен жасалады және оған жеке деректер (аты, жасы, т.б.) қосылады. Сосын медбике емделушіден сауалнама алады да емдік тарихын жинақтайды. Одан емделуші диагнозын қойып беретін және керек болса қажетті емделу курстарын жазып беретін дәрігермен бірінші консультациясын өтеді. Сценарий емдік тарихы жиналғанда не болатынын көрсетеді.

4.5.4. Үлгілі жағдайлар

Үлгілі жағдайлар – Objectory method (Jacobson et al., 1993) жұмысында бірінші таныстырылған техниканы ашу талаптары. Олар бірыңғайландырылған тіл модельдеуінің негізін құрушы болып келеді. Бұл жүйемен әрекеттесуді сипаттайтын қосымша ақпараттармен толықтырылады. Қосымша ақпараттар мәтін түріндегі сипаттама немесе реттілік немесе жағдай UML диаграммалары сияқты бір немесе бірнеше графикалық модельдер болуы мүмкін.

Үлгілі жағдайлар жоғары деңгейдегі үлгілі жағдай диаграммаларын қолдану арқылы құжатталады. Қолданудың көптеген түрлері барлық мүмкін болатын жүйе талаптарында сипатталатын әрекеттесулерді корсетеді. Бұл үдерісте адам немесе басқа да жүйе бола алатын субъектілер таяқша фигуралар секілді қарастырылады. Әр әрекеттесу эллипс түрінде келеді. Тағы да сызыққа бағыттағыш тілдер жалғануы мүмкін, бұл әрекеттесудің басталуын көрсету үшін. Емделушінің жүйе ақпараттарының кейбір үлгілі жағдайлары *4.15-суретте* сипатталған.

Онда сценарийлер мен үлгілі жағдай араларында қатты не тез болатын өзгерістер жоқ. Кейбір адамдар үлгілі жеке сценарий; басқалары Стивенс пен Пули (2006) айтқандай сценарий жинағын бір үлгілі жағдайға жинайды. Әр сценарий үлгілі жағдайлар арқылы бір жалғыз тізбек. Сонымен, бұл дұрыс әрекеттесуге сценарий және әр мүмкін ерекшеліктерге сценарий болар еді. Іс-тәжірибеде сіздер екі жолды да қолдануларыңызға болады.



4.15-сурет. MHC-PMS-ті пайдалану кескіні

Үлгілі жағдайларды жүйе мен оның пайдаланушылары немесе басқа жүйелер арасындағы жеке теңестірулерде қолданыңыз. Әрбір үлгілі жағдай мәтіндік сипаттамада құжатталуы тиіс. Олар UML-де сценарийді одан да толығырақ дамытатын басқа да модельдермен байланысуы мүмкін. Мысалы, *4.15-суретте* көрсетілген Консультацияны икемдеудің үлгілі жағдайларының қысқаша сипаттамасы мынадай болуы мүмкін:

> *Консультацияны икемдеу әртүрлі офисте жұмыс жасайтын екі не одан көп дәрігерлерге бірдей жазбаны бір уақытта көруге рұқсат етеді. Бір дәрігер алдындағы тізімнен онлайн адамдарды таңдай отыра шақырып консультацияны бастайды. Сосын емделушінің жазбасы олардың экранында көрсетіледі, бірақ ынтагерлік білдіріп отырған дәрігер тіркеуді өзгерте алады. Одан басқа, мәтіндік жазылулар әрекеттерді құрылымдастыру үшін жасалған. Бұл дауыстық байланыстарға жеке телефон-конференцияларының болуына сүйенеді.*

Сценарийлер мен үлгілі жағдайлар – жүйемен тікелей байланыстағы мүдделі тұлғалардың талаптарын анықтауда маңызды техника. Әрекеттесудің әртүрі үлгілі жағдай болып қарастырылуы мүмкін. Бірақ олар жүйемен әрекеттесуге бағытталғандықтан шегерулерді анықтауда немесе жоғары деңгей бизнесі мен функциялық емес талаптарды немесе заттар саласы талаптарын ашуда пайдалы емес.

UML объектіге бағытталған моделдеуге арналған де-факто стандарты, сондықтан үлгілі жағдайлар және қазір талаптарды анықтауда терең қолданатын үлгілі жағдайға негізделген анықтау қолданылады. Мен үлгілі жағдайларды *5-тарауда* талқылаймын және оларды басқа модель қастарында жүйе құрылысын құжаттауын көрсетемін.

4.5.5. Этнография

Оқшаулауда бағдарламалық жасақтама жүйесі жоқ. Олар әлеуметтік және ұйымдық шарттарда қолданылады және бағдарламалық жасақтама жүйе талаптары сол шарттарда болуы немесе шектелуі мүмкін. Осы әлеуметтік және ұйымдық талаптарды қанағаттандыру жүйенің сәтті болуына ықпал етеді.

Этнография – өндірістік үдерістерді түсінуде қолданатын және осы үдерістерге талаптар қолдауын алуға көмектесетін бақылаулық техника. Талдаушы өз-өзін жүйе қолданылатын жұмыс ортасына жүктейді. Күннен-күнге жұмыстар бақыланады және қатысушылар атсалысқан маңызды мәселелер жазбалары жазылады. Этнографияның мағынасы – ұйыммен табылған формальді үдерістерді емес, адамдардың жұмыс жасағандарының маңызды жолдарын тойтаратын күңгірт жүйе талаптарын анықтауға көмектесуі.

Көбіне адамдар өз жұмыстарында толық тұжырымдауды қиын санайды, себебі бұл оларға екінші табиғаттай. Олар өздерінің жұмыстарын түсінеді, бірақ жұмыстың басқа ұйыммен байланысын түсінбейді. Саясатты және ұйымдық факторлар жұмысқа әрекет етеді, бірақ жеке тұлғаға айқын еместер алдын ала қараушы байқағанда анық болады. Мысалы, жұмысшылар тобы топ мүшелері бір-бірінің жұмысын білетіндей және біреуі жоқ болса, оны уақытша ауыстыратындай өздері ұйымдастырыла алады.

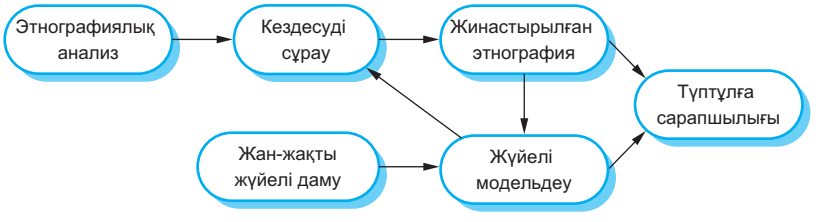
Suchman (1987) этнографияны алғаш рет офис жұмыстарын зерттеуде қолданылды. Ол жүйе автоматизациясының іс қағаздарын жүргізулеріне жобаланған жай модельдерге қарағанда негізгі жұмыс жаттығулары әлдеқайда мол, қиын және динамикалық болады. Crabtree (2003) зерттеудің кең шоғырын талдайды және содан бастап жүйе дизайнінде этнографияның қолдануын сипаттайды. Мен өз зерттеулерімде бағдарламалық жасақтама құрастыру үдерісі барысында оны құрастырушылар әдісі талаптарымен байланыстырып, этнография интеграциясының әдістерін зерттедім.

Этнография талаптардың:

1. Үдеріс анықтамалары бойынша айтылатын емес, адамдардың негізгі жұмыстарынан алынатын талаптардың жұмыс жасауы (Мысалы, авиадиспетчер ұшақтың қиылысатын ұшу траекторияларын анықтайтын хабарлау жүйесін өшіріп қоя алады, бірақ дұрыс процедураларда ол қолданылуы тиіс. Аспан кеңістігін басқаруда көмек болу үшін, олар әдейі ұшақты қарамақайшы жолға аздаған уақытқа қойып қояды. Олардың басқару стратегиясы осы ұшақтар мәселе тумағанша бір-бірінен алшақтауын қамтамасыз етуге белгіленген).
2. Талаптар қызметтестік пен басқа адамдардың қызметінен хабардарлықтан шығуы (Мысалы, авиадиспечерлер басқа диспечерлердің жұмысын білгендіктен басқару секторына қанша ұшақ шығатынын жорамалдап айтып бере алады. Сосын олар өздерінің басқару стратегиясын жобалама жүктемеге тәуелдене отырып ауыстыра алады. Осылайша, автоматтандырылған жүйе

диспечерлерге көрші секторлардағы жұмысты көруге болады) сияқты екі түрін айқындауда маңызды.

Этнография түп тұлғамен байланысуы мүмкін (*4.16-сурет*). Этнография түп тұлғаға талап анықтамалары қажет еместігін хабарлайды. Одан басқа, түп тұлғаны құрастыру этнографияға этнографпен ақылдасатын проблемалар мен сұрақтарды анықтауға негізделеді. Ол сосын келесі жүйені зерттеу (Sommerville et al., 1993) фазасында осы сұрақтарға жауап іздеуі тиіс.



4.16-сурет. Талаптарды анализдеудің этнографиясы және үлгілеуі

Этнографиялық зерттеулер тіптен басқа талаптарды анықтау методтары қалдырып кеткен маңызды деген үдеріс детальдарын айқындайды. Бірақ, бұл амал ұйымдық немесе заттар саласы талаптарына әрдайым жүре бермейді, себебі бұл соңғы пайдаланушыларға көзделген. Олар жүйеге қосылу керек жаңа функцияларды әрдайым анықтай алмайды. Демек, этнография өзін-өзі анықтауда толық ықпал етпейді және бұл талдау үлгілі жағдайлары сияқты басқа амалдарға қосымша ретінде қолданылуы тиіс.

Талаптар шолуы

Талаптар шолуы бұл жүйе тұтынушысы мен жүйе жасақтаушысы тараптарынан талаптар құжатын мұқият оқып,қателерін, қалыпсыз жерлерін және сәйкессіздіктерін тексерген бірнеше адаммен жүргізілетін үдеріс. Егер олқылықтар анықталып, тіркеуге алынған жағдайда, тұтынушы мен жасақтаушы анықталған мәселелердің шешілу жолдары туралы келіссөздер жүргізеді.

http://www.SoftwareEngineering-9.com/Web/Requirements/Reviews.html

4.6. Талаптарды қабылдау

Талаптарды қабылдау – клиент шынымен не қалайтын талаптар жүйесін нақты анықтайтын тексеру үдерісі. Бұл талаптардағы проблемаларды іздестірумен байланысындағы талдауға ұқсайды. Талаптарды қабылдау өте маңызды, себебі та-

лапты құжаттаудағы қателіктер даму үдерісі барысында немесе одан кейін табылса және жүйе пайдалануға енгізілсе қайта жөндеуде өте қымбат шығынға түседі .

Жүйеде өзгерістер енгізу барысында талаптардың проблемасын бекіту бағасы әдетте, конструкцияны жөндеу немесе кодтау қателіктерінен де қымбатырақ болады. Себебі әдетте, талаптарды өзгерту конструкция мен құрастырылу да ауыстырылуы керек екендігін білдіреді. Бұдан басқа, жүйе қайтадан тексерілуі қажет.

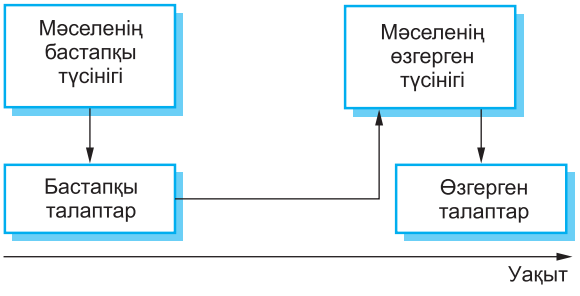
Талаптарды қабылдау үдерісі барысында және талаптарды құжаттауда талаптардың неше түрлі тексерістері жүзеге асырылуы тиіс. Бұл тексерістерге мыналар кіреді:

1. *Қабылдау тексерістері.* Пайдаланушылар жүйені кейбір қажет функцияларды орындау үшін керек деп ойлауы мүмкін. Бірақ, одан кейінгі ойлар мен талдау сұраныс болатын қосымша немесе әртүрлі функцияны орындауы мүмкін. Жүйе әртүрлі, өздерінің не бір ұсыныстары мен мүдделі тұлғалар ұйымына кері қайтарымсыз келісім, талап жинақтары бар мүдделі тұлғалардан тұрады.
2. *Реттік тексерістер.* Құжаттағы талаптар шатаспауы керек. Ол үшін мұнда келіспейтін шектеулер мен бірдей жүйе функцияларының түрлі сипаттамалары болмауы керек.
3. *Аяқталған тексерістер.* Талаптарды құжаттау барлық функциялары анықталған және жүйе пайдаланушылары арқылы қойылған шегерулер талаптарын игеруі тиіс.
4. *Құрастыру тексерістері.* Бар технологиялық білімді қолдана отырып, талаптардың орындалуына сенімді болу үшін оны тексеру керек.
5. *Тексерілімділік.* Тапсырысшы мен құрастырушы арасындағы келіссөздікті азайту үшін әрқашанда жүйе талаптары жазылуы тиіс, себебі олар тексеруге ыңғайлы. Бұл сіздің сынақтама жинақтарын жаза алатыныңызды білдіреді.

Мұнда бірнеше талаптарды жеке немесе бір-бірімен байланыстыра қабылдау методтары бар:

1. *Талаптарды шолу.* Талаптар жүйелі түрде, қателіктер мен келіспеушіліктің бар жоғын тексеретін шолушы топтарымен талданады.
2. *Түптұлға.* Бұл амалдағы қабылдау – соңғы пайдаланушылар мен тапсырысшыға көрсету сұрағында жүйенің атқарушы моделі. Олар осы модельмен, шынайы сұраныстарына жауап беретіндігін көру үшін, эксперимент жасай алады.
3. *Оқиға буындары.* Талаптар тексерілетіндей болуы керек. Егер сынақтамалар қабылдау үдерісінің бір бөлігі болып құрастырылса, бұл әдетте, талаптардың проблемалары болып қарастырылады. Егер сынақтаманы жобалау қиын және мүмкін емес болса, онда талаптарды жүзеге асыру қиындық туғызады және оны қайта қарау қажет болады. Пайдаланушылар талаптарынан сынақтаманы құрастырудан бұрын экстремалдық бағдарламалаудың ажыратылмайтын бөлігі болатын код жазылады.

Қабылдау талаптарымен байланысқан мәселелерді аса бағаламауларыңыз керек. Пайдаланушыларға жүйені жүріс барысында және бұл жүйе олардың жұмысымен қалай байланысатынын көрсете отырып ұсыну керек. Осындай түрдегі абстрактілі талдауды орындау тіпті, білікті компьютер мамандарына да қиын, ал жүйе пайдаланушыларына одан да қиынырақ болады. Нәтижесінде, талаптарды қабылдау үдерісінде талаптардың проблемаларын табу өте сирек оқиға. Кейінгі талаптарда талап құжаттамасымен келісілген соң, қателіктер мен түсініспеушіліктерді жөндеу барысындағы өзгерістерді шарасыз ету мүмкін емес.



4.17-сурет. Талаптардың эволюциясы

4.7. Талаптарды басқару

Ірі бағдарламалық жасақтама жүйесіне арналған талаптар әрдайым өзгергіп отырады. Бұның бір себебі әдетте, осы жүйелерде «анықтаулы» мәселелер, яғни толығымен анықталмаған мәселелердің құрастырылуы. Мәселелердің толық анықталуы мүмкін емес болғандықтан, бағдарламалық жасақтама талаптары толық болмауына тиісті. Бағдарламалық жасақтама үдерісі барысында мүдделі тұлғалардың проблеманы түсінуі толығымен өзгереді (4.17-сурет). Жүйе талаптары одан кейін өзгерген проблема көрінісін қайтару үшін дамуы қажет.

Сосын жүйе орнатылып және тұрақты қолданылған соң, жаңа талаптар болуы даусыз. Бұл пайдаланушылар мен жүйе тапсырысшыларына жаңа жүйенің бизнес үдерістеріне және жұмыстың бітуіне қандай салдар тигізетінін болжау қиын. Соңғы пайдаланушылар жүйеде тәжірибесі болған соң, олар жаңа тұтынушылықтар мен түптұлғалар ашады. Өзгерістердің шарасыздығының бірнеше себебі бар:

1. Бизнес және техникалық жүйе ортасында әдетте өзгерістер орнатылғаннан кейін келеді. Жүйенің басқа жүйемен байланыс жасауына маңызды болатын жаңа жабдықтамалық жасақтама енгізіледі және оның бизнес басымдылығы өзгеруі мүмкін (кейінгі өзгерістер жүйе қолдауында талап етеді), сондықтан жүйені міндетті түрде ұстау үшін жаңа заң шығару мен нормативтік актілер енгізілуі мүмкін.
2. Жүйеге төлейтін адамдар мен осы жүйенің пайдаланушылары бір адам болуы өте сирек. Жүйе тапсырысшысы ұйымдық және бюджеттік шегеру үшін

талаптар ұсынады. Олар соңғы пайдаланушылардың талаптарына қарсы болуы мүмкін, жеткізілгеннен кейін, егер жүйе өз мақсаттарын жүзеге асырса, пайдаланушылар қолдауына жаңа функциялар қосылуы мүмкін.

3. Ірі жүйелер, әдетте, пайдаланушылар ұйымының түр-түрінен тұрады және әр пайдаланушыда қарсы немесе қарсы болмайтын әртүрлі талаптар мен басымдылықтары болады. Соңғы жүйе талаптары мен тәжірибе арасындағы даусыз келісімшарт ауыстырылуға тиісті түрлі пайдаланушыларға берілген жиі-жиі қолдау балансын айқындайды.

Тұрақтылық және тұрақсыздық талаптары

Кейбір талаптар басқаларған қарағанда күмәнді болып келеді. Тұрақтылық талаптары мекеменің ішкі және баяу өзгеретін іс әрекеттерге қатысты талаптарды білдіреді. Тұрақтылық талаптары күрделі жұмыс қызметтерімен байналысты. Тұрақсыздық талаптары өзгеріске ұшырап тұрады. Әдетте олар мекеме жұмысының өзі емес, мекеме сол жұмысты қалай орындауын көрсететін қолдау қызметтеріне байланысты болады

http://www.SoftwareEngineering-9.com/Web/Requirements/EnduringReq.html

Талаптарды басқару – жүйе талаптарының түсінуі мен басқару өзгерістерінің үдерісі. Сіздерге талаптардың өзгеріс ықпалын бағалау үшін, жеке талаптарды қадағалап және тәуелді талаптар арасындағы байланысты қолдау керек. Сіздерге өзгеру ұсыныстары мен жүйе талаптары арасында байланыс жасау үшін формальді үдерісті анықтау қажет. Талаптарды басқарудағы формальді үдерістер талаптарды құжаттау жобасы пайда болғаннан бастап бірден басталуы тиіс. Бірақ сіздер талаптарды анықтау үдерісі барысында өзгермелі талаптарды қалай басқаруын жоспарлауды бастауларыңыз керек.

4.7.1. Талаптарды басқаруды жоспарлау

Талаптарды жоспарлау үдерісінде негізгі де бірінші кезең – жоспарлау. Жоспарлау кезеңі талаптарды бақылауда сұралатын деңгейлерді анықтайды. Талаптарды басқару кезеңі барысында сіздер мынаны шешулеріңіз жөн:

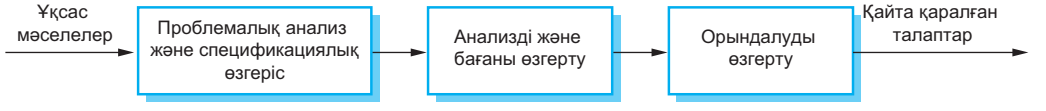
1. *Талаптарды анықтау.* Басқа талаптарға айқын нұсқау мен бақылау бағаларын қолданатын әрбір талап бір мағыналы болып анықталуы қажет.
2. *Басқару үдерісінің өзгерісі.* Бұл әрекеттесу – бағасы мен өзгеру құны болатын қызмет жинақтары. Мен бұл үдерісті келесі тарауда толығырақ талдаймын.
3. *Бақылау саясаты.* Бұл саясат әрбір талап пен талаптар арасындағы және жазылуы тиіс жүйе дизайны арасындағы қарым-қатынасты анықтайды.

Бақылау саясаты осылайша, осы жазылулар қалай сақталатынын анықтауы керек.

4. *Құрал қолдаулары.* Талаптарды басқару талаптар жайындағы үлкен көлемдегі ақпараттың үдерісін қосады. Жүйедегі талаптарды басқару мамандарынан жай дерекқор электронды кестелерге дейінгі диапазонда құралдар қолданылады.

Талаптарды басқаруда автоматтық қолдау және жоспарлау кезеңінде анықталған бағдарламалық жасақтама құралдарын талап етеді. Сіздерге қолдау құралы керек:

1. *Талаптарды сақтау.* Талаптарды құрастыру үдерісіндегі әрбір пайдаланушыға ашық болатын қауіпсіз басқарылатын деректерді сақтау орны талаптарын қолдау керек.
2. *Өзгерісті бақылау.* Өзгерісті бақылау (*4.18-сурет*) үдерісі – егер активті көмектесу құралдары ашық болса жабдақтайды.
3. *Бақылау басқарулары.* Жоғарыда талданғандай, бақылау басқаруларына арналған қолдау құралдары тиісті талаптар анықталғанда рұқсат етіледі. Табиғи тіл үдерісінің методтарында талаптар арасындағы мүмкін қарым-қатынастарды табуда көмектесуге қолданылатын кейбір құралдар қолайлы.



4.18-сурет. Талаптардың өзгерістерін басқару

Кішігірім жүйелерге талаптарды басқаруда мамандандырылған құралдар қолдануда маңызды емес. Талаптарды басқару үдерісі жазу процессорында электронды кестелер мен ЖК дерекқорындағы мүмкін объектілер қуатталуы мүмкін. Солай болса да, ірі жүйелерге тым мамандандырылған құрал қолдаулары сұралады. Мен талаптарды басқару үшін кітаптың веб-бетіне құралдар жайлы ақпараттарға сілтеме қостым.

4.7.2. Талаптардың өзгеруін бақылау

Талаптардың өзгеруін басқаруды (*4.18-сурет*) жүйе талаптарындағы өзгерістерге талаптардың құжаттаулары орындалғаннан соң қолдануға болады. Жаңа талаптарды енгізу орындаудағы шығындарды қамтыса, сіздер өзгеріс басқарулары өзгертуін шешулеріңіз керек. Формальды үдеріс қолданудың артықшылығы өзгерту басқармасында барлық өзгерістер алдын ала және талаптарды құжаттауда өзгерістер басқармалы түрде қарастырылады.

Мұнда өзгеріс бақылау үдерістерінің үш принциптік кезеңі бар:

1. *Проблеманы талдау және сипаттаманы өзгерту.* Үдерістер қандайда бір проблемадан, талаптарды анықтаудан немесе кейде, нақты өзгерістер енгізу ұсыныстарымен басталады. Бұл кезеңде шынымен жарамды екенін тексеру үшін проблемалар немесе өзгерту сұраныстары анықталады. Бұл талдау кері қарай сұранысты өзгертуге қайтарады.

2. *Талдау өзгерістері және бағасы.* Ұсынылған өзгерістер эффектісі бақылау ақпараттары мен жүйе талаптары жайлы жалпы білім арқылы бағаланады. Өзгерістер енгізудің бағасы талаптар құжаттамасындағы өзгерістер көз-қарасынан, қажет болса құрастыру мен өндіру жүйелерінен тұрады. Осы талдау біткен бойда талаптардың өзгерістерін жалғастыру керек еместігі шешіледі.

3. *Құрастырудың өзгерісі.* Талап құжаттауының құрастырылымын өзгерту қажет жағдайда құрастыру және орнату жүйелері қайта қарауға тиесілі. Сіз талаптар құжаттамасын ішіне кең емес жазылымдар сұрайтын өзгерістерді жазуға болатындай ұйымдастырасыз. Программа секілді құжаттағы өзгерістер ішкі сілтемелерді азайту барысында және құжаттау салаларын барынша модульді жасауда жүзеге асады. Осылайша, жеке арнаулар ешқандай да құжаттау бөлімдеріне әрекет етпей-ақ өзгертілуі немесе ауыстырылуы мүмкін.

Талаптарды есепке алу

Талаптар арасындағы арақатынас жолдарын, олардың қайнар көзі мен жүйе дизайнын ұсынылған өзгерістер себебіне және сол өзгерістердің жүйенің басқа да бөліктеріне ықпалын тигізуіне талдау жасай алатындай сақтауыңыз қажет. Неге?

http://www.SoftwareEngineering-9.com/Web/Requirements/ReqTraceability.html

Егер де жаңа талап жедел түрде жасалуы керек болса, жүйені әрдайым ауыстыру, одан кейін ретроспективті талаптар құжаттамасын өзгерту керек. Сіздер бұдан алшақ болғандарыңыз жөн, бұл даусыз болғандықтан, техникалық тапсырмалар мен жүйенің кері жүруіне әкеліп соғады. Жүйеге бір сәтте өзгерістер енгізіліп болғанда, осы өзгерістерді және құрастырылумен байланыспайтын өзгерістерді талаптар құжатамасына қосу жиі ұмытылып кетеді.

Икемді даму үдерістері, экстремальді бағдарламалау секілді, даму үдерісі барысында өзгертілетін талаптарды орындау үшін жаратылды. Бұл үдерістерде пайдаланушылар талап өзгерістерін ұсынғанда, бұл өзгеріс формальді өзгеріс басқаруының үдерісіне өтпейді. Пайдаланушылар басымдылықтарын орнатулары жөн, егер өзгеріс жоғары басымдылықта болса, жүйенің қандай функциясы келесі қайталануда тасталу керектігін шешу қажет.

НЕГІЗГІ ТҰЖЫРЫМДАР

- Бағдарламалық жасақтама жүйесіндегі талаптар – жүйенің не жасау керектігін және функциялануы мен пайда болуына шегерулерді анықтау.
- Функциялық талаптар – жүйені қамтамасыз ететін немесе кейбір есептеулер қалай орындалу керектігін анықтайтын қызметтер жайлы пікір.
- Фунциялық емес талаптар көбіне жүйе құрастырылымдарын және қолданатын даму үдерісін шегереді. Бұл өндірімге арналған талаптар, ұйымдық талаптар немесе ішкі талаптар болулары мүмкін. Бұлар көбіне пайда болған жүйе ерекшеліктерімен байланысады, сондықтан жүйеге толығымен тиесілі.
- Бағдарламалық жасақтама талаптарын құжаттау жүйе талаптары пікірімен келісілген. Ол тапсырушылар мен бағдарламалық жасақтама құрастырушылар жүйелері қолданатындай ұйымдастырылуы қажет.
- Талаптардың техникалық үдерістері техникалық-экономикалық негіздерді анықтау мен талдау талаптарын, техникалық тапсырманы, талаптарды қабылдауды және талаптарды басқаруды өзіне қамтиды.
- Талаптарды анықтау және талдау – бұл спиральді қызмет түрінде көрсетелітін талаптарды ашу, салаландыру, ұйым талаптары, келіссөздер жүргізу талаптары мен құжаттау талаптарының үдерісі.
- Талаптарды қабылдау өзімен анықтық, қайшылықсыздық, толық, шынайылылық пен тексерімділік талаптар үдерісін көрсетеді.
- Бизнес, ұйымдық және техникалық өзгерістер даусыз бағдарламалық жасақтама жүйесінің талаптарының өзгерісіне әкеледі. Талаптарды басқару осы өзгерістерді басқару мен бақылау үдерісі болып табылады.

ҚОСЫМША ӘДЕБИЕТТЕР

Software Requirements, 2nd edition. This book, designed for writers and users of requirements, discusses good requirements engineering practice. (K. M. Weigers, 2003, Microsoft Press.)

'Integrated requirements engineering: A tutorial'. This is a tutorial paper that I wrote in which I discuss requirements engineering activities and how these can be adapted to fit with modern software engineering practice. (I. Sommerville, IEEE Software, 22(1), Jan–Feb 2005.) http://dx.doi.org/10.1109/MS.2005.13.

Mastering the Requirements Process, 2nd edition. A well-written, easy-to-read book that is based on a particular method (VOLERE) but which also includes lots of good general advice about requirements engineering. (S. Robertson and J. Robertson, 2006, Addison-Wesley.)

'Research Directions in Requirements Engineering'. This is a good survey of requirements engineering research that highlights future research challenges in the area to address issues such as scale and agility. (B. H. C. Cheng and J. M. Atlee, Proc. Conf on Future of Software Engineering, IEEE Computer Society, 2007.) http://dx.doi.org/10.1109/FOSE.2007.17.

ЖАТТЫҒУЛАР

4.1. Автоматтандырылған жүйеде айқындалған талаптардың төрт түрін қысқаша сипаттап анықтаңыз.

4.2. Келесі пайымдауда өзіңізге талаптардың билеттік жүйе бөлімдеріне екі ұштылық пен қателіктерді ашыңыз:

Автоматтандырылған билеттік жүйе теміржол билеттерін сатады. Пайдаланушылар олардың барыстарын және кредиттік карталарының кірістері мен жеке тұлғалық нөмерін таңдайды. Теміржол билеттері мен және олардың құндары кредиттік картадан алынады. Пайдаланушы бастау нүктесін басқанда потенцалды бағытталған тізім активтендіріледі және пайдаланушының қай бағытты таңдағаны жайлы хабарлайды. Таңдалған соң пайдаланушыдан оның кредиттік картасына кірісін сұрайды. Оның дұрыстығы тексеріледі және сосын пайдаланушыға енгізуін сұрайды.

4.3. Осы тарауда сипатталған құралымды ықпалды қолдана отырып, жоғарыда айтылған анықтамаларды қайта жаз. Табылған белгісіздіктерді қолайлы түрде шеш.

4.4. Күтілген сенімділігі мен жауап беру уақыты анықталған билеттік жүйеге функциялық емес талаптар топтамасын жаз.

4.5. Табиғи тіл сипаттары стандартты түрде көрсетілген, осында берілген техниканы қолдана отырып, келесі функцияларды орындауда пайдаланушылардың талаптарын толықтырып жаз:

Өзіне кредиттік карта есептеуін қосатын қараусыз бензин сорғыш жүйесі. Бензин тапсырыс беруші картасын оқитын құрал арқылы қанша жанармай керектігін анықтайды. Жанармай сатылады және ол тапсырыс беруші есебінен алынады.

Банкоматта қолма-қол беру функциясы.

Мәтіндік үдерісте жазылудың дұрыстығын тексеру және коррекция жасау.

4.6. Инженердің функциялық және функциялық емес талап арасындағы байланыстарды жүйеде бақылай отырып талаптардың сипаттамасын қалай құрастыруын ұсын.

4.7.Талаптардың көрінісінде кім атсалысуы қажет? Талаптар көрінісі қалай ұйымдастыруға болатындығының моделін сал.

4.8. Авариялық өзгерістер жүйенің бағдарламалық жасақтамасы мүмкін талаптар өзгерістері қабылданбай тұрып өзгеруі мүмкін. Бұл өзгерістерді қабылдайтын кепіл, талаптар құжаттау мен енгізу жүйелерінде келіссөз болмайтындай үдеріс моделін ұсын.

ПАЙДАЛАНЫЛҒАН ӘДЕБИЕТТЕР

Beck, K. (1999). 'Embracing Change with Extreme Programming'. *IEEE Computer*, **32** (10), 70–8.

Crabtree, A. (2003). *Designing Collaborative Systems: A Practical Guide to Ethnography*. London: Springer-Verlag.

Davis, A. M. (1993). *Software Requirements: Objects, Functions and States*. Englewood Cliffs, NJ: Prentice Hall.

IEEE. (1998). 'IEEE Recommended Practice for Software Requirements Specifications'. In *IEEE Software Engineering Standards Collection*. Los Alamitos, Ca.: IEEE Computer Society Press.

Jacobson, I., Christerson, M., Jonsson, P. and Overgaard, G. (1993). *Object-Oriented Software Engineering*. Wokingham: Addison-Wesley.

Kotonya, G. and Sommerville, I. (1998). *Requirements Engineering: Processes and Techniques*. Chichester, UK: John Wiley and Sons.

Larman, C. (2002). *Applying UML and Patterns: An Introduction to Object-oriented Analysis and Design and the Unified Process*. Englewood Cliff, NJ: Prentice Hall.

Martin, D., Rodden, T., Rouncefield, M., Sommerville, I. and Viller, S. (2001). 'Finding Patterns in the Fieldwork'. *Proc. ECSCW'01*. Bonn: Kluwer. 39–58.

Martin, D., Rouncefield, M. and Sommerville, I. (2002). 'Applying patterns of interaction to work (re)design: E-government and planning'. *Proc. ACM CHI'2002,* ACM Press. 235–42.

Martin, D. and Sommerville, I. (2004). 'Patterns of interaction: Linking ethnomethodology and design'. *ACM Trans. on Computer-Human Interaction,* **11** (1), 59–89.

Robertson, S. and Robertson, J. (1999). *Mastering the Requirements Process*. Harlow, UK: Addison-Wesley.

Sommerville, I., Rodden, T., Sawyer, P., Bentley, R. and Twidale, M. (1993). 'Integrating ethnography into the requirements engineering process'. *Proc. RE'93,* San Diego CA.: IEEE Computer Society Press. 165–73.

Stevens, P. and Pooley, R. (2006). *Using UML: Software Engineering with Objects and Components, 2nd ed*. Harlow, UK: Addison Wesley.

Suchman, L. (1987). *Plans and Situated Actions*. Cambridge: Cambridge University Press.

Viller, S. and Sommerville, I. (1999). 'Coherence: An Approach to Representing Ethnographic Analyses in Systems Design'. *Human-Computer Interaction,* **14** (1 & 2), 9–41.

Viller, S. and Sommerville, I. (2000). 'Ethnographically informed analysis for software engineers'. *Int. J. of Human-Computer Studies,* **53** (1), 1369–96.

5.

Жүйелі модельдеу

Мақсаттары

Бұл тарау техникалық талаптарға және үдерістерді әзірлеу жүйесіне сай кейбір жүйелі модельдердің түрлерін құруға арналған кіріспе болып табылады. Берілген тараудың мақсаты:

- графикалық модельдердің бағдарламалық жүйелерде қалай ұсынылатынын түсіну;
- модельдердің алуан түрлері контексті әрекеттестікте, құрылымда және режимде неге міндетті және бастапқы модельдердің жүйесі болып келетінін түсіну;
- (UML) әмбебап модельдеу тіліне бірнеше диаграммалардың түрлері енгізілген және бұл диаграммалар жүйелі модельдеуде қалай қолдануы мүмкін;
- жүйенің құрылымдық және режимдік модельдерден автоматты түрде тудыратын, модельдеу техникасының бастамасында жатқан пікірлерді біліп-оқу.

Мазмұны

Жүйелі модельдеу жүйенің абстракты моделінің дамуының үдерісі, және әрбір модель осы жүйенің немесе келесі бір жүйенің көзқарасын тамашалайды. Жүйелі модельдеуде бастысы жүйені кейбір графикалық негіздермен көрсету болып келеді, солардың бірі – қазіргі кезде көп қолданылатын Модельдеудің Әмбебап Тілі. Дегенмен, мұнда жүйенің ресми (математикалық) модельдерін, көбінесе, мамандандырылған егжей-тегжейлі жүйені құруға болды. Мен графикалық модельдің Модельдеудің Әмбебап Тілін қолдануымен осы тарауда және формальді модельді *12-тарауда* қарастырдым.

Модельдер жүйеге талап қоюға көмектесу үшін талаптарды өңдеу барысында, инженерлердің жүйеге енгізуге қажетті жүйенің сипаттамасын жобалау үдерісінің барысында және енгізгеннен кейін жүйедегі құжаттардың құрылымын сипаттау үшін қолданылады. Сіздер модельдерді бар жүйе ретінде жобалай аласыздар және жүйелер келесі талаптарға сай жобалану керек:

1. бар жүйелердің модельдері техникалық талаптар кезінде қолданылады. Олар бар жүйенің жұмысын түсінуге көмектеседі және өздерінің мықты немесе әлсіз жақтарын талқыға салудың бастамасы ретінде қолданылуы мүмкін.
2. жаңа жүйелердің модельдері жүйенің басқа мүдделі жақтарына беріліп отырған талаптарды түсіндіруге көмектесу үшін техникалық талаптар қою барысында қолданылады. Инженерлер бұл модельдерді жобалы ұсыныстарды және жүйені жүзеге асыруға қажетті құжаттарды талқылау үшін қолданылады.

Жүйелі модельдің өзінен кейін бөлшектер қалдыруы оның ең басты аспектісі болып табылады. Модель өзін жүйенің баламалы шешімі ретінде емес, зерттеліп жатқан жүйенің абстракциясы ретінде ұсынады. Тамашасында, жүйенің шешімі субъектінің барлық мағлұматтарын көрсету керек. Абстракцялар ең басты сипаттамаларды әдейі жеңілдетеді және бөліп шығарады. Мысалы, сирек жағдайда бұл кітап газетке шығады, ал қойылым болған жағдайда бұл кітаптың абстракциясы ғана басылады. Егерде бұл ағылшын тілінен итальян тіліне аударылған болса, бұл баламалы шешімі болып табылатын еді. Аудармашының ниеті – ағылшын тілінде көрсетілгендей барлық мағлұматты көрсету.

Сіз жүйенің жан-жақты мәселесін шешетін алуан түрлі модельдер құра аласыз. Мысалы:

1. Сыртқы көз қарастан контексті және жүйенің қоршаған ортасын қайда модельдеу.
2. Әрекеттестік – жүйенің және оның айналасының немесе жүйенің компоненттерінің арасындағы байланысын модельдеу.
3. Құрылымдық көзқарастан, жүйенің ұйымдарын және жүйе арқылы өңделетін деректердің құрылымын қайда модельдеу.
4. Жоспарланған көзқарастан жүйенің динамикалық тәртібін қайда модельдеу және оның оқиғаға қалай әсер ететінін талқылау.

Мұндай келешектер Krutchen 4 + 1 архитектура жүйесінің түрлерімен (Kruchten, 1995) көптеген ортақ белгілерге ие, сондай-ақ сіз жүйенің архитектурасы мен ұйымдарын жан-жақты көзқарастан құжаттандыруыңыз қажет. Мен бұл 4+1 амалын *6-тарауда* талқылап кеткенмін.

Бұл тарауда мен нысанға-бағытталған модельдеуге арналған стандартты модельдеу тілі UML-дің (Буч және басқалар, 2005;.. Рамбо және басқалар, 2004) анықталған диаграммаларын қолданамын. UML-де көптеген диаграммалар және т.б. орналасқан, ол алуан түрлі жүйелі модельдеуді құруға демеу көрсетеді. Дегенмен, 2007 жылғы зерттеулер бойынша UML пайдаланушылардың көп бөлігі, диаграммалардың төмендегідей бес түрлі жүйенің бастамасы болатынын көрсетілген:

1. Әрекеттестік диаграммасы үдерісте және ақпаратты өңдеу барысында оның белсенділігін көрсетеді.
2. Өнегелік диаграммасы жүйелер арасындағы әрекеттестік пен олардың қоршаған ортамен байланысын көрсетеді.
3. Жүйелік диаграммасы актерлер мен жүйелердің және жүйелердің құрамдастарының арасындағы әрекеттестікті көрсетеді.
4. Класс диаграммасы жүйедегі объектілердің класын және осы кластар арасындағы арақатынасты көрсетеді.
5. Күйлер диаграмасы жүйенің ішкі және сыртқы жағдайларға қалай әсер ететінін көрсетеді.

Менде барлық UML диаграммаларын сипаттап отыруға орын болмағандықтан, тек осы бес түрлерінің жүйелі модельдеу қалай қолданылатынын тоқталып кетемін.

Жүйелі модельді құрған кезде, сіз графикалық белгілерді еркін қолдана аласыз. Модельдердің нақты және дәлме-дәл болуы сіздің оны қалай қолдануыңызға байланысты. Графикалық модельдердің:

1. Бар және ұсынылған жүйені талқылауға көмектесу үшін құрал ретінде.
2. Бар жүйені құжаттандыру тәсіді ретінде.
3. Құрылған жүйені іске асыру үшін жүйенің толыққанды сипаттамасы ретінде колдануға болады.

Бірінші жағдайда, моделдің мақсаты жүйені дайындауға қатысатын бағдарламаны әзірлеушілердің арасындағы пікірталасты ынталандыру болып табылады. Модельдер толыққанды болмауы мүмкін (сол мезгілдерге дейін болғандықтан олар талқылаудың маңызды сәттерін қамтиды) және олар модельдеудің белгілерін бейресми түрде қолдануы мүмкін. Ереже бойынша, бұлар көбінесе, «тез модельдейтін» (Амблер және Джеффрис, 2002) деп аталатын модельдерде қолданылады. Сіз қалағандай, жүйенің кейбір бөліктерін құру үшін модельдерді құжат ретінде қолданған кезде, олар толыққанды болмауы керек. Дегенмен, бұл модельдер дұрыс болу қажет, олар белгілерді дұрыс қолдануы және жүйенің сипаттамасын нақты көрсетуі керек.

Бірыңғайланған үлгілеу тілі (UML)

Бірыңғайланған үлгілеу тілі бағдарламалық жасақтамалар жүйесін үлгілеуге қажет болуы ықтимал әртүрлі 13 диаграмма түрінен тұрады. Бұл UML жасауға ұқсас нысанға бағытталған шартты белгілер жүйесі біріктірілген 1990-шы жылдардағы нысанға бағытталған үлгілеу еңбегінен пайда болды. Негізгі тексеріс (UML 2) 2004 жылы аяқталды. UML тілі бағдарламалық жасақтама жүйелерінің үлгілерін жобалауға арналған стандарттық тәсіл ретінде жанжақты қабылданды.

http://www.SoftwareEngineering-9.com/Web/UML/

Үшінші жағдайда, модельдің даму үдерісінің бастысы модель болып қолданған кезде, модельдің жүйесі толық және дұрыс болуы қажет. Оның басты себебі – олардың жүйені әзірлеу кодының бастамасы болып келуі. Сонымен, таяқ және ұшы бар нұсқар секілді әртүрлі мәні бар белгілерді шатастырмау үшін өте мұқият болуыңыз керек.

5.1. Контексті модельдеу

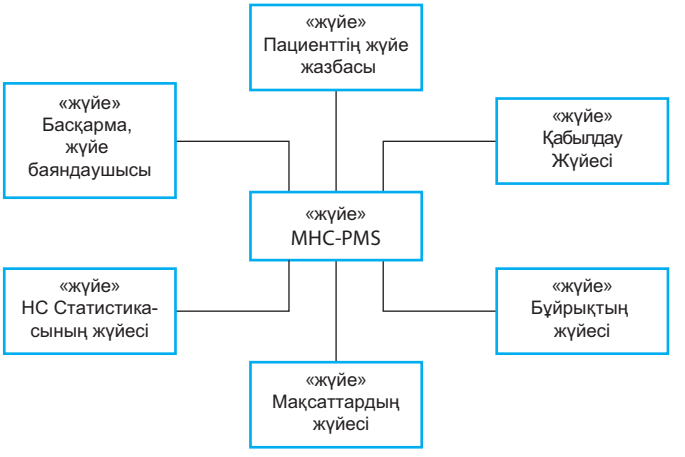
Жүйенің ерекше белгілерін айырықшалау кезеңінде, сіз жүйенің шекарасы жөнінде шешім қабылдауыңыз қажет.

Кейбір жағдайларда, жүйе мен оның қоршаған ортасымен арасындағы шекара біршама айқын. Мысалы, автоматтандырылған жүйе бар қодты немесе компьютерленген жүйені алмастырған жағдайда, жаңа жүйенің қоршаған ортасы бар жүйенің қоршаған ортасына сәйкес келеді. Басқа жағдайларда икемділік басым болады, сонымен талаптарды әзірлеу үдерісінде шекараның жүйе мен оның қоршаған ортасының арасында қандай рөл атқаратынын сіз өзіңіз шешесіз.

Мысалға, сіз психиатриялық көмек керек емделушілердің ақпараттық жүйесінің спецификациясын әзірлеп жатырсыз делік. Бұл жүйе психиатриялық денсаулық және емделу емханаларына келіп тұратын емделушілер туралы ақпаратты басқаруға арналған. Бұл жүйенің спицификациясын құрған кезде, сіз жүйенің тек дәрігерлердің кеңестеріне ғана назар аудару қажеттілігін (емделушілердің жеке бас ақпараттарын жинау үшін басқа да жүйелердң қолданумен) немесе ол емделушілердің жеке ақпараттарын да жинау керектігін шешуіңіз қажет. Басымдылық жағы, емделушілердің деректерін бөгде жүйелерден сұрастырған жағдайда қайталанудан сақтайды. Дегенмен, ең басты кемшілігі, басқа жүйелерді қолданған жағдайда деректерге рұқсат алу үдерісін бәсеңдету мүмкін. Егер бұл жүйелер болмай қалғанда, МНС-ПМС қолданылмайды.

Жүйенің шекарасының анықтамасы құндылықтардың пайымдауынан бос бола алмайды. Әлеуметтік және ұйымдық мәселелер жүйенің шекарасының жағдайы

техникалық емес факторлармен анықталуы мүмкін дегенді білдіреді. Мысалы, жүйенің шекарасы талдаудың үдерісі бір сайтта жасалып, әдейі орналасуы мүмкін.



5.1-сурет. МНС-PMS-тің мәтіні (контексі)

Бірнеше шешімдер жүйенің шекарасында жасалған шақта, талдау әрекетінің бөлігі осы жүйенің қоршаған ортасының контекстінің және тәуекелділіктің анықтамасы болып табылады. Ереже бойынша, қарапайым құрылысты жасап шығару осы әрекеттің алғашқы қадамы болып келеді.

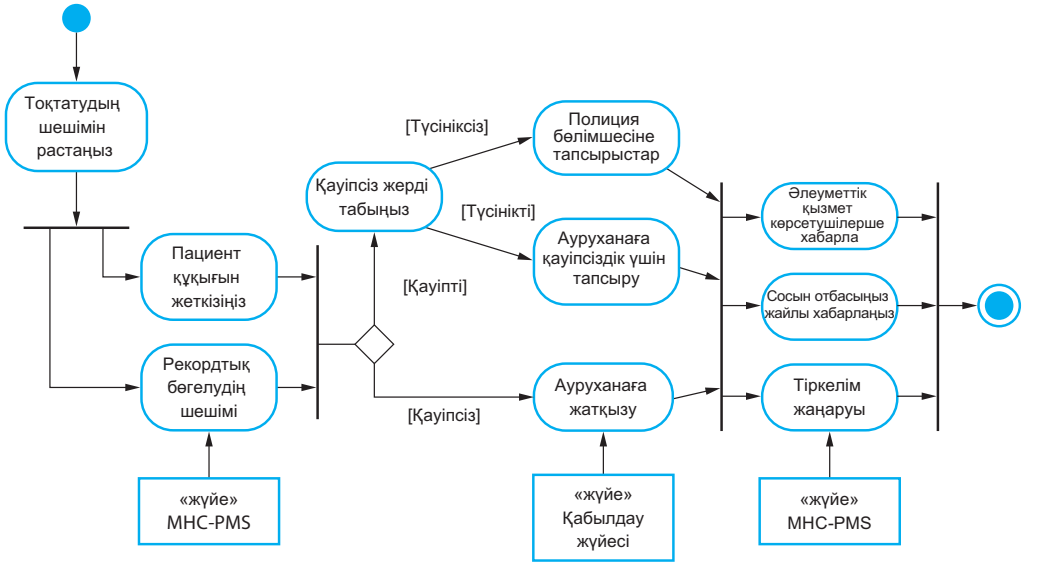
5.1-суретте емделушінің ақпараттық жүйесі және айнасындағы басқа жүйелерді көрсететін қарапайым контексі ұсынылған. *5.1-суретте* тағайындалған және жалпы жүйеге жазылған оның бірге бөлісетін ақпараттарының МНС-ПМС-ке қосылғандығын көруге болады. Ақырында, ол жүйе емделушілердің медициналық емдеу жоспарын жасауда қолданылады.

Әдетте, контекс модельдері өзіне бірнеше автоматтандырылған жүйелерді қосатындығын көрсетеді. Дегенмен, осы шақта анықталып жатқан жүйе мен оның қоршаған ортасының арасындағы қатынастардың түрлерін көрсетпейді. Сыртқы жүйелер деректерді өңдей алады немесе деректерді жүйеден қолданады. Олар жүйедегі деректермен алмаса алады, сондай-ақ олар желі арқылы тура қосыла алады немесе мүлде қосылмайды. Олар бір-бірін физикалық тұрғыда қиыстыра алады немесе бөлек ғимараттарда орналасады. Барлық бұл қатынастар талаптарға әсер етуі мүмкін, сондай-ақ жүйенің құрылымы анықталады және назарға алынуы қажет.

Сонымен, қарапайым контексті модельдер іскерлік-үдеріс модельдер секілді модельдермен қатар қолданылады. Олар нақты бағдарламалық жүйеде адамды және автоматтандырылған үдерістерді сипаттау үшін қолданылады.

5.2.-суретте МНС-ПМС қолданылған үдерістердің ең басты жүйе үдерісінің моделі бейнеленген. Кейде, психикалық ақаулардан зардап шегетін емделушілер басқаларға немесе өзіне қауіпті болуы мүмкін. Олар демек өздерінің қалауынан тыс емханада ұсталуы қажет. Мұндай ұсталулар жарияланбауға қатаң құқықтық

кепілдендіреді, мысалға, адамдар мерзімсіз, дәлелсіз себептермен ұсталмауы үшін емделушілердің денсаулығы қайта-қайта тексерілуі қажет. МНС-ПМС-тің бір міндеті – осындай кепілдемелердің іске асырылуын қамтамасыз ету.



5.2-сурет. МНС-PMS-ті пайдалану үлгісі

5.2.-суретте UML диаграммасының әрекетестігі бейнелеген. Сызбаның белсенділігі жүйелік үдерісті құрайтын әрекеттестікті және бір әрекеттестіктен келесісін басқаратын ағымдарды көрсетуге арналған. Үдерістің бастамасы толған домалақтың, ал соңы толған домалақтың ішіндегі келесі домалақтың бағытын көрсетеді. Домаланған бұрыштары бар тіктөртбұрыштар орындалу қажет нақты суб-үдерістердің әрекеттестіктерін бейнелейді. *5.2.-суретте* мен алуан түрлі үдерістерді әзірлеуге қолданылатын жүйелерді көрсеттім.

UML әрекеттестік диаграммасындағы нұсқаулықтар бір әрекеттестіктің келесіге ауысу жұмысының ағымын белгілейді. Жалпы түзу әрекеттің координациясын белгілеуде қолданылады. Сонымен, *5.2-суретінде* келесі емделушілердің туысқандарын ақпараттандыру және әлеуметтік көмек көрсету, сондай-ақ ұстау қажет жаңарту жұмыстары бір мезетте орындалады.

Нұсқаулықтар ағымдар алғанда емделушінің ахуалын негіздейтін сақшыларымен көрсетілуі мүмкін. *5.2-суретте* сіздер сақшылардың қауіпті және әлеуметке зияны жоқ емделушілердің ағымдарын нұсқап тұрғандығын байқай аласыздар. Әлеуметке қауіп тигізетін емделушілер бөлек қорғалған ғимаратта ұсталуы қажет. Дегенмен, өз-өзіне қол жұмсау әрекетіне баратын емделушілерді емхананың арнайы бөлімінде орналастырған жөн.

5.2. Әрекеттестік модельдеу

Барлық жүйелер қандай да бір әрекеттестікті қамтиды. Бұл пайдаланушының кіру және шығуын қамтитын пайдаланушының әрекеттестігі, дайындалып жатқан жүйе мен өзге жүйелердің арасындағы әрекеттестігі немесе жүйенің компоненттерінің арасындағы әрекеттестік болуы мүмкін. Пайдаланушының әрекеттестігін модельдеу маңызды, өйткені ол пайдаланушының талаптарын анықтауға көмектеседі. Жүйенің модельдеуі жүйеде пайда болу мүмкін коммуникативті мәселелердің әрекеттестігіне аса көңіл бөледі. Құрамдастардың әрекеттестігін модельдеу ұсынылып отырған жүйенің құрылысын қажетті өнімділікпен және сенімділікпен түсінуге көмегін тигізеді.

Бұл бөлімде, мен әрекеттестік модельдеудің өзара байланысқан екі тәсілін баяндап беремін:

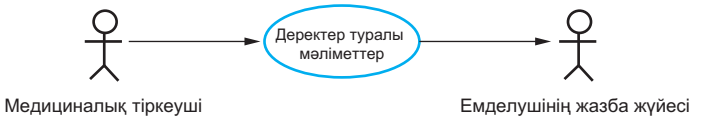
1. Өнегелік модельдеу, көбінесе, жүйе мен сыртқы субъектілердің арасындағы әрекеттестіктерді модельдеу үшін қолданылады.
2. Реттілік диаграммасы жүйенің құрылымдарының арасындағы әрекеттестіктер үшін қолданылады, дегенмен, ішкі факторлар да қосылуы мүмкін.

Өнегелік модельдеу және реттілік диаграммасы – тәптіштеудің алуан деңгейіндегі нағыз әрекеттестігі, сондықтан олар бірге қолданылуы мүмкін. Өнегелік модельдеудің жоғарғы деңгейіндегі әрекеттестіктің егжей-тегжейі реттілік диаграммасында сипатталуы мүмкін. UML диаграммасы әрекеттестікті модельдеу үшін қолданылу мүмкін диаграммалардың қатынастарын да қамтиды. Мен мұны бұл жерде талқыламаймын, өйткені олар сызбаның қосалқы түсініктемесі болып табылады. Шын мәнінде, кейбір құралдар реттілік диаграммасынан байланыстық диаграммаларын жасап шыға алады.

5.2.1. Өнегелік модельдеу

Өнегелік модельдеу бастапқыда Якобсон және басқалармен (1993) әзірленген, 1990 жылдары UML-дің алғашқы басылымына қосылған болатын (Рамбо және қаламдастары, 1999). Мен *4-тарауда* талқылағанымдай, өнегелік модельдеу анықтаудың талаптарын демеу үшін кеңінен қолданылады. Өнегелік пайдаланушының жүйеден күтетін талаптарын сипаттайтын қарапайым сценарий ретінде қабылдануы мүмкін.

Әрбір өнегелік сыртқы жүйелермен әрекеттесетіндігін өзінде қамтитын дискретті мәселелер болып табылады. Өзінің қарапайым түрінде, қатысушылардың әрекеті суретте өнегелігі түзу секілді эллипс тәріздес болып көрсетілген. *5.3-суреті* МНС-ПМС-тен жалпылама жүйеге емделушілердің саны туралы деректерді шығару мәселесін түсіндіретін МНС-ПМС-тің өнегесін көрсетеді. Бұл жалпылама жүйе МНС-ПМС-те жазылатын әрбір кеңестер жайлы деректерді емес, емделушілер жайлы жинақты деректерді қамтиды.



5.3-сурет. Дерек тасымалын пайдалану үлгісі

Есіңізде болсын, бұл жағдайда деректердің тасымалын қамтамасыз ететін оператор мен емделушілерді тіркейтін жүйе секілді екі қатысушы қолданылады. Айшықтаманы белгілеу бастапқыда адамдардың әрекеттестігін айқындау үшін әзірленген болатын, бірақ дегенмен, қазір ол өзгеде сыртқы жүйелерді және құралдарды түсіндіру үшін қолданылады. Ресми түрде, өнегелік диаграмманың нұсқаулығы UML-дің нұсқаулықтар секілді хабарламалардың ағымдарын, бағытын көрсететіндей нұсқаулығы жоқ түзулерді пайдалану қажет. Әлбетте, қолданған жағдайда хабарлама екі бағыттан да өтеді. Дегенмен, *5.3-суретте* нұсқаулықтар медициналық тіркеушінің транзакцияларды ынтығерлікпен өзгерткенде бейресми қолданылады және деректер емделушінің жазбаларының жүйесіне жіберіледі.

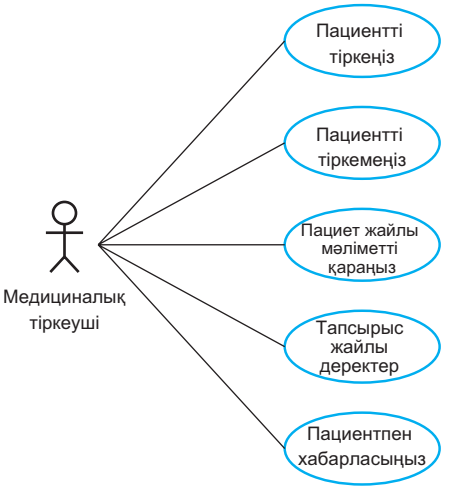
Өнегелік диаграммасы әрекеттестікті қарапайым көрсетеді, сондықтан сізге не туралы болып жатқанын түсіну үшін толыққанды ақпарат беру қажет. Бұл бөлшек қарапайым мәтіндік түсініктеме, кестедегі құрылымдық түсініктеме немесе төменде айтылғандай реттілік диаграммасы болуы ықтимал. Сіз өнегелікке айырықша қолайлы үлгіні таңдадыңыз және сіз ойлаған бөлшек болашақта модельде талап етіледі. Мен айырықша пайдалы стандартты кестелік үлгені іздестірдім. *5.4-суретте* «Деректерді табыстау» нұсқасының кестелік түсініктемесі қолдануын көрсетеді.

MHC-PMS: Дерек тасымалы	
Кейіпкерлер	Медициналық қабылдаушы, науқастар жазылатын жүйе (PRS)
Баяндама	Қабылдаушы деректерді MHC-PMS-тан науқастар жазылатын (PRS) жүйесіне тасымалдай алады
Дерек	Науқастың жеке ақпараты, қорытынды
Ынта	Медициналық қабылдаушының әзірлеген әмірі
Жауап	PRS-тың жаңғарғаны туралы құптауы
Түсінік	Науқастың жеке мәліметіне және PRS-ке қол жеткізу үшін қабылдаушының арнайы рұқсаты болуы керек

5.4-сурет. 'Дерек тасымалының' сипаттамасы

4-тарауда талқылағанымдай, құрамдас өнегелік диаграммалар өнегеліктердің алуан түрінің санын көрсетеді. Кейде мүмкін болатын жүйемен әрекеттестіктердің барлығын бір құрамдас өнегелік диаграммасына қосуға болады. Дегенмен, бұл қолданудың бірнеше санынан жүзеге аспауы әбден мүмкін. Мұндай жағдайда, сіз

әрбіреуі өнегелікпен байланысатынын көрсететін бірнеше сызба әзірлей аласыз. Мысалы, *5.5-суретте* «Медициналық Тіркеуші» қатысушының қосуымен МНС-ПМС-те қолданылатын барлық нұсқалары көрсетілген.



5.5-сурет. “Медициналық қабылдаушы” рөлінің баяндамасы

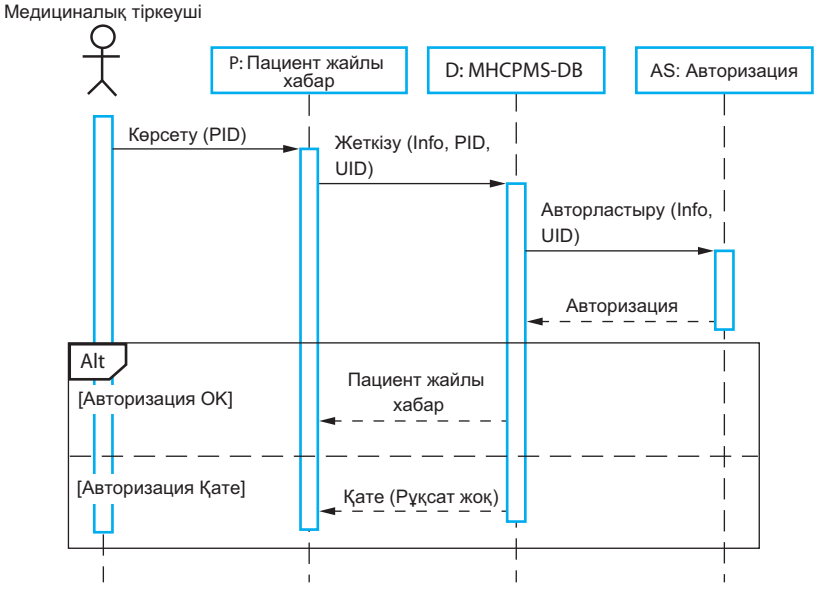
5.2.2. Реттілік диаграммасы

UML-де реттелік диаграммасы негізінде субъектілер мен объектілердің жүйемен және объектілердің өзара арасындағы әрекеттестіктерді модельдеу үшін қолданылады. UML көптеген алуан түрлі әрекеттестіктерді модельдеуге рұқсат беретін, реттелік диаграммаға арналған синтаксистер қорына бай. Мен мұнда барлық мүмкін болатын түрлерді баяндамаймын, тек қана осы диаграмманың басты түрлеріне ғана тоқталып кетемін.

Тақырыптың атауына сай, реттілік диаграммасы өнегелікті немесе өнегеліктің мысалын қолданған кездегі болатын әрекеттестіктердің реттілігін көрсетеді. *5.6-суретте* реттілік диаграммасының басты белгілерді бейнелегенінің мысалы келтірілген. Бұл модельдеу диаграммасында медициналық тіркеушінің емделушілердің кейбір ақпараттарын көре алатын, Емделушілерді бақылау өнегелігінің әрекеттестігі қосылған.

Жоғарыда келтірілген диаграмманың бөлігіне қатысатын объектілер мен субъектілер көлденеңнен үзік-үзік сызықпен көрсетілген. Объектілердің арасындағы әрекеттестер аннотациялық нұсқаулықтармен белгіленген. Үзік-үзік сызықтардың үстіндегі тіктөртбұрыштар қарастырылып отырған объектінің құтқарушы дөңгелегін бағыттап көрсетеді (бұл уақытта, объектінің нұсқасы есептеуде қатысады). Сіз реттілік әрекеттестікті бастан- аяқ оқыдыңыз. Нұсқаулықтардағы аннотациялар объектілердің шақыруына, оның параметрлеріне және қайтарылатын мәніне нұсқау көрсетеді. Мен бұл мысалда, сондай-ақ

белгілер баламалы белгілерді көрсету үшін де қолдандым. Alt атты жолақ шаршы жақшаларда көрсетілген шарттармен қолданылады.



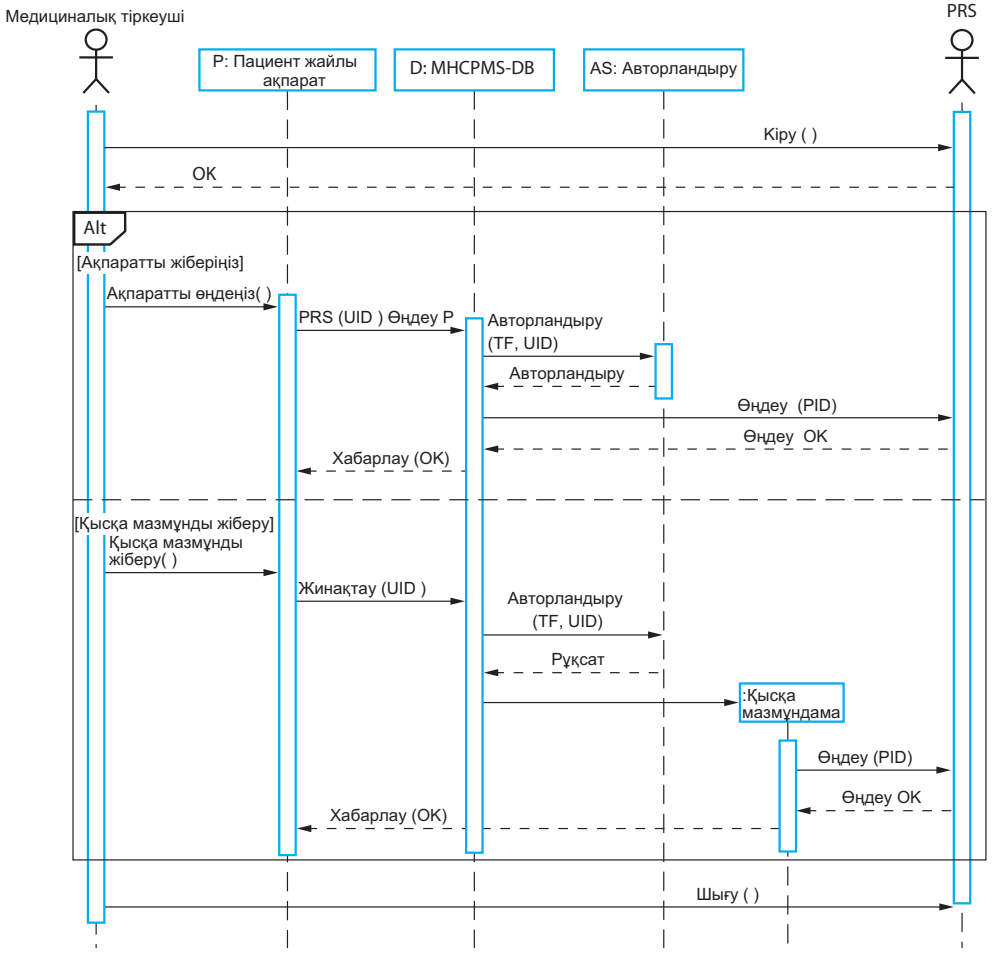
5.6-сурет. Науқастың ақпаратын көрсететін тізбектік кескіні

Сіз *5.6-суретті* келесі кейіпте оқи аласыз:

1. Медициналық тіркеуші емеделушінің сәйкестендіргішін **PID-қа** қоя отырып, **P** нұсқасы **PatientInfo** объект классындағы **ViewInfo** амалын шақырады. **P** емделушінің ақпаратын көрсететін нысан түрінде суреттелетін қолданушының интерфейсінің объектісі болып табылады.
2. Мысалы, **P** қауіпсіздікті тексеру үшін қажетті сәйкестендіргішті тіркеушіге қоя отырып, ақпаратты қайтаруға арналған деректер қорын шақырушы.
3. Пайдаланушының осы әрекетке авторластырылғанын деректер қоры авторластырылу жүйесімен тексереді.
4. Егер рұқсат етілсе, емделушінің ақпараты қайтарылады және пайдаланушының экранында үлгі пайда болады. Рұқсат етілмеген жағдайда, қате туралы хабарлама қайтарылады.

5.7-сурет сол жүйедегі қосымша екі мүмкіндіктерді бейнелейтін реттілік диаграммасының екінші мысалын сипаттайды. Бұл – жүйедегі қатысушылар мен реттілік операциялар шегіндегі объектілерді құру арасындағы тікелей байланыс. Бұл мысалда PRS-та енгізілу қажет жиынтық құжаттар сақталуға арналған негізгі объектінің келбеті құрылады (емделушіні жазу жүйесі). Сіз бұл сызбаны келесі жағдайда оқи аласыз:

1. Администратор PRS-қа кіреді.
2. Екі жол бар. Олар емделуші туралы жаңартылған ақпаратты тікелей PRS-дан тасымалдауды және жиынтық денсаулық деректерін МНС-ПМС-тан PRS-қа тасымалдауын қамтамасыз етеді.
3. Әрбір жағдайда тіркеушінің рұқсаттамасы авторластырылу жүйесін қолдана отырып тексеріледі.
4. Жекебас ақпарат тікелей пайдаланушы интефейсінің объектісінен PRS-қа жіберіле алады. Сонымен қоса, хаттама деректер қорынан әзірленуі мүмкін, және ол жазба кейін жіберіледі.
5. Тапсыру аяқталғанда PRS пайдаланушының жүйедегі жағдайы мен жүйеден шығуы туралы хабарламаны береді.



5.7-сурет. Дерек тасымалының тізбектік кескіні

Нысанға бағытталған талаптар талдауы

Нысанға бағытталған талаптар талдауында нысан кластарын пайдалану арқылы шынайы әлем болмысын үлгілей аласыз. Сіз нысан үлгілерінің әртүрлі түрлерін жасап шығара аласыз және оған қоса, нысан кластарының бірімен бірінің өзара байланысын, нысандардың басқа нысандарға түрлену жолдарын, нысандардың басқа нысандармен өзара әрекеттесу жолдарын және тағы басқа амалдарды көрсете аласыз. Бұл жүйе туралы әрбір бірегей ақпарат ерекшеліктері анықталады.

http://www.SoftwareEngineering-9.com/Web/OORA/

Егер сіз реттілік диаграммасын кодты немесе толыққанды құжаттарды генерациялау үшін қолданбасаңыз, онда сіз өзіңізді бұл сызбалардағы барлық әрекеттестіктерге қоспауыңыз керек. Егер сіз даму үдерісінде техникалық талаптарды және жоғарғы деңгейдегі дизайнды қолдау үшін ертеректе жүйелі модельдеуді құрған болсаңыз, жүзуге асыру шешіміне қатысты әрекеттестіктер көп болды. Мысалы, *5.7-суретте* пайдалаушының сәйкестендіргішін қалай алалатындығы және кейінге қалдыруы мүмкін шешімдерді тексеруі көрсетілген. Жүзеге асқанда, әрекеттестіктің пайдаланушы объектісімен байланысы болуы мүмкін, дегенмен, бұл берілген кезеңде аса маңызды емес және реттілік диаграммасына қосылуы керек.

5.3. Құрылымды модельдеу

Бағдарламалық жасақтаманың құрылымды модельдеуі – құрамдастың көз қарасынан жүйенің ұйымдастыруының бейнесі, және осы жүйенің құраушылармен және оның өзара қатынасы. Құрылымды модельдеу құрылымды жүйенің дизайны ретінде көрсететін статикалық модельдер немесе орындау барысында жүйенің ұйымын көрсететін динамикалық модельдер болуы мүмкін. Бұлар бірдей емес – жүйенің динамикалық ұйымы өзара әрекеттесетін ағымдардың жиынтығы ретінде модельдің статикалық құрамдастарынан айырыша ерекшеленеді.

Сіз жүйенің құрылысын талқылаған және әзірлеген кезде, жүйенің құрылымды модельдерін құра аласыз. Құрылыстық әзірлеу бағдарламалық жасақтаманы және UML-дің құрамдастарын дайындау тақырыбында аса маңызды болып келеді. Құрылыстық модельдерді келтіре отырып, өрістету қаптамалары мен диаграммаларының бәрі қолдануы ықтимал. Мен құрылыстың және құрылыстық модельдеудің алуан түрлі аспектілерін *6-, 18-* және *19-тарауларда* қамтимын. Бұл бөлімде мен бағдарламалық жасақтаманың жүйесіндегі статикалық класс объектілерін модельдеу үшін диаграммалар кластарын пайдалануына тоқталамын.

5.3.1. Диаграммалар класы

Диаграммалар класы жүйедегі нысанға-бағытталған модельдеуді әзірлеу барысындағы жүйедегі кластар және осы кластар арасындағы байланысты көрсету үшін қолданылады. Еркінде, объектілер классы жүйенің бір түрінің объектісін жалпы анықтама ретінде қарастыруға болады. Ассоциация кластар арасында байланыстың бар екенін көрсететін осы кластар араларын байланыстыратын түйін болып табылады. Демек, әрбір класта өзімен байланысқан кластар туралы ақпараттар бар.

Модельдердің бағдарламалық жасақтаманы дайындау үдерісінің ертерек қадамындағы әзірлеу барысында, объектілер өзін әлдебір нақты әлемдегі емделушілер, дәріқағаз, дәрігер және басқалар секілді көрсетеді. Мен мұнда объектілердің нағыз әлемдегі модельдеудің талаптардың бөлігіне немесе бағдарламалық жасақтаманың дайындау үдерісінің бастамасына тоқталып кетемін.

UML-дағы диаграммалар классы алуан түрлі талдап тексеру деңгейінде айтылуы мүмкін. Модельді дайындау барысында, бірінші кезеңде, әдетте жан-жаққа қарау, басты объектілерді анықтау және оларды кластар ретінде қарастыру. Оларды жазудың ең оңай тәсілі – кластардың атын қорапшада жазу. Сондай-ақ сіз ассоциациялардың барын кластар арасында түзу жүргізу арқылы айшықтай аласыз. Мысалы, *5.8-суретте* екі класты көрсететін қарапаайым класс келтірілген, емделуші мен емделуші сырқауының тарихы арасындағы ассоциациямен келтірілген.

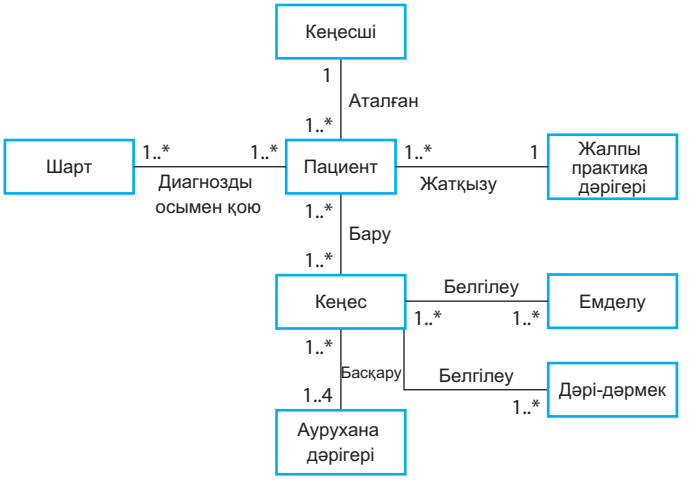
5.8-сурет. UML кластары және олардың өзара байланысы

5.8-суретте мен диаграмма класының тағы бір ерекшелігін, яғни бір бірлесуде көптеген объектілердің қатыса алатынын көрсету мүмкіндігін суреттеймін. Бұл мысалда әрбір ассоциацияның ұшы 1 таңбасымен белгіленеді, демек, ол бұл класста объектілердің арасында 1:1 қатынас бар екендігін білдіреді. Яғни әрбір емделуші біз жазбаға ғана ие және әрбір жазба нақты тек бір емделушінің ақпаратын сақтайды. Өзіңіз келесі мысалдардан көріп тұрғандай, басқа да қысқартпаларды келтіруге болады.

5.9-сурет осы диаграмма класының түрін, Емделуші класының объектілері де, сондай-ақ басқа кластармен қатар байланысатынын дамытады. Мен бұл мысалда, сіз оқырманға бар байланыстардың түрін көрсететін ассоциацияларды атай алатындарыңызды баяндаймын.

Осы талдаудың деңгейінде, диаграммалар классы модельдің мағыналық деректері ретінде көрсетілген. Модельдің мағыналық деректері деректер қорын әзірлеу барысында қолданылады. Олар онымен байланысқан анықтауыштардың деректердің жүзін және осы жүздердің арасындағы байланысты көрсетеді. Мұндай модельдеудің тәсілі алғашында 1970 жылдардың ортасында Ченоммен (1976)

ұсынылған болатын; содан бері сол бастапқы үлгіде бірнеше нұсқалары (Кодд, 1979; Серп және Маклеод, 1981; Халл және патша, 1987) дайындалған.



5.9-сурет. MHC-PMS-тегі класстар және олардың өзара байланысы

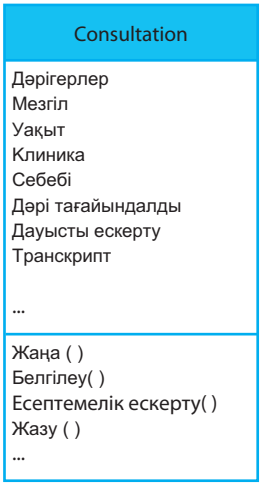
UML-ді мағыналық деректердің моделі ретінде қолдана аласыз. Сіз жеңілдетілген кластарды (олардың операциялары болмайды) мағыналық деректер моделі ретінде, атрибуттарды кластар арасындағы байланыстарды атағанымыздай объект класының қасиеті және қатынасы ретінде ұсына аласыз.

Кластар арасындағы ассоцияцияларды көрсеткен кезде, бұл кластарды қарапайым түрде елестету өте ыңғайлы. Оларды нақты анықтау үшін өз атрубуттарыңыз (объектінің мінездемесі) және операцияларыңыз (объектіден талап ете алатыныңыз) жайлы ақпаратты қосуға болады. Мысалы, емделуші, объектінің мекен-жай деген атрибуты болады, сонымен сіз өзіңізді мекен-жайды өзгерту, емделушінің бір мекен-жайдан екіншіге көшкен жағдайда шақырылатын операциясына қоса аласыз. UML-да сіз классты ұсынатын тіктөртбұрышты кеңейту арқылы атрибуттар мен операцияларды көрсетесіз. Бұл *5.10-суретінде* көрсетілгендей:

1. Объект класының аты жоғарғы бөлігінде орналасқан.
2. Класстың атрибуттары ортаңғы бөлікте емес. Бұл атрибуттың аттары мен мүмкін олардың түрлерін қамту керек.
3. Операциялар (Java және басқа объектіге-бағытталған бағдарламмалау тілінде аталатын амал-тәсілдер), тіктөртбұрыштың астыңғы бөлігіндегі кластың объектісімен байланысқан сияқты ережелерге сәйкес келеді.

5.10-суреті кеңес беру класының мүмкін болатын атрибуттары мен операцияларын көрсетеді. Мен бұл мысалда, дәрігерлердің дыбыстық жазбаларының кейін толыққанды кеңес беру ақпараттарының транскрипциялаудың жазбалары

деп пайымдаймын. Ем белгілеу үшін, дәрігер алдымен электронды дәріпарақ құру Prescribe тәсілін қолдану қажет.



5.10-сурет. Кеңес (консультация) класы

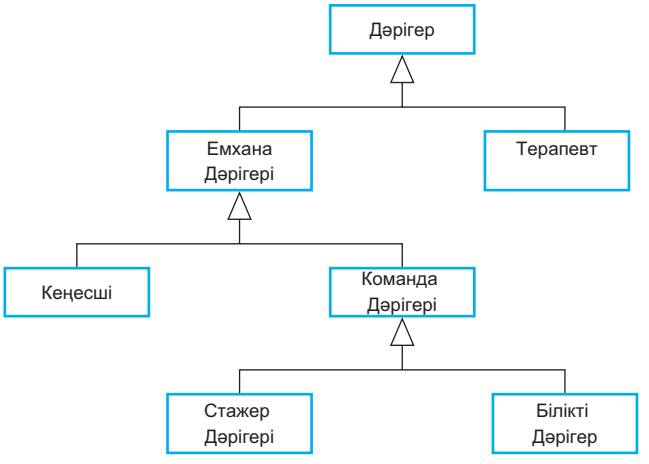
5.3.2. Жалпылау

Жалпылау біздің қиындықты басқару үшін қолданылатын күнделікті әдісіміз болып табылады. Біз әрдайым сезініп жүрген объектілердің мінездемелерін толығырақ білудің орнына, оны жалпы кластардың (жануарлар, автокөліктер, үйлер және т.б.) ішіне енгіземіз, сонымен қоса бұл кластардың мінезмелерін көрсетміз. Бұл бізге осы кластардың әртүрлі мүшелерінің кейбір жалпы мінездемелері (мысалы, ақтиіндер мен егеуқұйрықтар, кеміргіштер болып табылады) бар екен деген тұжырым жасауға септігін тигізеді. Біз барлық кластардың мүшелеріне (мысалы, барлық кеміргіштердің кеміруге арналған тістері бар) қолданылатын жалпы негіздеме жасай аламыз.

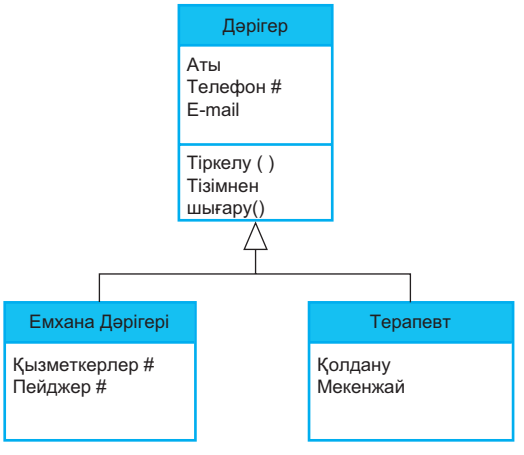
Жүйелерді модельдеуде, жалпылау мүмкіндігі барын көру үшін жүйедегі кластарды зерттеу әрдайым тиімді болып келеді. Бұл жалпы ақпараттардың барлығы бір жерде сақталатынын білдіреді. Бұл дизайнның жақсы тәжірбиесі, өйткені өзгеріске ұшыраған жағдайда, жүйедегі кластардың өзгеріске ұшырағанын көру үшін барлығын тескерудің қажеті жоқ. Java секілді объектіге-бағытталған тілдерде, жалпылау тілдерде орналасқан кластардың мұралану механизмдерін қолдана отырып жүзеге асады.

5.11-суреттегідей көрсетілгендей UML жалпылауды белгілеу үшін ассоциацияның спецификалық түріне ие. Жалпылау жоғара бағытталған жалпылама кластарын көрсететін нұсқаулық тәріздес белгіленген. Бұл жалпы тәжірбиенің және емхананың дәрігерлері барлығын, бір дәрігерлердің үш түрін енгізуге болатынын көрсетеді, Дәрігер – жаңадан медициналық мектепті тамамдап, байқаудан

өтіп жүргендер (Сынақ мерзімінен өтіп жүрген дәрігер), және бақылауды қажет етпейтін, кеңес беру тобының мүшесі ретіндегі дәрігерлер (Тіркелген дәрігер) және толық дайындығы бар және өздігінен шешім қабылдай алатын жоғарғы дәрежелі дәрігерлер.



5.11-сурет. Қарапайымдау иерархиясы

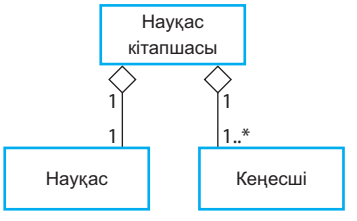


5.12-сурет. Қарапайымдау иерархиясының толық көрінісі

Жалпылауда, жоғарғы деңгейлі кластармен байланысқан атрибуттар мен операциялар, сондай-ақ тым төмен деңгейдегі кластармен де байланысады. Асылында, төменгі деңгейлі кластар және класс тармақтары атрибуттар мен операцияларды суперкластардан мұрагерлікке алады. Мысалы, барлық дәрігерлердің аты мен телефон нөмірлері бар, барлық дәрігерлердің жұмыскерлері мен бөлімдері бар, дегенмен, жалпы тәжірбие дәрігерлерінің мұндай атрибуттары жоқ, өйткені олар бір-бірінен тәуелсіз жұмыс істейді.»

5.3.3. Агрегациялау

Бүгінгі таңда объектілер бірнеше бөлшектерден тұрады. Мысалы, оқыту бағдарламасының өзі бір кітаптан, бірнеше PowerPoint-та жасалған слайдтардан, бақылау жұмыстардан және келесі оқуға байланысты ұсыныстардан тұруы мүмкін. Кейбір жүйелер осы бөліктерді көрсетуі қажет. UML бірнеше кластар арасындағы қатынасты бір жүйеге келістіруге мүмкіндік береді. Осы қатынасты модель жүйесінде көрсету үшін алмас түріндегі фигура қолданылады. *5.13-суретте* емделуші жазбасы, Емделуші мен Кеңестер кластары арасындағы қатынасы көрсетілген.



5.13-сурет. Агрегацияның ассоциациясы

5.4. Режимді модельдеу

Режимді модельдер жүйенің динамикалық өзгерістерін сипаттау үшін қолданылады. Олар жүйе ортаның тигізетін әсеріне қалай жауап беретінін және қалай өзгеретінін айқын көрсетеді. Осы әсерлер қатарына:

1. Ақпарат. Жүйе түрлі ақпараттарды өңдеу арқылы өзгеретіні.
2. Оқиғалар. Кейбір жүйелер осы оқиғаларға түрлі мәнде жауап беретіні жатады.

Деректер ағынының диаграммалары

Деректер ағынының диаграммасы (DFDs) әрбір берілген деректің жеке функцияны немесе үдерісті білдіретін функционалды тұжырымдаманы көрсетуші жүйе үлгілері болып табылады. Деректер ағынының диаграммасы (DFDs) деректердің бірізді өңдеу қадамдары арқылы берілу жолдарын көрсету үшін пайдаланылады. Мысалы, тұтынушының деректер қорындағы көшірмеленген жазбалардың сүзгіленуі өңдеу қадамы болуы мүмкін. Дерек келесі қадамға өту алдындағы әрбір қадамда түрленіп отырады. Бұл өңдеу қадамдары немесе түрленулер деректер ағынының диаграммасы (DFDs) бағдарламалық жасақтама дизайнының құжатына қолданылатын бағдарламалық жасақтама үдерістерін білдіреді..

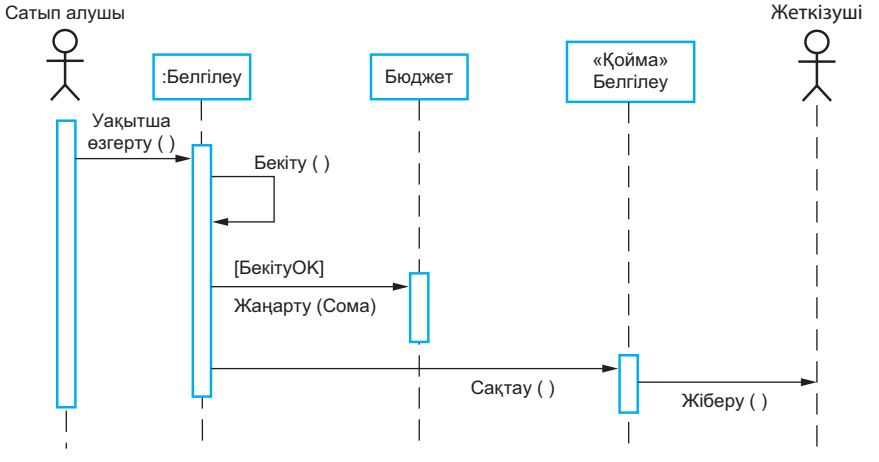
http://www.SoftwareEngineering-9.com/Web/DFDs

Осы оқиғалар ақпаратпен тікелей байланыста болады. Көптеген бизнес үдерістер ақпаратпен тікелей жұмыс жасайды. Олар белгілі бір үдерістің әсерінен пайда болатын ақпарат арқылы басқарылады. Осы үдеріс бірнеше ақпаратпен жұмыс жасай отырып, белгілі бір өнім алады. Мысалы, телефон қызметінің есеп құрастырушылары бүкіл тұтынушылардың жасаған қоңырауларына байланысты есепшотын шығарып береді. Бірақ нақты уақыттағы жүйелер ақпараттан гөрі үдерістерге көп көңіл бөледі. Мысалға айтатын болсақ, стационарлық телефон арқылы қоңырау шалу үшін көптеген үдерістерден өткеннен кейін ғана ақпарат жіберіледі.

5.4.1. Ақпарат арқылы басқарылатын модельдеу

Ақпарат арқылы басқарылатын модельдер бірнеше үдерістерден құрылып, белгілі бір кірістен өнім алуға бағытталған. Олар жүйе талаптарын анализден өткізу кезінде айтарлықтай пайдалы болып келеді, өйткені бүкіл үдеріс бастапқы қадамнан соңғы қадамына дейін қарастырылады. Осы модель бүкіл үдерісті қадам-қадамымен көрсету арқылы соңғы өнімнің қалай алынатындығын көрсетеді.

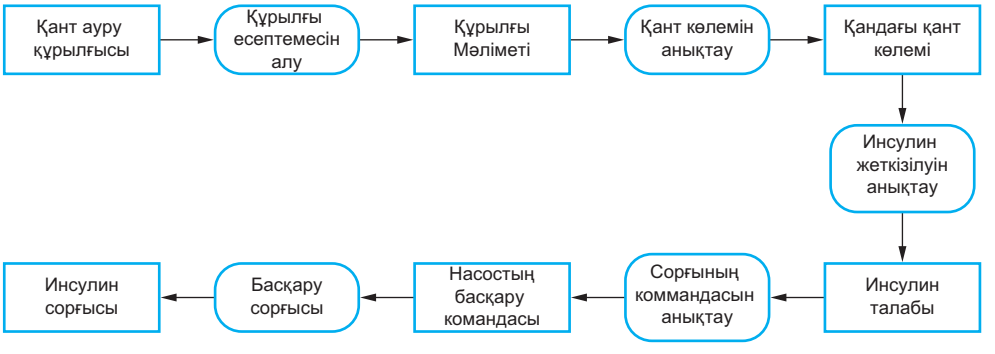
Ақпарат арқылы басқарылатын модельдер графикалық модельдеу жүйелерінің ең алғашқы түрлеріне жатады. 1970 жылы Де Марко ұсынған құрылымды анализ (DeMarco, 1978) барысында ақпарат ағынының диаграммалары (data-flow diagrams) атты жаңа модель енгізген болатын. Бұл модельдер анализ жасайтындар мен дизайнерлер үшін өте тиімді, өйткені бұл модель ақпарат пен үдерістің қатынасын айқын көрсете алады. Ақпарат ағыны модельдері түсінуге өте жеңіл әрі түсінікті, сондықтан тапсырыс берушілер жүйе тестілеуі кезінде өтетін үдерістердің қадамын нақты түсініп біліп алуға мүмкіндіктері бар.



5.14-сурет. Инсулин сорғышының операцияларының әрекеттер үлгісі

UML ақпаратты үдерістер барысында өңдеу үшін құрастырылғанмен, ақпарат ағыны диаграммаларын қолданбайды. Бұған негізгі себеп, ол – ақпарат ағыны диаграммалары жүйенің объектілерінен гөрі жүйе функцияларына көбірек

негізделген. Бірақ ақпарат арқылы басқарылатын жүйелер бизнес ортасында танымал болғандықтан, UML 2.0 ақпарат ағынының диаграммаларына жақын белсенділік диаграммаларын енгізген болатын. Мысалы, *5.14-суретте* инсулин сорғы жүйесінің жұмыс принципінің үдерістері көрсетілген. Бұл диаграммада өтіп жатқан үдерістердің реттілігі (әрекет ретінде) мен осы үдерістердегі ақпарат ағынын (объектілер ретінде) көрсетеді.



5.15-сурет. Тапсырыстарды орындау үдерісі

UML реттілік диаграммалары өтіп жатқан үдерістерді көрсетуге арналған тағы бір құралы ретінде сипатталады. Осы диаграммалар модельдегі үдерістер мен объектілер арасындағы қарым-қатынасты көрсетпей-ақ, осы бөлшектер арасында өтіп жатқан ақпаратты солдан оңға қарай бағыттайды да, осы үдерістердің реттілік қасиетін көрсетіп шығады. *5.15-суретте* тұтынушы тапсырыс жасаған кезден бастап, осы тапсырыстың тапсырыс берушінің қолына жеткізілуіне дейінгі бүкіл үдерістердің реттілігі көрсетілген. Реттілік модельдер жүйенің объектілерін бөліп шығарса, ақпарат арқылы басқарылатын диаграммалар жүйе әрекеттеріне негізделеді. Тапсырыстың орындалу үдерісі баламалы ақпарат арқылы басқарылатын диаграмма арқылы кітаптың веб-сайтында берілген.

5.4.2. Оқиға арқылы басқарылатын модельдеу

Оқиға арқылы басқарылатын модельдер жүйенің ішкі әрі сыртқы әсерлерге қалай жауап қайтаратындығын көрсетеді. Осы модельдердің құрастырылу принципі әрбір жүйенің шектеулі жағдайлары болады және пайда болатын түрлі оқиғалар әсерінен жүйе бір жағдайдан екінші жағдайға көшіп отыратындығына тікелей негізделген. Мысалы, клапан жұмысын басқаратын жүйе оператор командасына сай «Клапан ашық» жағдайынан «Клапан жабық» жағдайына ауыса алады. Осындай модельдеу әдісі нақты уақыта жұмыс істейтін жүйелер үшін тиімді. Оқиға арқылы басқарылатын модельдерді Вард пен Меллор (1985) және Харел (1987, 1988) нақты уақыттағы дизайн әдістері арқылы түсіндіріп енгізген болатын.

Харел ұсынған жағдай графиктеріне (Harel, 1987, 1988) негізделген UML жағдай диаграммаларын қолдана отырып, оқиға арқылы басқарылатын модельдеу әдісін қолдайды. Жағдай диаграммалары жүйенің статусын көрсете отырып, бір

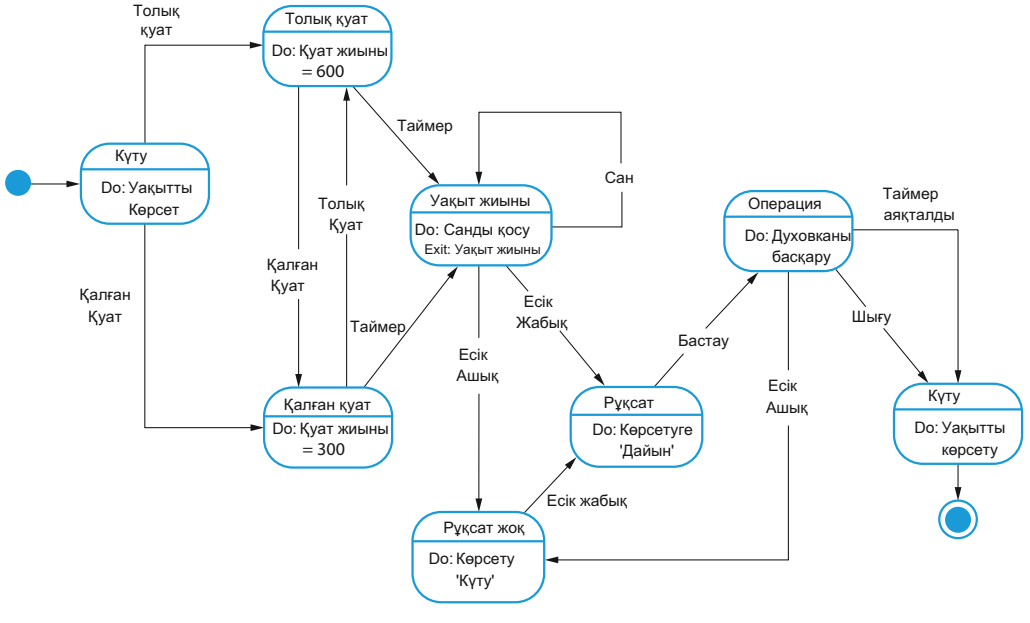
деңгейден екіншіге көшу үдерісін көрсетеді. Олар ақпараттың қалай өтетіндігін қарастырмайды, бірақ түрлі үдерістер кезінде өтетін есептемелерге көңіл бөледі екен.

Мен оқиға арқылы басқарылатын модельдеу әдісін көрсету үшін шағын толқынды пештің бағдарламалық қамсыздандыруының жұмыс жасау негізін ұсындым. Кәдімгі толқынды пештерінің жұмыс жасау принциптері одан әрі қиын болып келеді, бірақ осы қарапайым жүйені түсіну жеңіл. Қарапайым толқынды пеші қуат ауыстырғыш, уақытты таңдауға арналған нөмірленген пернетақтадан, қосып-сөндіретін бөліктен және дисплейден тұрады.

Осы пешпен жұмыс жасаған кезде:

1. Қуатты таңдау (жарты немесе бүкіл қуатпен жұмыс жасайтындығы).
2. Тамақты даярлау уақытын нөмірленген пернетақтасы арқылы таңдау.
3. Бастау пернесін басып, белгіленген уақыт күтеміз деген әрекеттер орындалады деп шештім.

Қауіпсіздік үшін толқынды пеш есігі жұмыс жасаған кезде жабық болуы қажет, тек пеш жұмысы аяқталып, белгілі бір дыбыс болғаннан кейін ғана есікті ашуға болады. Бұл пеш кішкентай әрі қарапайым дисплейді қателіктер мен хабарламаларды шығару үшін қолданады.



5.16-сурет. Микротолқынды пештің тұрақты кескіні

UML жағдай диаграммалары белгілі жағдайды дөңгелектелген фигура ретінде көрсетеді. Олар шағын түсініктемеден (қандай әрекет жасайтындығы туралы) тұрады. Белгіленген меңзер осы жүйенің бір жағдайдан екіншіге өтуін белгілейді. Жүйе үдерісінің басын мен соңын боялған дөңгелек арқылы көрсетуге болады.

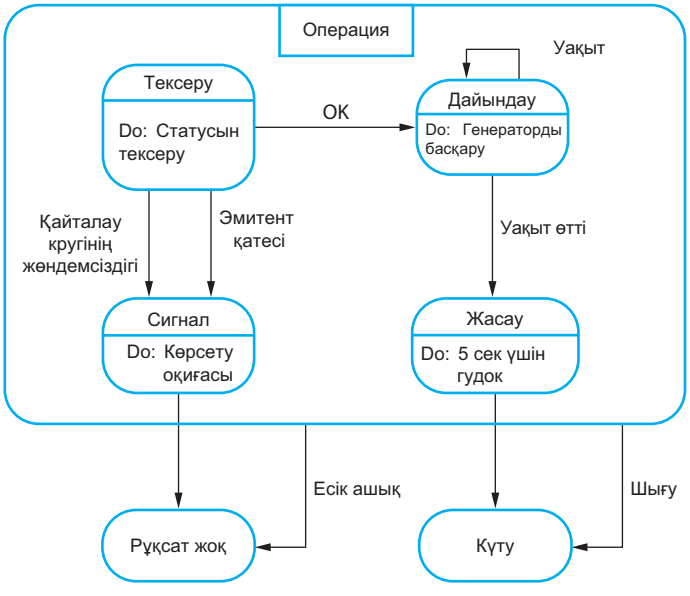
5.16-суретте жүйе күту режимінен басталып, жартылай қуат пен толық қуат баспаларына көшетіндігі сипатталған. Пайдаланушылар өз ойын өзгертіп, басқа бір баспаны басса да болады. Егер пештің жұмыс жасау қуаты таңдалып, уақыт енгізіліп, пеш есігі жабық болса, онда «Бастау» баспасы іске қосылуы мүмкін. Осы баспа басылғаннан кейін пеш өз жұмысын белгіленген уақытта атқарады. Бұл үдеріс жүйенің соңғы статусы болып қалады да, жүйе қайтадан күту режиміне көшеді.

Белгілі жағдай кезіндегі пайда болатын оқиғаларды UML арқылы белгілеуге болады. Бағдарламалық қамсыздандыруды сипаттау кезінде түрлі оқиғаларды жүйеге әсерін және жағдайларын енгізу қажет. *5.17-суретте* жүйеде өтіп жататын әрбір жағдайды және бір кезеңнен екіншіге көшу үрдісі айқын сипатталады.

Күй	Баяндама
Күту	Пеш кіретін ақпаратты күтеді. Дисплей қазіргі уақытты көрсетеді
Жартылай қуат	Пеш 300 ватқа қойылды. Дисплей жартылай қуатты көрсетеді.
Толық қуат	Пеш 600 ватқа қойылды. Дисплей толық қуатты көрсетеді.
Уақытты қою	Пісіру уақыты адамның енгізген санына байланысты қойылады. Дисплей адамның енгізген уақытын көрсетеді.
Сөндірілген	Пеш операциялары қауіпсіздік мақсатында сөндірілген. Ішіндегі шам жанады. Дисплей пештің дайын еместігі туралы ақпаратты көрсетеді.
Қосылған	Пеш операциялары қосылған. Ішіндегі шам сөндіріледі. Дисплей пештің пісіруге дайын болғаны туралы ақпаратты көрсетеді.
Қызмет атқаруда	Пеш қызмет атқаруда. Ішіндегі шам жанады. Дисплей уақыттың өзгерісін көрсетеді. Пісіріп біткеннен кейін дыбыс ретіндегі 5 секундтық хабарлама беріледі. Пеш шамы қосылған. Дыбыс хабарламасы беріліп тұрғанда дисплей пісіру.
Ынта	**Баяндама**
Жартылай қуат	Пайдаланушы жартылай қуат түймесін басты.
Толық қуат	Пайдаланушы толық қуат түймесін басты.
Таймер	Пайдаланушы таймер түймелерінің біреуін басты.
Сан	Пайдаланушы сан түймелерінің біреуін басты.
Есік ашық	Пештің есігі жабылмады.
Есік жабық	Пештің есігі жабылды.
Старт	Пайдаланушы старт түймесін басты.
Түзету	Пайдаланушы түзету түймесін басты.

5.17-сурет. Шағын толқынды пештің күйлері және ынталары

Белгілі бір әрекеттер арқылы басқарылатын модельдеудің ең басты мәселесі, ол – жүйеде пайда болатын жағдайлар санының көптігінде. Кешенді жүйелерді құрастыру кезінде көптеген мәселелерді модель барысында жасырады. Осы принцип бірнеше оқиғалардан тұратын кешенді жағдай көмегімен жүзеге асырылады. Бұл кешенді жағдай жоғарғы деңгейлік модельден бірнеше диаграммаларға бөлініп құрастырылады. *5.15-суретте* Жұмыс жағдайы диаграммасында көрсетілген осы принциптің жұмысын көре аласыз. Кешенді жағдайдың мысалы *5.18-суретте* келтірілген.



5.18-сурет. Микротолқынды пештің операциялары

Жұмыс жағдайы бірнеше оқиғалардан құрастырылады. Бұл белгілі бір жағдай басталған мезетте өтіп жатқан үдеріс тексерістен өтіп, кейбір қателіктер пайда болатын болса, қауіп хабарламасы көрсетіліп, операция тоқтатылады. Толқынды пештің жұмыс жасау принципі осыған да негізделген, белгіленген уақыттан кейін жұмыс тоқталады. Егер пештің есігі осы операция орындалып жатқан кезде ашылып қалса, онда пеш жұмысын тоқтатып жібереді (*5.15-сурет*).

5.5. Модель арқылы басқарылатын әзірлеу

Модель арқылы басқарылатын әзірлеу (MDE) кезінде жүйені құрастырудың өнімі ретінде бағдарламалар емес модельдер алынады (Kent, 2002; Schmidt, 2006). Бағдарламалар металл бұйымдар мен бағдарламалық қамсыздандыру арқылы орындалып, автоматты түрде модельдер көмегімен жүзеге асырылады. MDE әдісін жақтаушылар бағдарламаны жасақтаудың абстракциясы жоғарғы деңгейге

жеткеннен кейін бағдарлама жасақтаушылар бағдарламалау тілімен тығыз жұмыс жасауға деген қажеттілігі жоғалады.

Модель арқылы басқарылатын әзірлеу 2001 жылы Object Management Group (OMG) ұсынған модель арқылы басқарылатын архитектура әдісімен сабақтас. Осы екі модельдеу әдістері бір-біріне жақын болып қарастырылады. Бірақ, менің ойымша, модель арқылы басқарылатын әзірлеу модель арқылы басқарылатын архитектура әдісіне қарағанда кешенді көрінеді. Модель арқылы басқарылатын архитектура тек жобаны жүзеге асыруды және дизайн кезеңдерін қарастырса, модель арқылы басқарылатын әзірлеу әдісі осы жобаның бүкіл кезеңдерін толығымен қарастырып шығады екен. Сондықтан модель арқылы басқарылатын талаптарды құрастыру және тестілеу кезеңдері осы модель арқылы басқарылатын әзірлеу әдісі арқылы көрсетіледі.

Модель арқылы басқарылатын архитектура 2001 жылдан бері қолданыста болғанымен, модель арқылы басқарылатын құрастыру үдерісі үшін осы әдістің әсері әлі түсініксіз. Модель арқылы басқарылатын әзірлеу әдісі үшін және қарсы келтірілген аргументтер:

1. *Модель арқылы басқарылатын әзірлеу үшін.* Осы әдіс көмегімен бағдарламаны жасақтаушылар жүйені жоғары абстракциялық деңгейде қарастыра отырып, жүйені құрастыру кезіндегі ұсақ бөліктерге көңіл бөлмеуге мүмкіншіліктері бар. Осыған байланысты жүйені әзірлеуге кететін уақыт азаяды, пайда болатын қателіктер саны төмендейді, платформадан тәуелсіз әрі көп мәртелік жүйені құрастыру мүмкіндігі пайда болады. Мықты бағдарламалар көмегімен бір модель арқылы жасалған жүйені бірнеше платформада жүзеге асыруға болады.Сондықтан жүйені жаңа платформаға бейімдеу үшін аударушы бағдарламасын қолданған жөн. Бұл бағдарламаны қолданған кезде, бүкіл платформадан тәуелсіз жүйелер басқа платформада жұмыс жасауға тезірек бейімделеді.
2. *Модель арқылы басқарылатын әзірлеуге қарсы.* Осы тараудың басында айтып кеткенімдей, модельдер жүйе дизайнын талқылау кезінде қолданған қолайлы. Бірақ, осы модельдер арқасында пайда болатын абстракция жүйені құрастыру үшін пайдалы әрі жарамды бола бермейді. Сіз жүйенің дұрыс емес дизайн сипаттамасын құрастырып алып, оның негізінде жүйені құрастыру кезеңіне көшуіңіз мүмкін. Платформа тәуелсіздігі тек кешенді жобалар үшін қолданған дұрыс. Бірақ, бұл жүйелер жиынтығы үшін құрастыру кезеңі сондай үлкен мәселе ретінде қарастырылмайды, өйткені жобаның ең басында талаптарды дұрыс қойып алу, жүйе қауіпсіздігін қарастыру, жүйені енгізу және тестілеу үдерістері одан әрі маңызды болып саналады.

IBM және Siemens сияқты кешенді компаниялар өз жүйелерін құрастыру кезінде модель арқылы басқарылатын әзірлеу әдісін қолданылуы сәтті өткен эксперименттері OMG вебсайтында сипатталған болатын (www.omg.org/mda/products_success.htm). Кешенді әрі ұзақ әуе қозғалысын басқару бағдарламасын

құрастыру модель арқылы басқарылатын әзірлеу әдісі кең қолданылған болатын. Бірақ осы мақаланы жазған уақытта модель арқылы басқарылатын әдістер бағдарламалық қамсыздандыруда кең қолданбаған. *12-тарауда* модель арқылы басқарылатын әдістердің маңыздылығы жақсы сипатталған. Бірақ осы әдістер баға және қателіктер тарапынан қаншалықты пайдалы екені белгісіз.

5.5.1. Модель арқылы басқарылатын архитектура

Модель арқылы басқарылатын архитектура (Kleppe, et al., 2003; Mellor et al., 2004; Stahl and Voelter, 2006) – жүйеге сипаттама беру үшін UML модельдеріне негізделген модельдеу принципі. Бұл принцип негізінде түрлі абстрактілі деңгейдегі модельдер құрастырылады. Жоғары деңгейлік әрі платформаға тәуелсіз модель көмегімен жұмыс жасайтын бағдарламаны жасап шығаруға болады.

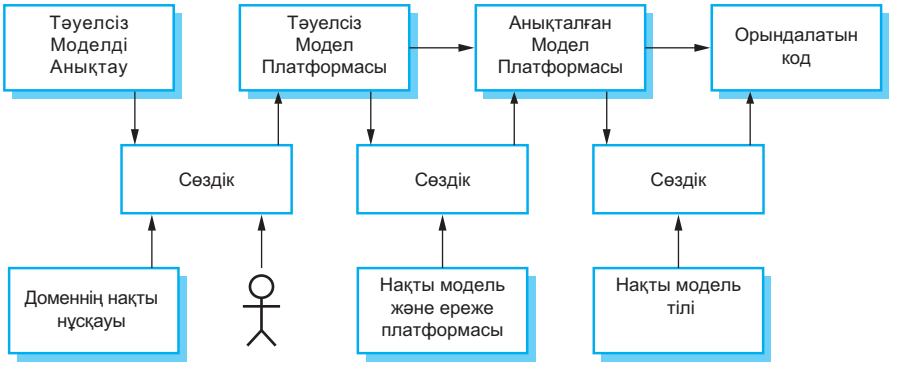
Модель арқылы басқарылатын архитектура әдісі келесі үш абстрактілі модель жүйесін ұсынады:

1. Тәуелсіз есептеулер модель (CIM) жүйенің басты бөліктерін сипаттайды. Осы модельді, негізінде, басты бөліктер моделі деп атайды. Сіз бірнеше тәуелсіз есептеулердің моделін жасап жүйені түрлі жақтан сипаттайтын көрініс ала аласыз. Мысалы, қауіпсіздікке байланысты жасалған тәуелсіз есептеулер емделуші туралы мәліметтің сақталуы туралы абстрактілі жүйесін сипаттап шығады.
2. Тәуелсіз платформа моделі (PIM) жүйенің жұмыс жасау принципін сипаттап береді. Бұл модель UML диаграммалары көмегімен жүйенің статикалық моделін және осы модельдің ішкі әрі сыртқы оқиғаларға қалай жауап қайтаратындығын айқын көрсетеді.
3. Әрбір қолданбадағы платформа тәуелсіз платформалық модельге бейімделген нақты платформа моделі (PSM) арқылы жүзеге асырылады. Бұл модель бірнеше кезеңдерден тұрады әрі әрбір кезеңде жаңа платформалық қосымшалар енгізіледі. Біріңші деңгейде модель дерекқордан тәуелсіз болады. Ақпарат жиынтығы құрастырылғаннан кейін, модель осы дерекқорға тәуелді болып қалады.

Айтып кеткенімдей, модельдер арасындағы өзгерістерді жүзеге асыру үшін бағдарламалық қолданбалар қолданылады. Соңғы өзгерістер арасындағы автоматтандырылған көшу үдерісі *5.19-суретте* келтірілген. Осы өзгерістерді жүзеге асыру үшін жұмыс жасайтын код әзірлеу қажет.

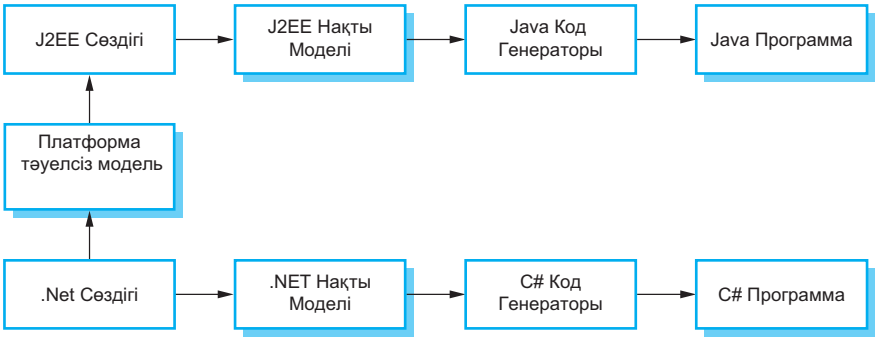
Осы мақала жазылған мезетте автоматтандырылған тәуелсіз есептеу моделін тәуелсіз платформа моделіне көшіру үдерісі әлі түгелімен аяқталмаған болатын. Сондықтан болашақта толығымен жасалып біткен автоматтандырылған көшіру бағдарламалары қолданысқа түсуі үлкен күмән. *5.19-суретте* келтірілген диаграмма жүзінде осы үдеріске деген адамдық әсерді айқын байқауға болады. Тәуелсіз есептеулер моделі бір-бірімен тікелей байланысты, сондықтан көшіру үдерісі

кезінде осы байланыс қарастырылады. Мысалы, тәуелсіз есептеулер моделінің қауіпсіздікке байланысты тұжырымы тәуелсіз есептеулер моделінің емхана жұмысшыларына қолданылуы мүмкін. Бір тәуелсіз есептеу моделінен екіншіге көшіру әдістерін Меллор мен Балсер (2002) «көпір» ретінде қарастырады.



5.19-сурет. MDA тасымалдануы

Тәуелсіз платформа моделін нақты платформа моделіне көшіру әбден нығайтылған үдеріс және бірнеше комерциялық жабдықтар тәуелсіз платформа моделін Java және J2EE платформаларға сай көшіру үшін қолданады. Тәуелсіз платформа моделін нақты платформа моделіне көшіру үдерісі ауқымды платформа кітапханасына және платформалық ережелерге байланысты. Әрбір тәуелсіз платформа моделінің жүйесі үшін бірнеше нақты платформа модельдері бар. Егер бағдарламалық жүйе түрлі платформаларда жұмыс жасай алатын болса (мысалы, J2EE және .NET), онда тәуелсіз платформалық модельді құрастырған жөн. Әрбір платформа үшін тәуелсіз әрі нақты платформалық модель құрастырылып шығады. Бұл үдеріс *5.20-суретте* көрсетілген.



5.20-сурет. Көп платформалы сипаттамалардың үлгілері

Бірақ, модель арқылы басқарылатын архитектура бағдарламалық жабдықтар басқа платформаларға көшіретін бағдарламаларды қолданғанымен тәуелсіз плат-

форма моделін нақты платформа моделіне көшіру үдерісі толығымен өтпеуі мүмкін. Көбінесе, жүйенің орындау ортасы дәстүрлі платформа орындалудан (мысалы, J2EE және NET) күрделірек. Бұл басқа бағдарламалық жүйелерден, осы бағдарламаларға байланысты ақпарат дерекқорларынан және пайдаланушы интерфейсінің кітапханасынан тұрады. Осының бәрі әр компанияларда өзгеше болып келеді, бәріне бір стандартты қолдау жоқ. Сондықтан модель арқылы басқарылатын архитектура әдісі енгізілген кезде көшіру бағдарламалары белгілі бір компанияның үйреншікті қасиеттерін есепке алуы қажет. Кейбір жағдайларда (мысалы, пайдаланушы интерфейстері үшін) түгелімен автоматтандырылған тәуелсіз платформа моделінен нақты платформа моделіне көшіру мүмкін емес.

Икемді әдістер мен модель арқылы басқарылатын архитектура әдістері арасындағы қатынас күрделі. Ауқымды модельдеу ұғымы икемді манифестінің басты идеяларына қарсы шығады, менің ойымша, бірнеше икемді құрастырушылар модель арқылы басқарылатын құрастыру үрдістерімен жұмыс жасағанды дұрысырақ санайды. Модель арқылы басқарылатын архитектураны құрастырушылар осы модельдің кезеңдік құрастыруды қолдайтындықтан икемді әдістермен бірге қолданыла алады деп айтады екен (Mellor, et al., 2004). Егер көшіру үдерісі толығымен автоматтандырылып әрі тәуелсіз платформа моделі арқылы жасалған бағдарлама дұрыс жұмыс атқаратын болса, онда модель арқылы басқарылатын архитектура әдісі икемді құрастыру техникасымен бірге қолданылуы мүмкін. Бірақ мен білетінімдей модель арқылы басқарылатын архитектура әдісі кемімелдік анализ бен тестілеу арқылы басқарылатын дамыту үрдісін қолданбайды.

5.5.2. Атқарылатын UML

Модель арқылы басқарылатын құрастыру негізінен модельден код жүзіне автоматтандырылған көшу үдерісін жүргізу мүмкін болуы керек. Осы үрдісті жүзеге асыру үшін мағынасы дұрыс графикалық модельдерді құрастырылған жөн. Осыған қоса осы модельдерге жаңа жүйелік операцияларды енгізу мүмкіндігі болуы қажет. Бұл үрдіс үшін Атқарылатын UML немесе xUML деп аталатын UML2-ның бір бөлігін қолданады (Mellor and Balcer, 2002). xUML-ді толықтай сипаттап бере алмаймын, сондықтан осы кітапта бұл модельдеудің басты қасиеттерін сипаттап бердім.

UML – ол бағдарламалау тілі емес, ол – бағдарламалық жүйенің дизайның құжат ретінде көрсететін тіл болып табылады. UML дизайнерлері көбінесе оның мағыналылығына емес бейнелілігіне көп көңіл бөледі. Олар қолдану нұсқасы деп аталатын жаңа ұғым енгізген болатын, бірақ бұл нұсқа жүйенің орындалу барысын дұрыс көрсете алмайды. Атқарылатын UML модельдеуін дұрыс құрастырып шығу үшін модель түрлерінің санын үшеуге дейін қысқартқан:

1. Басты модельдер жүйедегі нағыз мәселелерді шешуге бағытталған. Бұл объектілер, аттрибуттар және байланыстардан тұратын UML класс диаграммалары арқылы анықталады.

2. Класс модельдерінде әрбір класс өз аттрибуттары мен операциялары арқылы құрастырылады.
3. Жағдай модельдеріндегі жағдай диаграммалары әрбір кластың жұмыс жасау принципін сипаттап шығады.

Жүйенің динамикалық тәртібі объект тілін қолдана отырып немесе UML тілін қолдана отырып суреттеуге болады. Әрекет тілі – объектілермен жұмыс жасай отырып, оның аттрибуттарына арналған әрекеттерін анықтайтын жоғары деңгейдегі тіл.

НЕГІЗГІ ТҰЖЫРЫМДАР

- Модель – бұл бірнеше жүйелік детальдарды ескермейтін жүйенің абстрактілі көрінісі. Жүйенің мәнмәтінің, байланысын, құрылымын және әрекет ету принципін көрсету үшін қосымша жүйелік модельдер жасалады.
- Мәнмәтін моделі белгілі бір жүйенің ортада қалайша бейімделгені және басқа жүйелер мен операциялармен қалай жұмыс жасайтындығын көрсетеді. Бұл құрастырылып жатқан жүйе шекараларын анықтауға көмектеседі.
- Пайдалану нұсқасы мен ретті диаграммалар жүйе мен пайдаланушы арасындағы қарым-қатынасты сипаттау үшін қолданады. Пайдалану нұсқасы сыртқы пайдаланушылар мен жүйе арасындағы қатынасты анықтайды, ал ретті диаграммалар осы қатынасты объектілер арасындағы қатынас арқылы көрсетеді.
- Құрылым модельдері жүйе архитектурасы мен ұйымдастыру жолын көрсетіп шығады. Класс диаграммалары статикалық жүйе құрылымын және оның басқа жүйелермен қатынасын көрсетеді.
- Тәртіп моделі арқылы жүйенің динамикалық тәртібін суреттеуге болады. Бұл модель жүйе ақпараты арқылы және пайда болатын операцияларды көрсетеді.
- Белсенділік диаграммалары ақпаратты өңдеу үдерісін көрсетеді, әрбір операция өңдеу үдерісінің бір қадамын көрсетеді.
- Жағдай диаграммасы сыртқы және ішкі әрекеттерге жүйенің жауап беруін сипаттайды.
- Модель арқылы басқарылатын әзірлеу әдісі жүйені сипаттайтын бірнеше модельдерден тұрады әрі осы модель арқылы жұмыс жасайтын код құрастырылып шығады.

ҚОСЫМША ӘДЕБИЕТТЕР

Requirements Analysis and System Design. This book focuses on information systems analysis and discusses how different UML models can be used in the analysis process. (L. Maciaszek, Addison-Wesley, 2001.)

MDA Distilled: Principles of Model-driven Architecture .This is a concise and accessible introduction to the MDA method. It is written by enthusiasts so the book says very

little about possible problems with this approach. (S. J. Mellor, K. Scott and D. Weise, Addison-Wesley, 2004.)

Using UML: Software Engineering with Objects and Components, 2nd ed. A short, readable introduction to the use of the UML in system specification and design. This book is excellent for learning and understanding the UML, although it is not a full description of the notation. (P. Stevens with R. Pooley, Addison-Wesley, 2006.)

ЖАТТЫҒУЛАР

5.1. Неге жүйе мәнмәтінің модельдеу маңызды болып саналады? Егер жүйе құрастырушылар жүйені түгелімен түсінбей қалса, пайда болатын мәселелерге екі мысал келтіріңіз.

5.2. Құрастырылып алынған жүйе моделін қалайша қолдануға болады? Осындай жүйе үшін модельдің толық болмағаны неге маңызды емес екендігін түсіндіріңіз. Мұндай әдіс жаңадан жасалып жатқан жүйе үшін қолайлы ма?

5.3. Сіз үйлену тойы, туған күн, оқуды аяқтау сияқты үлкен мейрамдарды өткізетін жүйені құрастыру деген тапсырма алдыңыз. Белсенділік диаграммаларын қолдана отырып, әрбір тойдың өту үдерісін мен қажетті операцияларды сипаттап шығыңыз.

5.4. MHC-PMS жүйесі үшін емделушілерге қарайтын дәрігер мен емдеу үдерістері арасындағы қатынасты көрсететін орындау нұсқаларын құрастырып шығыңыз.

5.5. Студенттің жаңа курсқа жазылуын реттік диаграмма арқылы көрсетіңіз. Курсқа жазылатын студенттер саны шектелген, сондықтан жүйе бос орынға тексеріс жасау қажет. Студент белгілі курсты онлайн курстар тізбесі арқылы таңдай алады.

5.6. Электрондық пошта жүйесіндегі хаттар мен жәшіктердің қалай орналасқанын жақсылап қарап шығыңыз. Осы жүйені құрастыру үшін объектілі кластарын құрастырып шығыңыз.

5.7.Банкомат арқылы орындаған операцияларыңызды еске алып, осы үдерістерді белсенділік диаграмма жүзінде көрсетіп шығыңыз.

5.8. Жоғарыда айтылып кеткен жүйенің үдерісін неге белсенділік пен реттілік диаграммалары арқылы көрсеткенді жөн көресіз.

5.9. Әрбір төмендегі жүйеге жағдай диаграммасын сызып шығыңыз:

Автоматтандырылған кір жуатын машинасы үшін түрлі киімге арналған жүйе;

DVD плеерге арналған жүйе;

Жаңадан келген хабарламаларды жазып алатын және келген хабарлама санын дисплейде көрсететін телефон жүйесі. Жүйе пайдаланушыға кез келген жерден телефон шалуға, кез келген нөмір ретін енгізуге және келген хабарламаларды орындауға мүмкіндік беруі қажет.

5.10. Сіз жүйені құрастырушысыз және Сіздің командаңыз жүйені модель арқылы басқарылатын әдісті қолдануға ұсыныс берген. Осы әдісті қолданар алдында қандай басты факторларды қараудан өткізуіңіз қажет?

ПАЙДАЛАНЫЛҒАН ӘДЕБИЕТТЕР

Ambler, S. W. and Jeffries, R. (2002). *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process*. New York: John Wiley & Sons.

Booch, G., Rumbaugh, J. and Jacobson, I. (2005). *The Unified Modeling Language User Guide, 2nd ed*. Boston: Addison-Wesley.

Chen, P. (1976). 'The entity relationship model—Towards a unified view of data'. *ACM Trans. on Database Systems*, **1** (1), 9–36.

Codd, E. F. (1979). 'Extending the database relational model to capture more meaning'. *ACM Trans. on Database Systems*, **4** (4), 397–434.

DeMarco, T. (1978). *Structured Analysis and System Specification*. New York: Yourdon Press.

Erickson, J. and Siau, K. (2007). 'Theoretical and practical complexity of modeling methods'. *Comm. ACM*, **50** (8), 46–51.

Hammer, M. and McLeod, D. (1981). 'Database descriptions with SDM: A semantic database model'. *ACM Trans. on Database Sys.*, **6** (3), 351–86.

Harel, D. (1987). 'Statecharts: A visual formalism for complex systems'. *Sci. Comput. Programming*, **8** (3), 231–74.

Harel, D. (1988). 'On visual formalisms'. *Comm. ACM*, **31** (5), 514–30.

Hull, R. and King, R. (1987). 'Semantic database modeling: Survey, applications and research issues'. *ACM Computing Surveys,* **19** (3), 201–60.

Jacobson, I., Christerson, M., Jonsson, P. and Overgaard, G. (1993). *Object-Oriented Software Engineering*. Wokingham.: Addison-Wesley.

Kent, S. (2002). 'Model-driven engineering'. Proc. 3rd Int. Conf. on Integrated Formal Methods, 286–98.

Kleppe, A., Warmer, J. and Bast, W. (2003). *MDA Explained: The Model Driven Architecture—Practice and Promise*. Boston: Addison-Wesley.

Kruchten, P. (1995). 'The 4 + 1 view model of architecture'. *IEEE Software*, **11** (6), 42–50.
Mellor, S. J. and Balcer, M. J. (2002). *Executable UML*. Boston: Addison-Wesley.

Mellor, S. J., Scott, K. and Weise, D. (2004). *MDA Distilled: Principles of Model-driven Architecture*. Boston: Addison-Wesley.

Rumbaugh, J., Jacobson, I. and Booch, G. (1999). *The Unified Modeling Language Reference Manual*. Reading, Mass.: Addison-Wesley.

Rumbaugh, J., Jacobson, I. and Booch, G. (2004). *The Unified Modeling Language Reference Manual, 2nd ed*. Boston: Addison-Wesley.

Schmidt, D. C. (2006). 'Model-Driven Engineering'. *IEEE Computer,* **39** (2), 25 – 31.

Stahl, T. and Voelter, M. (2006). *Model-Driven Software Development: Technology, Engineering, Management*. New York: John Wiley & Sons.

Ward, P. and Mellor, S. (1985). *Structured Development for Real-time Systems*. Englewood Cliffs, NJ: Prentice Hall.

6.

Сәулеттік әрлем

Мақсаттары

Бұл тараудың мақсаты – бағдарламаның сәулеті мен сәулеттік әрлендірменің тұжырымдарымен таныстыру. Тарауды оқу барысында сіз:

- сәулеттік әрлендірменің маңыздылығын түсінесіз;
- сәулеттік әрлеу жасау барысындағы сәулеттік жүйе жайында жасалу керек шешімдермен танысып, маңызын түсінесіз;
- Сәулеттік қалыптардың идеясымен, сәулеттік жүйені ұйымдастырудың қолайлы жолдарымен танысасыз әрі оларды жүйенің әрлендірмесінде қолдануды үйренесіз.
- әртүрлі қосымша бағдарламалар жүйесінде қолданылатын сәулеттік жүйелерді білетін боласыз, олардың қатарына келісімдерді өңдеу жүйесі мен тіл өңдеу жүйесі де кіреді.

Мазмұны

Сәулеттік әрлем (әрлендірме) жүйенің дұрыс құрылу түсінігі мен жүйенің жалпы құрылымы әрлемінің түсінігіне тікелей байланысты. *2-тарауда* көрсетілгендей, Бағдарламалық қамтамасыздандыруды дамыту барысындағы үлгі бойынша, сәулеттік әрлеу – бағдарламалық қамтамасыздандыруды дамыту үрдісіндегі алғашқы қадамы болып табылады. Әрлеу мен техникалық талаптардың арасында сын байланыс бар, өйткені бұл жүйедегі маңызды құрылым құрауыштарын және олардың арасындағы байланысты анықтап көрсетеді. Сәулеттік әрлеудің нәтижесі жүйенің қалай құрастырылғанын байланысқан құрауыштар арқылы көрсететін сәулеттік үлгі болып табылады.

Алғыр үдеріс барысында үрдісті өңдеудің басы – жүйе сәулетінің құрылуымен байланысты болу керек. Сәулетті біртіндеп дамыту әрқашан сәтті бола бермейді. Құрауыштардың факторларын өзгерістерге жауап ретінде өзгерту әлдеқайда оңай, ал жүйенің сәулеттік факторларын өзгерту қымбат болып келеді.

Жүйе сәулеті сізге толығырақ түсінікті болу үшін *6.1-суретті* қарастырыңыз. Ол роботтық жүйені орауға арналған дерексіз үлгісінің дамытылатын құрауыштарын көрсетеді. Бұл роботтық жүйе сан түрлі объектілерді қамти (біріктіре) алады. Ол конвейердегі объектілерді бөліп көрсету үшін айқын құрауыштарды қолданады, объектінің түрін (типін) анықтайды және біріктірудің дұрыс жолын таңдайды. Содан кейін жүйе объектілерді жеткізу конвейерінен біріктіруге жібереді де біріктірілген объектілерді басқа конвейерге ауыстырады. Сәулеттік үлгі сол құрауыштар мен олардың арасындағы байланысты көрсетеді.

Тәжірибе барысында талаптарды өңдеу үрдісі мен сәулеттік әрлеу арасында зор байланыс бар екені көрінеді. Дұрыс дегенде, жүйелік спецификада ешқандай әрлеуге байланысты мәлімет болмау керек. Бұл тек шағын жүйелерге ғана арналған жалған ерекшелік. Сәулеттік жіктеу көбіне спецификацияны құрылымдау және ұйымдастыру үшін керек. Сол себепті, сіз талаптарды өңдеу үрдісінің бір бөлігі ретінде, дерексіз сәулеттік жүйені елестете аласыз. Бұл жүйеде сіз жүйелік функциялардың немесе үлкен көлемді құрамдауыштардың,ішкі жүйелердің топтарын байланыстыра аласыз. Кейін бұл жіктеулерді жүйенің талаптары мен ерекшеліктерін делдалдар арқылы қарастыра аласыз.

Бағдарламаның сәулетін екі жолмен құруға болады. Оларды мен *Шағын сәулет және Үлкен сәулет* деп атаймын:

1. *Шағын сәулет* – сәулеттің жеке бағдарламаларымен байланысты. Бұл деңгейде біз жіктелген жеке бағдарламалармен жұмыс істейміз. Бұл тарау көбіне, сәулеттік бағдарламалармен байланысты.
2. *Үлкен сәулет* – құрамына басқа жүйелер, бағдарламалар және бағдарламалық құрамдауыштары кіретін комплекстік кәсіпорындарға байланысты. Бұл кәсіпорындар жүйесі әртүрлі компьютерлерге бөлінеді. Мен Үлкен сәулет жайлы *18-* және *19-тарауларда* толығырақ қарастырамын.

Бағдарламалық қамтамасыздандыру сәулеті өте маңызды, өйткені ол жүйенің өнімділігіне, жөндеуге жарамдылығына,таратушылығына және мықтылығына әсер етеді. Бошты (Bosch) айтуы бойынша, жеке құрамдауыштар жарамды

жүйенің талаптарын жүзеге асырады. Жарамсыз талаптар жүйенің сәулетіне байланысты, яғни құрамдауыштардың құрылу жолына және қарым-қатынасына байланысты. Көптеген жүйелерде, жеке құрамдауыштар жарамсыз талаптарға да әсер етеді, бірақ әлбетте, жүйенің сәулеті басты әсер береді.

Bass et al (2003) айтуы бойынша бағдарламамен қамтамасыздандыру сәулетінің анық әрлеуі мен құжаттауының үш негізгі артықшылықтары:

1. *Делдалдық қарым-қатынас*: Сәулет – жүйенің әртүрлі делдалдарының арасындағы талқылауға толық көңіл бөлу үшін қолданылатын таныстырудың ең жоғары таныстыру жолы.
2. *Жүйені талдау*. Жүйені дамытудың бастапқы кезеңінде жүйені сәулеттендіруді толықтандыру талдауды талап етеді. Сәулеттік әрлеу нәтижелері жүйенің қиын талаптарына ұрынуына қарамастан өнімділікке, мықтылыққа және жөндеуге жарамдылыққа әсері зор.
3. *Кең тарапты қайта қолданыс*. Жүйелік сәулеттің үлгісі шығын, жүйенің қалай құрылғаны мен құрауыштарының өзара қатынасуының баяндалуы оңай меңгеріледі. Жүйенің сәулеті талаптары ұқсас жүйелер кең тарапты бағдарламалық қайта қолдануды қабылдай алады. *16-тарауда* айтып кеткенімдей, белгілі бір жүйеде қайта қолданылатын сәулеттердің өнім қатарын дамытуға болады.

Хофмейтердің (2000) ұсынысы бойынша – бағдарламалық сәулетті алдымен жүйе талаптарының келісімі үлгісінің әрлемі ретінде, содан кейін клиентпен, дамытушылармен және әкімшілікпен талқылау құрылымдауының (структура) жолы ретінде қарастыруға болады.

Жүйелік сәулет әдетте, қарапайым блок диаграммалары арқылы құрастырылады.

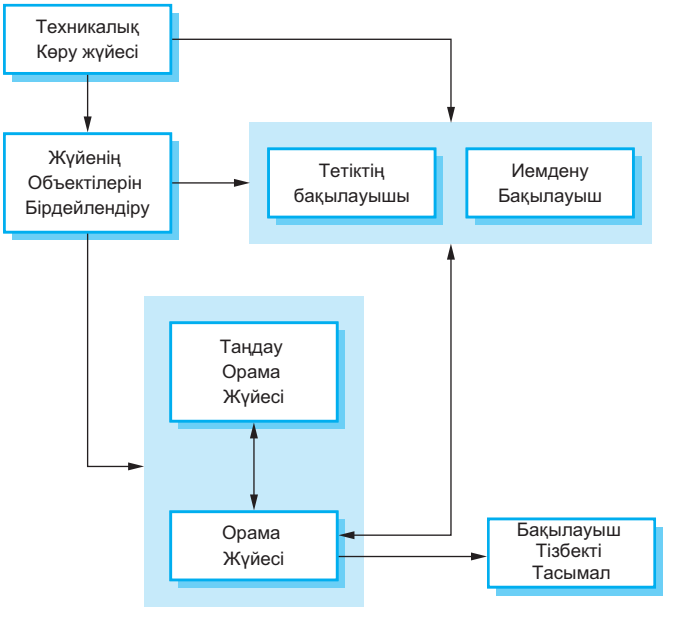
(6.1-сурет). Диаграммадағы әрбір ұяшық құрамдауышты білдіреді. Ұяшықтардың ішіндегі ұяшықтар ішкі құрамдауыштарға бөлінген. Нұсқауыштар мәліметтер немесе басқарушы сигналдардың құрамдауыштан құрамдауышқа нұсқауыштың бағытымен жолданғанын білдіреді. Бұл тәсілдің орындалу мысалдарын Боштың бағдарламасындағы сәулет каталогынан көре аласыз (Booch, 2009).

Блок диаграммалары жүйе құрылымының жоғарғы деңгейлі бейнесін көрсетеді, оны жүйені дамыту үрдісіне қабылданған әртүрлі пән мамандары оқып түсіне алады.

Бірақ олардың кең қолданысына қарамастан, Басс (2003) сәулетті бейнелеуге бейресми блок диаграммаларын қолдануды қолдамайды. Олардың айтулары бойынша бұл бейресми диаграммалар сәулетті ұсынудың нашар жолы, өйткені ол не жүйе құрауыштарының арасындағы байланыс түрін, не құрауыштардың сыртқы көрінетін қасиеттерін анық көрсетпейді.

Тәжірибе мен сәулеттік теориясы арасында анық қарама-қайшылықтар туындайды, өйткені сәулет үлгісінің программада қолданудың екі жолы бар:

1. Жүйе сәулетіне байланысты талқылауларда біріктіру жолы ретінде. Жүйенің жоғарғы дәрежелі сәулеттік нұсқсы жүйенің делдалдарымен және жобаны жоспарлаумен қарым-қатынасына өте қолайлы, өйткені ол бөлшектермен шатаспайды. Ал делдалдар болса, бұған байланыса алады әрі жүйенің дерексіз нұсқаларымен еш қиындықсыз байланысады. Содан кейін олар жүйені бір толық бүтін ретінде қарастырады, сонда бөлшектермен еш шатаспайды. Сәулеттік әрлеу үлгісі ерекше құрамдауыштарды анықтайды, олар қазір дамып келеді, дамығаны сонша, әкімшілік енді адамдарды жүйені дамыту бағдарламасына қабылдай алады.
2. Әрленген сәулетті құжаттандырудың бір жолы ретінде. Бұндағы мақсат жүйедегі әртүрлі құрамдауыштарды, олардың интерфейстері (түйіндер) және олардың құрамдауыштарын көрсететін бүтін жүйе үлгісін өңдеп шығару.



6.1-сурет. Робот бақылаушы жүйесінің жиналу сәулеті

Блок диаграммалары – әрлеу кезеңінде жүйе сәулетін баяндаудың сәйкес жолы, өйткені олар үрдіске қабылданған адамдар арасындағы қарым-қатынастың жақсы жолы. Көптеген жобаларда әдетте, жалғыз құжаттама болады. Сонда да, егер жүйе сәулеті жан-жақты құжаттандырылған болса, сәулеттік баяндауға арналған, жақсы анықталған семантикасы бар белгімен қолданған дұрыс. Бірақ, менің *6.2-бөлімде* талқылағанымдай, кей адамдар тым егжей-тегжейлі, нақты құжаттандырылу не қолайлы емес деп, не оған жұмсалған қаражатына түк тұрғысыз деп санайды.

6.1. Әрлеудің сәулеттік шешімдері

Сәулеттік әрлеу шығармашылық үрдіс, бұнда сіз мекеменің жарамды және жарамсыз жүйеге деген талаптарын қанағаттандыратын жүйе өңдей аласыз. Бұл шығармашылық жұмыс болғандықтан, үрдіске байланысты әрекеттердің барлығы өңделіп жатқан жүйенің типіне, сәулетшінің айналасы мен тәжірибесіне әрі жүйеге деген нақты талаптарға байланысты. Сол себепті сәулеттік әрлеуді қабылданатын шешімдер қатары ретінде қабылдаған, әрекеттер қатары ретінде қабылдағаннан әлдеқайда тиімдірек.

Сәулеттік әрлеу барысында, сәулетшілер жүйе мен оның дамуына байланысты құрылымдық (конструктивті) шешімдер қабылдағандары жөн. Олардың білімдері мен тәжірибелеріне байланысты, олар мына жүйеге байланысты негізгі сұрақтарды қарастырулары керек:

1. Жүйенің үлгісі ретінде қолданыла алатын өңделіп қойылған бағдарламаның жалпы сәулеті бар ма екен?
2. Жүйенің не ядроларының санына байланысты бөліне ме, не процессорларының санына байланысты бөліне ме?
3. Қандай сәулеттік үлгілер не стильдер қолданылады?
4. Жүйені құрылымдау үшін қандай негізгі тәсілдер қолданылатын болды?
5. Жүйедегі құралымдас құрауыштар қалай ішкі құрауыштарға бөлінетін болады?
6. Жүйедегі құрауыштарды реттеу барысын қадағау үшін қандай стратегиялар қолданылатын болады?
7. Жүйенің жарамсыз талаптарын анықтайтын қандай сәулеттік реттеу жолы жақсы болып саналады?
8. Сәулеттік әрлеу қалай бағаланады?
9. Жүйе сәулетшісі қалай құжаттану керек?

Әрбір жүйелік бағдарламаның өзіндік дара ерекшелігіне қарамастан, бір бағдарламалық аймақтағы жүйелердің сәулеттері бірдей болады, бұл аймақтың негізгі ұғымдарын көрсетеді. Мысалы, бағдарламалық өнім қатары - тұтынушының талаптарын қанағаттандыратын нұсқалары бар ядро сәулеті негізінде құралған бағдарламалар. Жүйенің сәулетін әрлеу барысында, осы уақытта сіздің жүйеңіз бен кеңтарапты бағдарламалық қатарларда ненің барын анықтап, сол білгеннің негізінде қаншасын қайта қолдануға болатынын шешкеніңіз жөн. Мен жалпы бағдарламалық сәулет жайында *6.4-бөлімде* қарастырып өттім және бағдарламалық өнімдер қатары жайында *16-тарауда* қарастырдым.

Енгізбелі жүйлер мен жеке компьютерлерге арналған жүйелерде әдетте, тек бір процессор болғандықтан сізге реттелген жүйе құрудың қажеті жоқ. Сонда да, ірі жүйелерде ішкі жүйелері бөлініп үлестірілген, ал олардың ішіндегі жүйелік бағдарламалар көптеген әртүрлі компьютерлер арасында үлестірілген. Сәулетті үлестіруге байланысты шешім жүйенің өнімділігі мен мықтылығына әсер ететін

негізгі шешім болып табылады. Бұл өз алдына маңызды тақырып әрі мен оны *16-тарауда* қарастырып кеттім.

Бағдарламалық жүйенің сәулеті белгілі бір сәулеттік үлгіге не стильге негізделуі мүмкін. Сәулеттік үлгі деген, жүйенің құрылымының сипаты (Garlan and Shaw, 1993), тұтынушы-сервер реттемесі немесе қабат-қабат сәулет сияқты. Сәулет үлгілері әртүрлі жүйеде қолданылған сәулеттің негізін қамтиды. Жүйенің сәулетіне байланысты шешімдер қабылдап жатқанда оның жалпы үлгілерін, олардың қайда қолданылатынын, мықты және әлсіз жақтарын білуіңіз керек. Жиі қолданылатын үлгілер жайлы *6.3-бөлімде* толық біле аласыз.

Гарлан және Шоу (Garlan and Shaw) сәулеттік стильдері (стиль мен үлгі түсініктері бүгін бір мағынада қолданылады) жоғарыда қарастырылған *4-* және *6-сұрақтарын* қамтиды. Сізге өзіңіздің жүйеңіздің талаптарын қанағаттандыратын тұтынушы-сервер не қабат-қабат реттеу сияқты ең тиімді жүйені таңдауыңыз керек. Құрылымдық жүйе бірліктерін бөлшектеп байланыстыру үшін, сіз алдымен, құрамдауыштарды шағын құрамдауыштарға бөлшектеп бөлу жолын таңдауыңыз керек. Қолдануға болатын амалдар неше түрлі сәулет типтерінің жүзеге асуына жол береді. Ақыр соңында соңғы модельдеу үрдісінде құрауыштарды өңдеу қалай бақыланатынын шешесіз. Сіз жүйедегі түрлі бөліктер арасындағы қарым-қатынасты бақылаудың жалпы үлгісін өңдейсіз.

Жарамсыз сәулет пен бағдарламалық сәулет арасындағы байланыс тығыз болғандықтан, сіздің жүйеге таңдайтын сәулеттік стиль мен құрылым жарамсыз талаптарға негізделуі тиіс:

1. *Өнімділік*. Егер өнімділік өте маңызды болып табылса, сәулет аз компоненттер арасындағы қиын операцияларды шектеу үшін жасалуы керек, әрі бұл құрауыштар бір компьютерде жүзеге асу керек, желіде емес. Бұл құрауыштардың арасындағы қарым-қатынасты азайтатын біршама үлкен құрауыштарда қолданылуы тиіс. Сіз жүйенің іске асыру уақытын реттеуін де қарастыра аласыз, бұл жүйенің әртүрлі процессорларда орындалып өңделуіне жол береді.
2. *Қауіпсіздік.* Егер қауіпсіздік маңызды талап болып табылса, ішкі қабаттардағы қорғалатын активтері бар сәулетке арналған қабат-қабат структура қолданылуы тиіс әрі сол қабаттарға қатынастың жоғарғы дәрежелі қорғанысы болуы керек.
3. *Қауіпсіздік.* Егер қауіпсіздік маңызды талап болса, сәулетші қауіпсіздікке байланысты операциялар кез келген бір құрамдауышта не аз ғана құрамдауыштар қатарында орналасуы керек. Бұл қауіпсіздік тексерісінің бағасы мен мәселелерін азайтады әрі жаңылу барысында жүйенің жұмысын қауіпсіз аяқтауға көмектеседі.
4. *Қолжетімділік.* Егер қолжетімділік маңызды талап болса, сәулет жүйені тоқтатпай жаңаландырып, ауыстыруға оңай болатындай керек емес құрамдауыштарды қосуға бағытталуы керек. *13-тарауда* екі бас тартуға шыдамды жүйе сәулеті туралы молырақ мәлімет ала аласыз.
5. *Жөндеуге жарамдылық.* Егер жөндеуге жарамдылық маңызды талап болса, жүйе сәулетінде ұсақ түйірлі, дербес құрамдастар қолданылып құрылуы

керек. Өңдеушілер тұтынушылардан дербестендірілу керек және ортақ мәліметтер құрылымдарынан аулақ болу керек.

Осы сәулеттердің арасында үлкен бір келіспеушілік бар екені анық. Мысалы, үлкен құрауыштарды қолдану өнімділікке, ал ұсақ құрамдауыштарды қолдану жөндеуге жарамдылыққа әсер етеді. Егер өнімділік те, жөндеуге жарамдылық та маңызды талап болса, белгілі бір келісім болу керек. Бұл кейде жүйенің әртүрлі бөліктеріне қатысты неше түрлі сәулеттік үлгілер немесе стильдер қолдану арқылы жүзеге асырыла алады.

Сәулеттік әрлеуді бағалау өте қиын, өйткені оның сапасын анықтайтын фактор – жүйенің жарамды-жарамсыз талаптарын қолдану барысында шешім қабылдау қабілеті. Бірақ сонда да, сіз өз дизайныңыз бен эталонды сәулеттермен немесе сәулеттік үлгілермен салыстыра отырып, кішігірім баға бере аласыз. Боштың (Bosch's (2000)) айтуы бойынша сәулеттік үлгінің жарамсыз талаптары баға беруге жақсы көмектеседі.

6.2. Сәулеттік көзқарастар

Бұл тараудың басында мен сәулеттік үлгілердің бағдарламалық жүйелерде, бағдарламаның талаптары мен әрлеміне байланысты талқылауларда қолданыла алатынын айтып өттім. Сонымен қатар олар көптеген толық әрлеулер мен өңдеуге және жүйенің болашақ дамытуларына негіз болатындай әрлеуді құжаттандыру үшін қолданыла алады. Осы тарауда мен мыналарға қатысты екі мәселені қарастыратын боламын:

1. Жүйенің сәулетін әрлеу мен құжаттау барысында қандай пікірлер мен перспективалар ең тиімді?
2. Сәулеттік үлгілерді бейнелегенде қандай белгілерді қолданған жөн?

Әрбір сәулеттік үлгі жүйенің барлық мүмкіндіктерін көрсететіндіктен, бір ғана үлгіде жүйенің сәулетіне байланысты барлық мәліметті көрсету мүмкін емес. Ол тек жүйенің қандай модульдерге бөлінетіндігін, үрдіс барысы қалай жүретіндігін немесе жүйе құрамдауыштарының желіде бөліну жолдарын көрсете алады. Бұлардың бәрі әрқалай қолайлы болады, сол себепті әрлеу мен құжаттау екеуіне де бағдарламалық сәулеттің бірнеше үлгісін құрғаныңыз жөн.

Сонымен қажетті үлгілердің бірнеше түрі бар. Крутчен (Krutchen (1995)) деген танымал бағдарламалық сәулеттің 4+1 үлгісінде негізгі қолдану жолдары немесе сценарийлерді қолданумен байланысқан 4 сәулеттік үлгі болу керек деген болжам білдіреді. Оның болжайтын үлгілері:

1. Объектілер ретінде не объектілер классы ретінде көрсететін негізгі абстракцияларды көрсететін логикалық үлгілер.

2. Үрдіс барысында жүйенің қалай өзара байланысқан үрдістерден тұратындығын көрсететін үрдістік үлгі. Бұл үлгі жүйенің өнімділік пен қолжетімділік сияқты жарамсыз мінездемесіне қатысты сындарды анықтауға көмектеседі.
3. Бағдарламаның өңдеуде қалай бөлінгендігін көрсететін өңделмелі үлгі, яғни бағдарламаның бір өңдеушімен немесе өңдеушілер тобымен құрамдауыштарға қалай бөлінгендігін көрсетеді. Бұл үлгі әкімшілік пен программистерге қолайлы.
4. Жүйелі техникалық қамтамасыздандыру мен жүйе құрамдауыштарының жүйеде қалай бөлінгендігін көрсететін физикалық үлгі. Бұл үлгі жүйе инженерлеріне жүйені қолдану жолдарын жоспарлауда қолайлы.

Гофмейстердің (Hofmeister et al. (2000)) болжауы бойынша, ұқсас үлгілерді қолданып қана қоймай бұл ұғымға концептуалдық үлгіні қосу керек. Бұл үлгі – жүйенің дерексіз үлгісі, ол ірі талаптарды мамандарға бөлуге егжей-тегжейлі көмектескенінің арқасында дара жүйеге қарағанда инженерлерге қайта қолданылатын құрамдауыштарды таңдауға және өнім қатарын ұсынуға көмектеседі. *6.1-суретте* көрсетілген жинаушы роботтың сәулеті концептуалды жүйе үлгісінің мысалы.

Тәжірибеде концептуалды үлгілер көбіне, әрлеу барысында өңделеді және сәулеттік шешімдерде қолданылады. Олар әртүрлі дәнекерлерге жүйенің мәнін жеткізетін бір тәсілі болып табылады. Әрлеу үрдісі барысында жүйенің түрлі қырлары қарастырылып жатқанда басқа да үлгілер өңделе алады, бірақ барлық жақтарынан түсіндірудің қажеті жоқ.

Сәулеттік түсінікке қатысты UML-да бағдарламалық сәулеттердің қолданылу не қолданылмауына қатысты көп пікірлер бар. 2006 жылғы сауалдама нәтижелері бойынша UML-ды қолданғанда көбіне, бейресми және еркін тәсілдер қолданылған. Мен бұл пікірмен келіспеймін. UML объектіге бағдарланған жүйелерді сипаттауға жасалған және сәулеттік әрлеу дәрежесінде жүйені әрқашан жоғарғы дәрежелі абстракцияда бейнелеуге мүмкіндік береді.

Сондықтан мен әрлеу барысында UML-дан гөрі бейресми тәсілдерді қолданғанды жақсы көрем, оларды қолданған тезірек әрі тақтада жазуға ыңғайлы. UML-дың қажеті – сәулетті егжей-тегжейлі құжаттағанда немесе үлгі негізінде құрылған өңдеулерде артады.

Өңдеушілер қатары арнайы сипаттаманың архитектуралық тілдерін ұсынады (CAT). CAT(ADLs). Негізгі элементтері – құрамдауыштар мен байланыстырушылар. Бірақ олардың арнайы негізіне, доменіне және аддендумына байланысты мамандар CAT-ты қолдану түсінуге қиын деп санайды. Белгілі бір домэнге арналып жасалған CAT үлгіге негізделген зерттемелерде қолданыла алады.

Қолданушылар егжей-тегжейлі құжатталған тәсілдер көбіне, қолданбайды деп айтады. Мен де бұл пікірмен келісемін, олар мүлдем қолайсыз.

6.3. Сәулеттік қалыптар

Бағдарламалық жүйелерді айырбастау пен қайта қолдану туралы білімдердің белгісі ретінде үлгі түсінігі бүгінгі таңда жиі қолданылады. Бұған себепкер болған нобайдың объектіге негізделген үлгілері туралы кітабының шығуы, бұл өз алдына басқа үлгілердің дамуына әсер етті. Архитектуралық үлгілер 1990 жылы «архитектуралық стиль» деген атпен енгізілді, 1996 – 2007 жылдар аралығында 5 томды анықтама кітаптары шықты (Buschmann et al., 1996; Buschmann et al., 2007a; Buschmann et al., 2007b; Kircher and Jain, 2004; Schmidt et al., 2000).

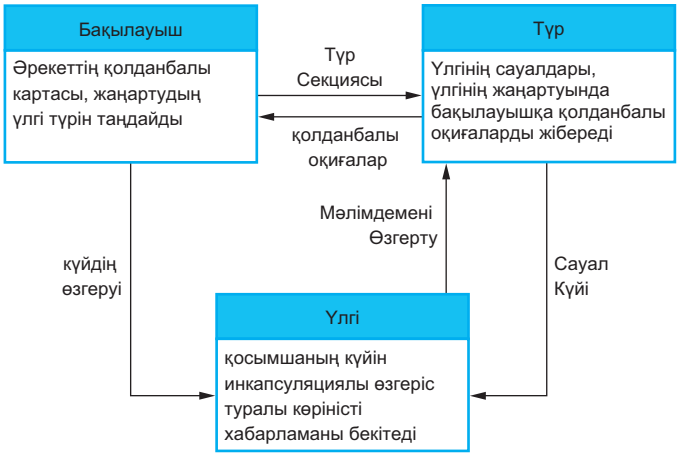
Бұл тарауда мен архитектуралық үлгілермен таныстырып, олардың қандай түрлері көбіне, қолданылатынын айтып кетем. Үлгілер мен олардың қолданысы жайлы толық мәліметті анықтама кітаптардан алсаңыз болады.

Архитектуралық үлгіні бір стильге келтірілген, тәжірибенің жақсы дерексіз сипаттамасы деп елестетсеңіз болады, олар басқа жүйелер мен орталарда сыналған. Сонымен, архитектуралық үлгі алдыңғы жүйеде сәтті өткен жүйенің құрылысын сипаттауы керек. Ол үлгінің қашан қолданылып – қолданылмау керектігін, мықты – әлсіз жақтарын толық сипаттау керек.

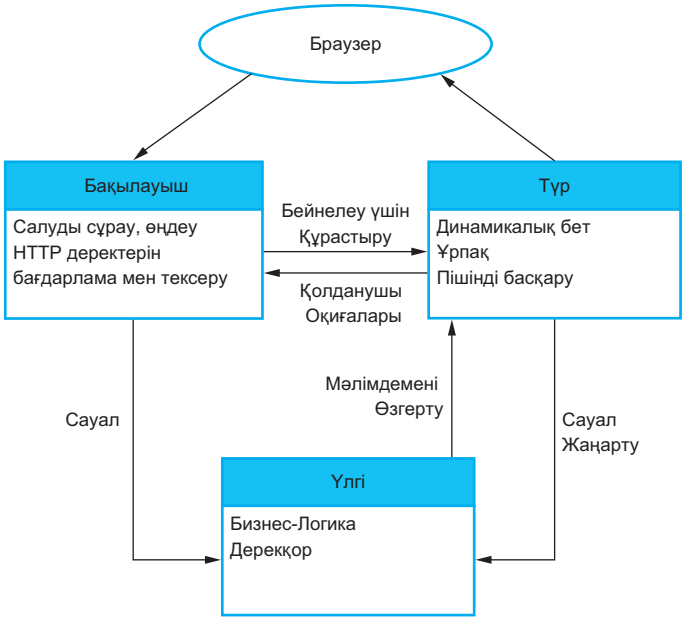
Аты	MVC (модельді көріп бақылау)
Анықтама	Дерек жүйесінен көрсетілім мен қатынастарды алшақтатады. Жүйе бір-бірімен байланысқан негізгі үш компоненттерінен құрылады.
Мысал	6.4-суретте MVC сызбасын қолданған вед-қосымшасының сәулеті көрсетілген.
Қолданыс уақыты	Деректермен қатынасу тәсілдерінің әртүрлі болған жағдайында қолданылады. Және де болашақ талаптардың белгісіз болған жағдайында қолданады.
Басымдылық	Деректердің көрсетілімінен тәулсіз өзгертілуіне мүмкіндік береді.
Кемдік	Деректердің қатынасы қарапайым болған жағдайында да, қосымша кодты қосып, жүйені күрделілендіру мүмкін.

6.2-сурет. MVC сызбасы

Мысалы, *6.2-суретте* әйгілі Үлгі-Моделін-Бақылау үлгісі. Бұл үлгі көптеген желілі жүйелердегі басқармалар қатынасының негізі. Бір стильге келтірілген үлгі сипаты оның атын, толық сипаттамасын және оның қолданылатын жүйесінің типінің мысалын қамтиды. Архитектураның MVC үлгісімен байланысқан графикалық үлгілер *6.3-* және *6.4-суреттерде* бейнеленген. Олар әртүрлі үлгілерден алынған архитектураны көрсетеді; *6.3-суретте* коцептуалды үлгі, ал *6.4-суретте* жұмыс үстіндегі архитектура бейнеленген, бұнда желілі жүйедегі басқарма қатынасы көрсетілген.



6.3-сурет. MVC-нің құрылымы



6.4-сурет. MVC құрылымын қолданатын веб қосымшасының сәулеті

Шағын тарауда бағдарламалық зерттемелерді қамтып айту мүмкін емес, бірақ мен сонда да негізгі деген, маңызды деген мысалдар туралы айтып кеттім.

6.3.1. Көп дәрежелі архитектура

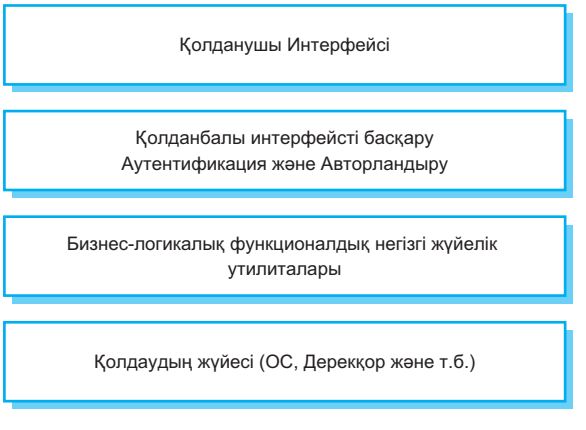
Бөліну мен дербестену архитектуралық әрлеудің негізі болып табылады, өйткені олар өзгерістерді жалпыландыруға көмектеседі. *6.2-суретте* көрсетілген MVC

үлгісі жүйедегі элементтерді жеке бөледі де, оларға дербес өзгеруге мүмкіндік береді. Көп дәрежелі архитектуралық үлгі – дербестік пен бөлінуге қол жеткізудің тағы бір жолы. Бұны *6.5-суреттен* көруге болады.

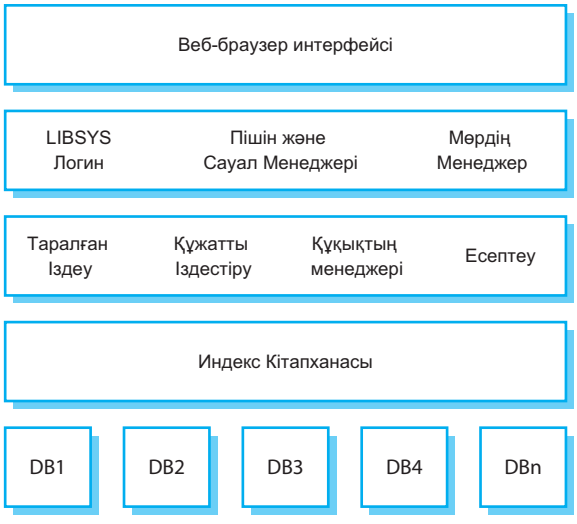
Аты	Реттелген сәулет
Анықтама	Жүйені функционал бойынша реттейді. Әр деңгей белгілі бір қызметтерді қамтиды. Негізгі қызметтер төменгі деңгейде тұрады.
Мысал	6.7-суретте көрсетілген құжаттарды көшіретін жүйенің ортақтасу үлгісі.
Қолданыс уақыты	Бар жүйелерге қосымша функционалды қосқанда қолданылады.
Басымдылық	Әр деңгейдегі қызметтердің орын ауыстыруына рұқсат береді.
Кемдік	Тәжірибеде белгілі бір қызметтің деңгейін анықтау қиын болады және төменгі деңгейдегі қызметтерге көңіл көп бөлінеді.

6.5-сурет. Сәулеттің реттелген сызбасы

Көп дәрежелі тәсіл жүйенің біртіндеп дамуын қабылдайды. Әр деңгейде кейбір қызметтер өңделеді және олар кей қолданушыларға қолайлы бола алады. Архитектура да өз алдына өзгермелі және қозғалмалы келеді. Егер интерфейс көп уақыт өзгертілмесе, оны ұқсас басқа интерфейспен алмастыруға болады. Көп дәрежелі жүйе ішкі дәрежелерде машинаға тәуелділігін кеңтараптандырғандықтан қолданбалы жүйенің бірнеше платформада жүзеге асырылуын жеңілдетеді. Тек ішкі машинаға тәуелді дәрежелер ғана қайта жүзеге асыруды талап етеді.



6.6-сурет. Тізбектелген сәулет



6.7-сурет. LIBSYS жүйесінің сәулеті

6.6-сурет көп дәрежелі архитектураның төрт дәрежелі мысалы. Ең төменгі дәрежеге бағдарламаны қолдайтын жүйе, әдеттегі мәліметтер базасы мен операциялық жүйе кіреді. Келесі дәрежеде іске асыру дәрежесі, оның ішіне айқындауыш фунционалдылығымен байланысты құрамдуыштар кіреді. Үшінші дәрежеде қолданушының аутентификациясы мен кіру рұқсатымен қамтамасыз ететін басқарма және қолданушы интерфейсінің мүмкіндіктерінің дәрежесі кіреді.

6.3.2. Мұрағаттың сәулеті

Көп дәрежелі архитектура мен MVC үлгілері жүйенің концептуалды құрылымын сипаттайтын үлгі мысалдары. Менің келесі келтірер мысалым – қоймалы үлгі, ол байланысқан құрамдауыштардың өзара қалай мәліметермен бөлісетіндігін көрсетеді.

Көп мәліметпен жұмыс істейтін жүйелердің көбісі ортақ мәліметтер базасымен немесе қоймалармен жұмыс істейді. Сол себепті бұл үлгі бір құрамдауышта құрылып, басқасында қолданылатын бағдарламаға келеді. Бұл типтің үлгісі бұйрық пен бақылау жүйесінен, мәліметті басқару жүйесінен, CAD жүйесінен және бағдарламаның диалогті өңдеу аймағынан құралады.

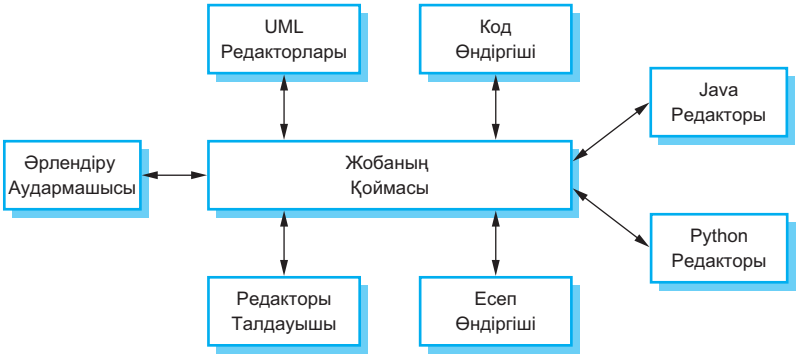
6.9-суретте қоймалардың қалай қолданылу керек үлгісі көрсетілген. Диаграмма үлгіге негізделген өңдеуді қолдайтын әртүрлі жабдықтардан тұратын IDE-ді көрсетеді. Бұл жердегі қоймалардың рөлі бағдарламадағы өзгерістерді басқарып, алдыңғы нұсқаларға қайтаруға негізделген.

Қоймалардың маңайында жабдықтарды ұйымдастыру мәліметтермен көп мөлшерде бөлісуге мүмкіндік береді. Мәліметті бір құрамдауыштан екінші

құрамдауышқа анық жеткізудің қажеті жоқ. Бірақ сонда да бұнда мәліметтің артықшылығы мен сәйкес келмеушілігімен байланысты мәселелер болуы мүмкін.

Аты	Сақтау қоймасы
Анықтама	Жүйенің деректері сақтау қоймасында реттеліп, жүйенің барлық компоненттеріне қолжетімді болады.
Мысал	6.9-суретте жүйенің әрленімі туралы ақпараттың сақтау қоймасында сақталғаны көрсетілген.
Қолданыс уақыты	6.9-суретте жүйенің әрленімі туралы ақпараттың сақтау қоймасында сақталғаны көрсетілген.
Басымдылық	Компоненттер бір-бірінен тәуелсіз және бір-бірі туралы хабарсыз болады.
Кемдік	Сақтау қоймасындағы сәтсіздіктер бүкіл жүйенің сәтсіз орындалуына алып келеді.

6.8-сурет. Сақтау қойманың сызбасы



6.9-сурет. IDE-нің деректер сәулеті

6.3.3. Тұтынушы-сервер архитектурасы

Қоймалы үлгі жүйенің орныққан структурасымен байланысқан және жұмыс барысындағы құрылысын көрсетпейді. Менің келесі келтіретін мысалым – кең қолданылатын таратылған жүйеге арналған жұмыс барысындағы құрылыс. Тұтынушы-сервер үлгісі *6.10-суретте* көрсетілген.

Тұтынушы-сервер үлгісі қызмет пен байланысқан серверлер қатары арқылы, қолданатын тұтынушылар арқылы құрылған. Бұл үлгінің маңызды құрамдауыштары:

1. Қызметтерді басқа құрамдауышқа ұсынатын серверлер қатары. Бұған мысал: Принтер.

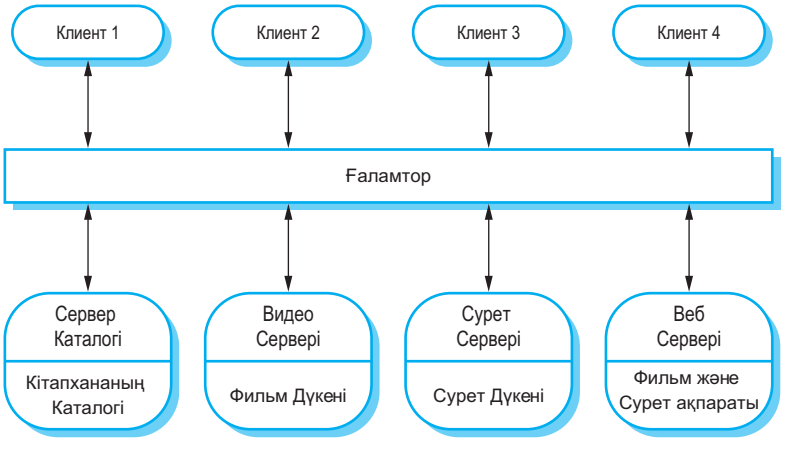
2. Сервермен ұсынылған қызметтерді шақыратын тұтынушылар қатары. Бұнда әдетте, тұтынушылар программасының бірнеше үлгісі болады әрі олар бірнеше компьютерлерде қатар орындалады.
3. Тұтынушыларға осы қызметтерге кіруге рұқсат беретін желі. Көптеген тұтынушы-сервер жүйелері бөлшектенген жүйе ретінде өңделеді әрі интернет хаттамалары арқылы байланысады.

Аты	Тұтынушы-сервер
Анықтама	Тұтынушы-сервер сызбасында әр қызмет әртүрлі сервермен тасымалданады. Пайдаланушылар осы қызметтерді пайдаланады және оларды пайдалы етеді.
Мысал	6.11-суретте DVD бейнетаспалар жинағы тұтынушы сервер қызметі ретінде көрсетілген.
Қолданыс уақыты	Деректердің ортақ болған жағдайларында қолданылады.
Басымдылық	Бұл жүйе желілерде қолдана алады. Барлық пайдаланушыларға қолжетімді болады.
Кемдік	Бір қызметтің сәтсіз орындалуы сол серверге байланысты басқа қызметтердің сәтсіздігіне алып келуі мүмкін.

6.10-сурет. Тұтынушы-сервер сызбасы

Тұтынушы-сервер аритектурасы әдетте, бөлшектенген жүйе ретінде жоспарланады, бірақ әртүрлі жүйелерде жүзеге асырылған тәуелсіз қызметтердің логикалық үлгісі бір компьтерде жасалады.

Тұтынушылар қамтамасыз етілген қызметтер мен жетімді серверлердің атын білулері тиіс. Тұтынушылар серверге кіру үшін шақыру командалары арқылы сұрақ-жауап хаттамаларын шақырады, мысалы, http хаттамасы – WWW.



6.11-сурет. Бейнетаспа жинағына арналған тұтынушы-сервер сәулеті

6.11-суретте тұтынушы-сервер үлгісінде негізделген үлгі көрсетілген. Бұл – бірнеше тұтынушыға арналған, желі жүйесінде негізделген фильмдер мен фотосуреттер кітапханасының жүйесі. Видео роликтер тез ойналу керек, бірақ әрине сапасы да төменделеді. Фотосуреттердің сапасының дәрежесін сақтау мүмкіндігі зор болғандықтан оны бірнеше серверлерде сақтауға болады.

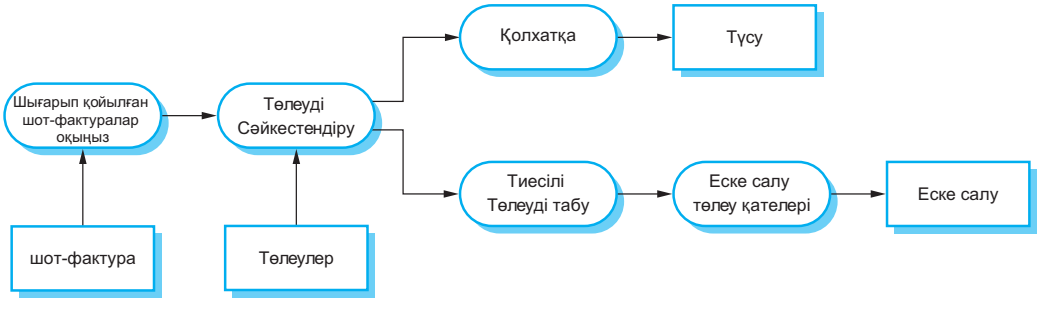
Тұтынушы-сервер үлгісінің негізгі артықшылығы оның бөлшектенгендігінде. Бұнда жұмыстың желі жүйесінде орындалуға қолайлы мүмкіндігі бар. Жаңа серверді жүйеге еш зиянын келтірмей қосып-өшіретін мүмкіндігі бар. Бұл туралы *18-тарауда* толық оқи аласыз.

Аты	Мұржа және сүзгі
Анықтама	Жүйедегі деректердің орындалуы әр компонентті тәуелсіз қылып реттейді. Кейін сол компоненттердің сипаттарына сай сүзгілеуге мүмкіндік береді.
Мысал	6.13-суретте инвойстарды орындайтын мұржа және сүзгі жүйесі көрсетілген.
Қолданыс уақыты	Деректерді орындайтын қосымшаларда қолданылады.
Басымдылық	Түсінікті және қайта қолдануды демейді.
Кемдік	Деректердің форматы келісілген болуы керек.

6.12-сурет. Мұржа және сүзгі сызбасы

6.3.4. Арналар мен фильтрлердің сәулеті

Менің соңғы келтірер мысалым – арналар мен фильтрлер үгісі. Бұл үлгіде жұмыс барысында жүйе енгізілген мәліметтер мен нәтижелерді басқарады. Мәлімет бір жерден екінші жерге жеткізіледі және олар цикл арқылы жүреді. Енгізілген мәлімет нәтижеге ауыстырылғанша өңделеді.



6.13-сурет. Арналар мен фильтр сәулетінің мысалы

«Арналар мен фильтрлер» деген атау Юникс жүйесінен енгізілді, мағынасы арналар арқылы үрдістерге қатынасуға болады. Бұл мәтінді бір үрдістен екінші

үрдіске жібереді. Осы үлгі арқылы жасалған жүйелер Юникс командаларын қолдана алады, «фильтр »деген ұғым енгізілген мәліметтің нәтижеге жеткенше өңделетіндігі үшін пайда болған.

Басқаруға арналған құрылымдық үлгілер

Бұл жүйе ішіндегі басқаруды ұйымдастыратын ортақ пайдаланатын жолдарды көрсететін арнаулы құрылымдық үлгілер болып табылады. Олардың құрамына басқа компоненттерді шақырушы бір компонент және жүйенің сыртқы оқиғаға жауап қайтару әрекеті жүретін оқиғаға негізделген басқару негізіндегі орталықтандырылған басқару қамтылған.

http://www.SoftwareEngineering-9.com/Web/Architecture/ArchPatterns//

6.4. Қолданбалы сәулеттер

Бағдарламалық жүйелер бизнестің немесе мекемелердің талаптарын қанағаттандыру үшін жасалады. Барлық бизнес орталықтарының ортақ талаптары: адамдарды жалдау, мәселелерді шешу, есептік жазбалар жазу және т.б. Ұқсас жүйеде жұмыс істейтін орталықтар көп, сол себепті оларға өз жүйелеріндегі қызметтерді басқару қажет.

Бұл ұқсастықтар бағдарламалық жүйенің толық жетілдіріліп өңделуіне жол ашты. Бірнеше орталықтың жүйесі ұқсас болуы мүмкін, бірақ әрине, атаулары мен кластары ерекшеленеді, сол себепті бір үлгіні өзгертіп қолдана беруге болады.

Кей жағдайларда жүйені қайта құрғанда жаңа архитектураны құрастырудың қажеті жоқ болады. Бұны кәсіпорындар қорының жаспарлау жүйесінен көруге болады, мысалы, SAP немесе Oracle. Бұл жүйелер арнайы кәсіпорындарға арналған. Мысалы, бір жүйе, мәліметтер базасы бірнеше жерде, мысалы, тұлғалық мәліметтер керек жерде және т.б. тиімді.

Қолданба құрылымдары

«Бұл кітаптың веб-сайтында қолданба құрылымдарына арналған бірнеше мысал келтірілген. Олардің ішіне буманың дерек өңдеу жүйелері, ресурстарды бөліп орналастыру жүйелері және оқиғаға негізделген өңдеу жүйелері кіреді.

http://www.SoftwareEngineering-9.com/Web/Architecture/AppArch/

Бағдарламаның дизайнері ретінде сіз сәулеттік үлгілерді бірнеше жолда қолдана аласыз:

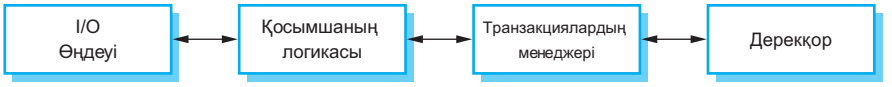
1. Сәулеттік әрлеу үрдісінің негізі ретінде. Егер сіз бұрын-соңды бұндай үлгімен жұмыс істемесеңіз, сіз дайын үлгілерді өзіңіздің бастауларыңызға негіз қылып алсаңыз болады.
2. Дизайнды сынау үшін. Өзіңіздің жасаған сәулеттік әрлеуіңізді басқа сәтті үлгілермен салыстырып, өзіңіздің жұмысыңызды жетілдірсеңіз болады.
3. Әзірлеушілер тобын ұйымдастыру үшін. Кейде жақсы нәтиже алу үшін басқа да дизайнерлемен бірге қатар жұмыс істеген қолайлы. Ортақ біліммен үлкен нәтиже алуға болады.
4. Қайта қолданылатын құрамдауыштардың мағынасын білу үшін. Егер сіз қайта қолдануға болатын құрамдауыштарды тапсаңыз, олардың түп мағынасын түсіну үшін танымал үлгілерді қарастырған қолайлы.
5. Бағдарламалар типінің сөздігі ретінде қолдану үшін. Егер қиын ұғымдарды қарастырып жатсаңыз, танымал үлгілердегі құрамдауыштардың қатарын сөздік ретінде қолдануға болады.

Бағдарламалық жүйенің бірнеше түрлері бар және олар өзара ерекше болып көрінеді, бірақ олардың көп ортақ тұстары бар, сол себептері бір бағдарламалық сәулет ретінде ұсыныла алады. Мен бұларды екі түрлі сәулеттерді сипаттау арқылы түсіндіріп өтемін.

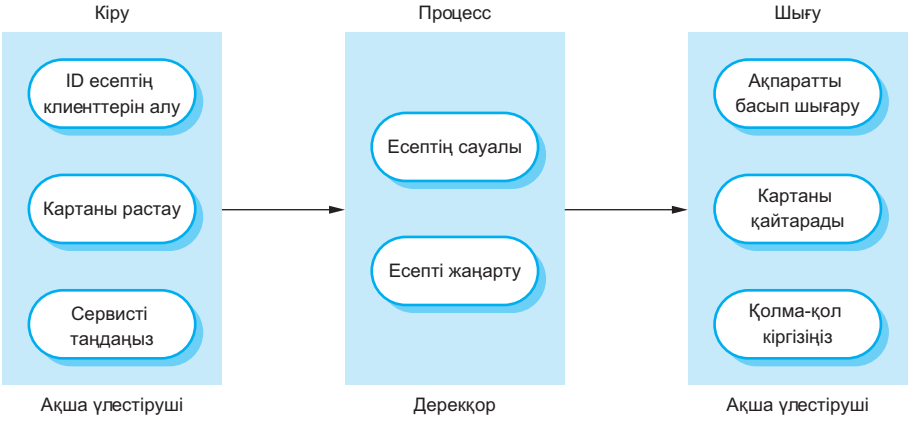
1. *Транзакцияларды өңдеу үрдістері.* Транзакция өңдеу бағдарламалары мәліметтер базасына негізделген бағдарламалар болып табылады, олар қолданушы сұрастырған мәліметтерді өңдеп, мәліметтер базасындағы мәліметтерді жаңартады. Бұл бизнес орталықтарында жиі сұранатын қызметтер. Олар тек сол орталықтардағы қолданушыларды: бухгалтериялық есептеулерді, желілі қызметтерді, кітапханаларды және т.б. байланыстырады.
2. *Тіл өңдеу жүйесі.* Тіл өңдеу жүйесі деген қолданушыға түсінікті тілмен жазылған бағдарламалар (мысалы, Java). Бұндай тілдер белгілі бір халықаралық кесіліген ортақ терминдер мен ұғымды қолданады. Олар жоғары деңгейлі тілдер деп саналады. Бірақ сонда да бұл тілдердің өздері мәліметтер базасының кейбір ұғымдарын қолданады,мысалы, XML.

6.4.1. Транзакцияларды өңдейтін жүйелер

Транзакцияларды өңдеу (ТӨ) жүйелері қолданушының мәліметтер базасынан сұрастырған мәліметтеріне қатынасуын жеңілдету үшін қолданылады. Техникалық тұрғыдан қарағанда, мәліметтер базасындағы транзакциялар операциялар жүйесі болып табылады, олар бір дәрежеде орындалады. Барлық өңдеулер база өзгергенше немесе сақталғанша жасалу керек.



6.14-сурет. Қосымшаларды орындау транцакцияларының құрылымы



6.15-сурет. АТМ жүйелерінің бағдарламалық сәулеті

Транзакцияларды өңдейтін жүйелер әдетте, желілі жүйелер болып келеді, бұнда қолданушылар асинхронды сұраныстар жасай алады. *6.14-суретте* ТӨ бағдарламасында жасаған концептуалды сәулетке мысал келтірілген. Транзакция менеджерге жасалып жіберіледі, ол болса, өз алдына базаның әкімшілік жүйесіне тіркелген.

ТӨ жүйесі арна мен фильтр сәулетінде де орындала алады, бұнда жүйедегі құрамдауыштар кіріске, өңдеуге және нәтижеге жауапты.

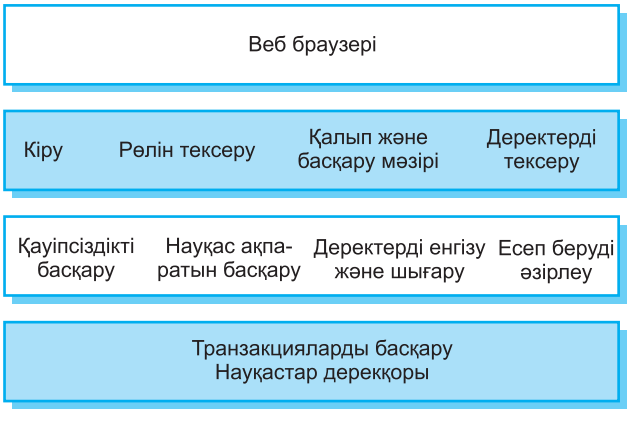
6.4.2. Мәліметтік жүйе

Мәліметтер жүйесімен байланысқан жүйелерді транзакцияға негізделген мәліметтер жүйесі ретінде қарастыруға болады. Мәліметтер жүйесі үлкен көлемді мәліметтерге қатынасты қамтамасыз етеді, мысалы, кітапханалық каталог, ұшақтардың күтізтізбесі, ауруханадағы науқастар жайлы мәліметтер. Көбіне, бұндай жүйелер желілі түрде құрылады.

6.16-сурет мәліметтік жүйенің жалпы мысалы. Жүйе қоймалы тәсілді қолданылып, құрылған,бұнда жоғарғы қабаттар қолданушының интерфесі мен жүйелік мәліметтер базасы болып табылады. Бұндағы кіріс пен нәтижені өңдеудің өзіндік ерекше жолы қолданылған.

Қоймалы үлгінің бір экземпляры ретінде *6.17-сурет* көрсетілген, ол MHC-PMS сәулетін сипаттайды. Бұл жүйеде жындыханадағы науқастардың ем алу жайындағы мәліметтері көрсетілген.:

6.16-сурет. Жүйе сәулетінің тізбектелген ақпараты



6.17-сурет. MHC-PMS-тің сәулеті

1. Жоғарғы қабат пайдаланушының интерфейсін (ПИ) жариялауға жауапты. ПИ жариялау браузердің көмегімен құрастырылған.
2. Екінші қабат ПИ-дің функционалдығын браузерге үлестіреді. Бұнда пайдаланушының рөлдерін басқаратын құрамдауыштарды қадағалайтын компоненттер бар.
3. Үшінші қабат жүйенің функционалдылығын және жүйенің қауіпсіздігін қамтамасыз ететін құрамдауыштарды жариялайды.
4. Ең соңғы қабат коммерциялық мәліметтер базасын басқару жүйесімен құрастырылған, ол транзакцияларды басқаруын және тұрақты мәліметтер жадын қамтамасыз етеді.

Мәліметтер мен қорды басқару жүйесі бүгінгі таңда көбіне, желілі болып табылады, мұнда пайдаланушының интерфейсі браузерде жарияланады.

Бұл жүйелердегі серверлерді реттеу *6.16-суретте* көрсетілгендей көбіне, төрт қабатты жалпы үлгіде түсіндіріледі. Бұл жүйелер көбіне, көп дәрежелі тұтынушы сервер/сәулеттерде жарияланады, бұл *18-тарауда* сипатталғандай:

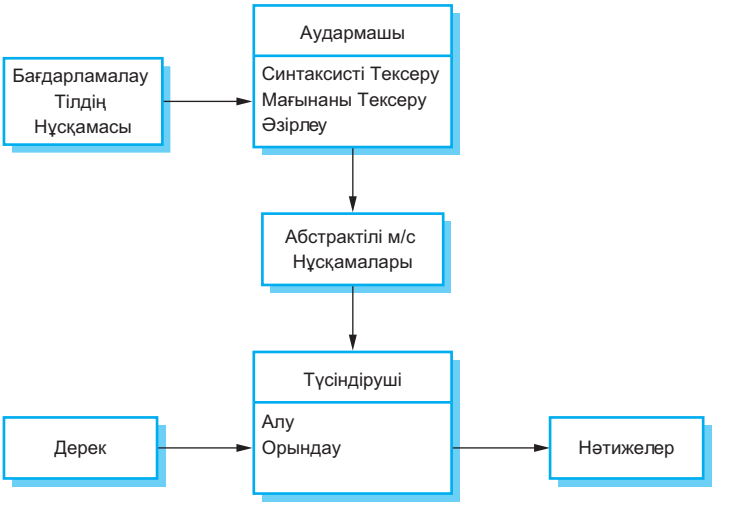
1. Желілі сервер пайдаланушының интерфейспен браузерді қолдана отырып қатынасуына жауапты.
2. Бағдарламаның сервері бағдарламаның ерекше логикасын жүзеге асыруына жауапты, сонымен қатар мәліметтерді сақтау мен мәліметтерді іздеу операцияларына жауапты.
3. Мәліметтер базасы мәліметтерді базадан енгізіп, өшіреді және транзакцияларды басқарады.

Бірнеше серверлерді қолдану тез өткізгіштік пен жүздеген транзакцияларды орындауға мүмкіндік береді. Егер талап өссе, әр қабатқа қосымша серверлер қосуға, сонда қосымша үрдістерді жүргізуге болады.

6.4.3. Тілді өңдеу жүйесі

Тіл өңдейтін жүйелер шынайы әрі жасанды тілдерді сол тілдің басқа түріне аударады, енгізілген кодты да аудара алады.

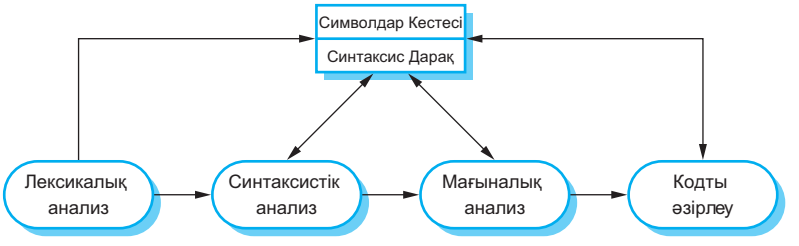
Бағдарламалық қамтамасызданыруда компиляторлар жасанды программалық тілдерді машинаның тіліне аударады. Басқа тіл өңдейтін жүйелер XML-дан түскен мәліметтерді базаның тіліне не жасанды XML тіліне аударады. Табиғи тілдерді бір табиғи тілден екінші табиғи тілге, мысалы, ағылшын тілін қазақ тіліне аударады.



6.18-сурет. Тілді орындау жүйесінің сәулеті

Тіл өңдейтін жүйенің тілді программалауға мүмкін сәулет *6.18-суретте* бейнеленген. Программаның командалары оның орындай алатын мүмкіндіктерін сипаттау керек. Бұл командалар кейін өңдеу командаларын шақырып, олардың орындалуын басқаратын басқа құрамдауыштар арқылы сипатталады.

Әрине, көптеген компиляторлар үшін аудармашылар машинаның командаларын өңдейді және дерексіз машина шын машина болып табылады. Сонда да тілдің Питон сияқты үдемелі типті аудармашылар бағдарламаның құрамдауышы бола алады.



6.19-сурет. Арна және фильтр компиляторының сәулеті

Программалау тілінің компиляторлары – жалпы программалау ортасының жалпы сәулетінің бір бөлігі (*6.19-сурет*), олардың құрамына мыналар кіреді:

1. Лексикалық өңдеуші, енгізілген таңбаларды ішкі түрге айналдырады.
2. Таңбалар кестесі, аударылатын мәліметтерді сақтайды.
3. Синтаксистік өңдеуші, енгізілген мәліметтің синтаксисін тексереді де синтаксистік шежіре құрады.
4. Синтаксистік шежіре, компилятордан өткізілетін мәліметтің синтаксисін құрады.
5. Семантикалық өңдеуші, синтаксистік кесте мен шежіреден мәліметтерді алып, семантикалық қателерге тексереді.
6. Кодтық генератор, синтаксистік шежіре арқылы бір тексеріп шығып дерексіз машиналық кодқа аударады.

Сілтемелік құрылымдар

Сілтемелік құрылымдар егеліктегі жүйе құрылымының маңызды мүмкіндіктерін қамтиды. Іс жүзінде, олар қолданба құрылымына кіретіндердің барлығын қамтиды, ал шындығында жеке қолданбалардық сілтемелік құрылымда көрсетілген барлық мүмкіндіктерді өзіне қамтып алу аса қажетсіз болып табылады. Сілтемелік құрылымның негізгі мақсаты жоба негіздемелерін бағалау және салыстыру мен осы егеліктегі адамдарға құрылымдық сипаттамалар туралы білім беру.

http://www.SoftwareEngineering-9.com/Web/Architecture/RefArch.html

Бұлардан басқа да құрамдауыштар қосуға болады, олар синтаксистік шежірені өңдеп, тасымалдап отырса, машиналық код өңделуінің сапасын арттырады, артық мәліметті тауып өшіруге мүмкіндік береді. Басқа тілдердің типтерінде табиғи тілдердің аудармашыларындағыдай қосымша сөздік сияқты құрамдауыштар қосылуы мүмкін.Бұл аударудың сапасын арттырады.

Тіл өңдеу жүйесінде қолдануға болатын көптеген дайын сәтті сәулеттік үлгілер бар. Компиляторлар арналар мен фильтрлер арқылы жасала алады. Лексикалық, синтаксистік және симантикалық анализ жүйе жүзінде (*6.19-сурет*), ортақ таңбалар арқылы қатынасу үшін құрылған.

6.20-сурет. Тілді орындау жүйесінің деректер сәулеті

Арналар мен фильтрлер үлгісі деректер пакетімен жұмыс істегенде қолайлы. Бірақ егер компилятор басқа тілмен ықпалданса, структуралы реттеу немесе ПреттиПринтер программасын өңдегенде қолайсыз болады. Бұлардың мысалы *6.20-суретте* сипатталған.

Бұнда тіл өңдейтін жүйе программаның қолдайтын жабдықтармен интегралданып, олардың бір бөлігі бола алады. Бұл мысалда таңбалар кестесі мен синтаксистік шежіре екеуі орталық мәліметтердің қоймасы қызметін атқарады.

НЕГІЗГІ ТҰЖЫРЫМДАР

- бағдарламалық сәулет – бағдарламалық жүйенің қалай құрылғанын сипаттайды. Өнімділік, қауіпсіздік, қолжетімділік сияқты қасиеттер сәулеттің қолданылуына байланысты.
- әулеттік шешімдерге бағдарламаның типі жайлы, сәулеттік стильдерге байланысты және сәулетті құжаттандыру жайында қабылданған шешімдер жатады.
- сәулеттер бірнеше үлгілер мен перспективаларға қатысты құрыла алады.

- сәулеттік үлгілер қайта қолданылатын білімдерді сипаттап түсіндіреді.
- Әдетте, қолданылатын сәулеттік үлгілерге Үлгі-Модель-басқармасы, қомалы сәулет, тұтынушы-сервер және арналар мен фильтр жатады.
- дайын үлгілер бағдарламаның мәнін түсінуге, салыстыруға, дизайнын жетілдіруге және қолдану ортасын кеңейтуге көмектеседі.
- тіл өңдеу жүйесі енгізілген тілді бір тілден екінші тілге аударады. Олардың ішіне аудармашы мен енгізілген тілді өңдейтін машина кіреді.

ҚОСЫМША ӘДЕБИЕТТЕР

Software Architecture: Perspectives on an Emerging Discipline. This was the first book on software architecture and has a good discussion on different architectural styles. (M. Shaw and D. Garlan, Prentice-Hall, 1996.)

Software Architecture in Practice, 2nd ed. This is a practical discussion of software architectures that does not oversell the benefits of architectural design. It provides a clear business rationale explaining why architectures are important. (L. Bass, P. Clements and R. Kazman, Addison-Wesley, 2003.)

'The Golden Age of Software Architecture' This paper surveys the development of software architecture from its beginnings in the 1980s through to its current usage. There is little technical content but it is an interesting historical overview. (M. Shaw and P. Clements, *IEEE Software,* **21** (2), March–April 2006.) http://dx.doi.org/10.1109/MS.2006.58.

Handbook of Software Architecture. This is a work in progress by Grady Booch, one of the early evangelists for software architecture. He has been documenting the architectures of a range of software systems so you can see reality rather than academic abstraction. Available on the Web and intended to appear as a book. http://www.handbookofsoftwarearchitecture.com/.

ЖАТТЫҒУЛАР

6.1. Жүйені суреттегенде әрленімді әзірлеудің маңыздылығын түсіндіріңіз.

6.2. Мысалы, сіз инженер емес басқарушыға жаңа жүйенің сәулетін ұсынасыз. Ұсынуыңыздың негізгі қағидаларын жазып шығыңыз. Негізінде жүйенің сәулеті дегеніміз не екенін түсіндіру керексіз.

6.3. Қолжетімділігі және қауіпсіздігі маңызды жүйелердің сәулеттерінің қақтығыстарын суреттеңіз.

6.4. Төменде көрсетілген жүйелердің әрленімдерін салыңыз:

- Вокзалда пайдаланылатын автоматтандырылған билет сату жүйесі
- Бейнені бір уақытта бірнеше адамға қолжетімді ететін компьютерлік жүйе.
- Жазық аулаларды тазалайтын роботтың бағдарламалық жүйесі

6.5. Үлкен жүйелердің сәулетін анықтаған кезде, әрленімің әртүрлі үлгілерін неліктен қолданатынын түсіндіріңіз.

6.6. Ғаламторға аудио деректерді жүктейтін жүйенің әрленімін салыңыз.

6.7. JAVA сияқты бағдарламалау тілдерінің мамандыра ұсынған IDE-лерді салыстыру кезінде, CASE қоршаған ортасы қалай пайдаланады?

6.8. Қарапайым тілді командаларды қабылдайтын тілді өңдеу жүйесінің әрленімін салыңыз.

6.9. Ақпараттық жүйенің негізгі үлгісін пайдалана отырып, белгілі бір аэропорттан ұшу және келу бағыттарын суреттейтін әрленімді салыңыз.

6.10. Сіздің ойыңызша, бағдарлама сәулетінің әрленімін даярлайтын арнайы маман керек пе? Әлде сол жұмыспен айналысатын арнайы компаниялар қажет пе? Жауабыңызды толық түсіндіріңіз.

ПАЙДАЛАНЫЛҒАН ӘДЕБИЕТТЕР

Bass, L., Clements, P. and Kazman, R. (2003). Software Architecture in Practice, 2nd ed. Boston: Addison-Wesley.

Berczuk, S. P. and Appleton, B. (2002). Software Configuration Management Patterns: Effective Teamwork, Practical Integration. Boston: Addison-Wesley.

Booch, G. (2009). 'Handbook of software architecture'. Web publication. http://www.handbookofsoftwarearchitecture.com/.

Bosch, J. (2000). Design and Use of Software Architectures. Harlow, UK: Addison-Wesley.

Buschmann, F., Henney, K. and Schmidt, D. C. (2007a). Pattern-oriented Software Architecture Volume 4: A Pattern Language for Distributed Computing. New York: John Wiley & Sons.

Buschmann, F., Henney, K. and Schmidt, D. C. (2007b). Pattern-oriented Software Architecture Volume 5: On Patterns and Pattern Languages. New York: John Wiley & Sons.

Buschmann, F., Meunier, R., Rohnert, H. and Sommerlad, P. (1996). Pattern-oriented Software Architecture Volume 1: A System of Patterns. New York: John Wiley & Sons.

Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R. and Stafford, J. (2002). Documenting Software Architectures: Views and Beyond. Boston: Addison-Wesley.

Coplien, J. H. and Harrison, N. B. (2004). Organizational Patterns of Agile Software Development. Englewood Cliffs, NJ: Prentice Hall.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Reading, Mass.: Addison-Wesley.

Garlan, D. and Shaw, M. (1993). 'An introduction to software architecture'. Advances in Software Engineering and Knowledge Engineering, 1 1–39.

Harold, E. R. and Means, W. S. (2002). XML in a Nutshell. Sebastopol. Calif.: O'Reilly.

Hofmeister, C., Nord, R. and Soni, D. (2000). Applied Software Architecture. Boston: Addison-Wesley.

Hunter, D., Rafter, J., Fawcett, J. and Van Der Vlist, E. (2007). Beginning XML, 4th ed. Indianapolis, Ind.: Wrox Press.

Kircher, M. and Jain, P. (2004). Pattern-Oriented Software Architecture Volume 3: Patterns for Resource Management. New York: John Wiley & Sons.

Krutchen, P. (1995). 'The 4+1 view model of software architecture'. IEEE Software, 12 (6), 42–50.

Lange, C. F. J., Chaudron, M. R. V. and Muskens, J. (2006). 'UML software description and architecture description'. IEEE Software, 23 (2), 40–6.

Lewis, P. M., Bernstein, A. J. and Kifer, M. (2003). Databases and Transaction Processing: An Application-oriented Approach. Boston: Addison-Wesley.

Martin, D. and Sommerville, I. (2004). 'Patterns of interaction: Linking ethnomethodology and design'. ACM Trans. on Computer-Human Interaction, 11 (1), 59–89.

Nii, H. P. (1986). 'Blackboard systems, parts 1 and 2'. AI Magazine, 7 (3 and 4), 38–53 and 62–9.

Schmidt, D., Stal, M., Rohnert, H. and Buschmann, F. (2000). Pattern-Oriented Software Architecture Volume 2: Patterns for Concurrent and Networked Objects. New York: John Wiley & Sons.

Shaw, M. and Garlan, D. (1996). Software Architecture: Perspectives on an Emerging Discipline. Englewood Cliffs, NJ: Prentice Hall.

Usability group. (1998). 'Usability patterns'. Web publication. http://www.it.bton.ac.uk/cil/usability/patterns/.

7.

Өңдеу және іске асыру

Мақсаттары

Тараудың мақсатына объектілі-бағытталған жүйелердің UML қолданылуын зерттеу және маңызды қиындықтарды жүзеге асырулығын көрсету жатады. Осы тарауды оқыған соң сіз:

- Объектілі-бағытталған жүйелердің ең маңызды әрекеттерін;
- Объектілі-бағытталған жүйелердегі әр түрлі модельдердің қолданылуын;
- Зерттеу үлгілерімен қатар, екінші қолданыс тәсілдері мен тәжірибелерін;
- Басты нәтижелерін анықтау және бағдарламалық жасақтаманың орындалуын, соның ішінде бағдарламалық жасақтаманың екінші қолданылуымен ашық код зерттемесін жүргізілуін анықтайсыз.

Мазмұны

Бағдарламалық жабдықтаудың зерттеуі мен орындалуы оның өңдеу кезеңіндегі үдерісі болып табылады және ол арқылы бағдарламалық жабдықтауды орындайды. Кейбір қарапайым жүйелермен бағдарламалық жабдықтауды және басқада әрекеттерді біріктіреді. Алайда, ірі жүйелерге бағдарламалық жабдықтаудың зерттемесімен орындалуы біршама үдерістердің қатарына жатады.

Бағдарламалық жабдықтаудың зерттемесі және орындалуы өзара алмасады. Бағдарламалық жабдықтауды зерттеу бағдарламаның құрамдас бөлігі және олардың қатынасы болып табылатын шығармашылық әрекетке жатады. Орындалуына ұсынған барысында бағдарлама түрінде жүзеге асады. Кейде бөлек кезеңі үшін зерттеменің үлгілеуі және құжаттауы жүргізіледі. Кей уақытта зерттеме бағдарламашының ойында немесе қағаз күйінде болады. Зерттеме мәселесінің шешімі зерттеу барысында жүзеге асады. Сонда да, әрқашан қажеттілігі тумайды, UML немесе басқада моделдерді бейнелеу арқылы қолданылуының жөнділігі сипатталады.

Зерттеме мен орындалу өзара тығыз байланысты және зерттеме кезінде жүзеге асады. Мысалы, егер сіз объектілі-бағытталған тілдерде (C#, Java) бағдарлама жасасаңыз жүйелерді құжаттау үшін UML қолданасыз. Ал егер Python секілді серпінді типтік тілде және жүйені пішін үйлесімі арқылы орындағаныңыздың мағынасы болмайды. *3-бөлімде* айтып кеткендей, әдетте, икемді әдістер бейресми зерттеу кескінінде жұмыс істейді және зерттеушілерге қорытындылау үшін беріледі.

Бағдарламалық жасақтаманың бастапқы кезеңінде қолданбалы бағдарламалық жасақтаманы жасау маңызды шешім болып табылады. Қазіргі уақытта әртүрлі саладағы сатылымға түскен пайдаланушыларға ыңғайлы (COTS) жүйесін алуға болады. Мысалы, медицина жүйесін жүзеге асырғыңыз келсе, емханаларда пайданатын әдістерді сатып аласыз. Бұл үнемдеуге көмегін тигізеді және уақытыңыз бағдарламаны жазуға кетпейді.

Осындай жолмен қосымшаны орындасаңыз, түрлі икемдеу функцияларын қалай қолданылуын білмегендіктен кескіндеу үдерісі уайымға түседі. Негізі, жүйенің кескін зерттемесі дамымайды, ол туралы *16-бөлімде* COTS жүйелері туралы айтылып кетеді.

Менің болжауым бойынша, оқырмандардың көбісінде зерттеме мен оның орындалуының тәжірибелері бар. Java немесе Python тілін меңгеріп қана қоймай, бұл тілдерде бағдарлама жазуды да білетін боласыз. Сіздер мүмкін түрлі бағдарлама жасау тілдерін білетін шығарсыз және осы бағдарламаларды міндетіп игердіңіз. Сол себептен мұнда мен бағдарламалық тақырыптарды қамтыған жоқпын. Оның орнына бұл тараудың басты екі мақсаты:

1. Жүйенің қалай үлгілеуін құрылымдық кескінімен көрсету (*5-* және *6-тарауда* қамтылған) дамытпалы объектілі-бағытталған жүйелерде тәжірибе жүзінде қолданылады.
2. Маңызды шешімді жүзеге асыру үшін біркелкі бағдарламалау кітаптарда қамтылмаған. Оларға бағдарламалық жабдықтауға қайта қолданылу, кескін пішіні және бастапқы код зерттемесі жатады.

Құрылған жоба әдістері

Құрылған жоба әдістері бағдарламалық жасақтама жобасының әдістемелік жолмен жасақталуын ұсынады. Жүйені жобалау әрекеті әдістің келесі қадамдарын және толық дерекпен қамтамасыз етудің үдемелі деңгейіндегі жүйе жобасының нақтылануын қамтиды. 1990-жылдары, нысанға бағытталған жобаға арналған бірнеше бәсекелес әдіс болған еді. Алайда, жалпыға кеңінен қолданылатын әдістердің өнертапқыштары біріге отырып, түрлі әдістерде қолданылатын шартты белгілер жүйесін бірегейлендіруші UML тілін жасақтап шығарды. Әдістерге бағытталған назарға қарағанда, қазіргі таңда бағдарламалық жасақтамасын өңдеу үдерісінің бір бөлігі ретінде байқалатын жобаның үдерістер туралы тақырыбы кеңінен талқылану үстінде. Рационалды біріңғайланған үдеріс (RUP) біртекті өңдеу үдерісіне жақсы мысал бола алады.

http://www.SoftwareEngineering-9.com/Web/Structured-methods/

Түрлі көп мөлшерлі зерттеме платформалары бар болғанымен, бұл тарау нақты бағдарламалау тіліне немесе зерттеме технологиясына бағытталмаған. Бұл әдіспен Java немесе Python тілдерінде емес, олардың орнына UML қолданылған мысалдарды ұсындым.

7.1. Нысанға-бағытталған жүйелердің UML қолданылуы

Нысанға бағытталған жүйе өзара әрекеттесетін нысандардан тұрады. Мемлекеттік көрсетілімі тек жекеше болады және нысаннан сырттай қолайсыз. Нысанға-бағытталған жүйе нысан топтарын және олардың өзара қарым-қатынасын жобалау барысын да қамтиды. Бұл топтар нысандарды жүйеде анықтайды. Зерттеме бағдарлама орындайтын күйінде жүргізілгенде, объектілер анықталған топтардан өсіңкілік тудырады.

Функционалдық жақындау қолданылудың өңделуіне қарағанда нысанға-бағытталған жүйе оңай өзгертеледі. Нысандарға мағлұматтар және олардың меңгерілуі жатады. Сол себептен олар өзара ұғымды да, дербес субъект түрінде өзгереді. Жүзеге асқан объектінің өзгеруі немесе қызметтің қосылуы басқа жүйелердің объектілеріне ықпал жасамау керек. Нысанның жүзеге асу өзгерісі немесе қосылу қызметі басқа объект жүйелеріне әсерін тигізбеу керек. Нысандар құбылыстармен байланысты болғандықтан, нақты объектлермен бақылау жүйелеріндегі нысандар арасындағы сәйкестілік бар. Ұғымдылық жақсартылады, демек, жөндеуге жарамдылығы артады.

Жүйені жетілдіру үшін тұжырымдамадан толық түрінде талдауға бару үшін объектілі-бағытталған жүйеде:

1. Мәнмәтінді және жүйемен сыртқы өзара қатынасты түсініп қана қоймай, анықтау;
2. Жүйенің архитектурасын жобалау;
3. Жүйенің басты объекттерін шығару;
4. Әрленім үлгісін өңдеу;
5. Интерфэйс нұсқаулығын әзірлеу сияқты бірнеше қадамнан өту керек.

Басқа әрекеттер сияқты жобалау айқын емес, жүйелеу үдерісі болып келеды. Сіз ойланып жобалау құрдыңыз, шешімдер ұсынып және мәліметтер келген бетте шешімдерді айқындадыңыз. Сізге сөзсіз кері қайтуға тура келеді де, мәселені шешу үшін қайтадан талаптарды орындау керек. Егер жұмыс істесе қарастырып, кейде бұлжытпай нұсқаларды үйрену керек, басқа жағдайда үдерістің соңын күтіп елемеу керек. Демек, саналы түрде үдерісті көрсетпей, қарапайым сұлба арқылы көрсетуге болады, себебі, кескінді жүйелі әрекет ретінде қарастыруын білдіреді. Шынында, төбеде айтылған шаралар өзара алмасады, сондай әсері бар. *1-тарауда* айтып кеткендей, мен әрекет барысының метеобекетке арналған бағдарлама қамтамасыздандыруды өңдеу арқылы көркемдедім. Осы метеобекеттер алыстағы аудандарда орналасқан. Әр метеобекеттің ауарайы туралы мәліметі жазылады және уақыт сайын оның жүйесіне ауарайы туралы мәліметтерді спутниктік байланыс арқылы жібереді.

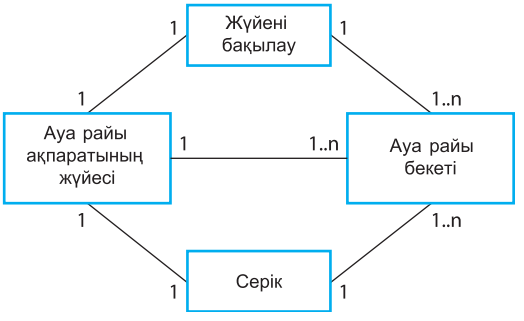
7.1.1. Мәнмәтіндік жүйе және қарым-қатынасы

Кез келген бағдарлама қамтамасыздандыру барысының алғаш кезеңінде оның арасындағы қарым-қатынасының түсінушілігінің өркендеуін дамытады. Жүйенің қажетті функционалдік қамтамасыздандыру үшін шешім қабылдауда және жүйенің қоршаған ортамен қатынасының құрылымы үшін маңызды мәні бар. Мәнмәтінді түсіну жүйенің шегін орнатуға көмектеседі.

Жүйенің шегін орнату көптеген мәселелерді шешуге бейімделген, мысалы, қандай өңделетін функциялар жүйеде жүзеге асады, қандай функциялардың басқа жүйелермен қатынасы бар. Бұл жағдайда, барлық метеобекеттер үшін функционалдік басқару жүйесіне және онымен бірге кіріктірме бағдарламалық қамтамасыздандыруға таратылады.

Жүйенің мәнмәтін үлгісі және арақатынастың үлгісі жүйенің және қоршаған ортаның арасындағы мынадай түрлі қосалқы қатынастарды көрсетеді:

1. Үлгілі жүйенің байланысы жүйенің өңдейтін ортада басқа жүйелермен көрсетілеуі арқылы құрылымдық нысаны болып саналады.
2. Үлгінің қатынасы жүйенің қалай қоршаған ортамен байланысын және оның қолданылуын серпінді күйде көрсетеді.



7.1-сурет. Ауа райы бекетінің жүйелік мәтіні (контексі)

Үлгінің мәнмәтіні байланыс арқылы келтіріледі. Байланыс қатынастардың мәндерінің арасындағы байланысты білдіреді. Қатынастың сипаты анықталады. Жүйенің ортасына қарай қарапайым топтама-тәсім ретінде қатынаспен олардың арасындағы байланыс көрсетіледі. Метеобекет жүйесінде *7.1-суретінде* бейнелегендей, жүйенің ортасында ауарайы туралы түрлі ақпараттар бар.

Жүйемен оның ортасы арасындағы қатынасты үлгілеу үшін абстракт амалын қолдануға болады. Оның бір тәсілі өнеге үлгісін қолданылуы. *4-* және *5-тарауда* айтып кеткендей, әр нұсқаның қолданылуы жүйенің өзара қатынасын көрсетеді. Әр мүмкін болатын қатынас элипс тәрізді және қатынасқа қатысатын сыртқы мәні пішінделеді.

7.2-сурет. Ауа райы бекетінің пайдалануы

7.2-суретте бейнеленгендей, метеобекет жүйесіне өнеге үлгісі көрсетілген. Бұл үлгі метеобекет жүйесіндегі ауа райы туралы мәліметтермен оның қатынасын білдіреді. Басқа басқару жүйесімен қатынастар нақты командалар бекет ауа райының басқаруы үшін шығару мүмкін. *5-тарауда* айтып кеткендей, UML-ді басқа жүйелермен қатынасын көрсету үшін қолданады. Әр қолданылатын нұсқа құрылымдық тілде бейнелену керек. Бұл әрлеушілерге объектілердің жүйедегі орнымен олардың түсінушілігін арттырады. Мен қалыпты форматты қолдануын шештім, себебі мәліметтің араласуы қалай өтетінін көру керек. *7.3-суретте* ауа райының *7.2-суретіндегідей* қолданылуы көрсетілген. Біздің сайтымыздан басқа жағдайдағы қолданылуын көре аласыз.

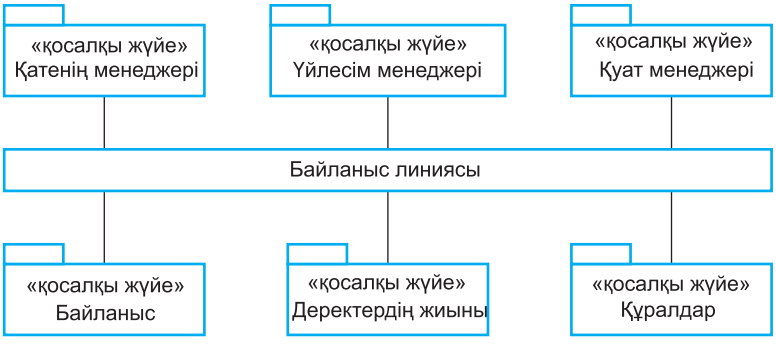
Жүйе	Ауа райы бекеті
Пайдалану	Ауа райы есебі
Кейіпкерлер	Ауа райы ақпараттар жүйесі, ауа райы бекеті
Dat	Ауа райы бекетіне жиналған ақпараттардың қорытындысы қажет. Сақталатын ақпарат жердің және ауаның ең төмен, ең жоғарғы және орташа температуралардан құралады.
Ынта	Ауа райының ақпараттық жүйесі ауа райы бекетінің сырттай жүйелермен қатынасуын қамтамасыз етеді және деректердің тасымалдануына жауап береді.
Жауап	Қорытындылған ақпараттар ауа райы бекетінің бағдарламасына жіберіледі.
Түсінік	Ауа райы бекеттері сағат сайын есеп беруге мәжбүрленеді. Бірақ бұл жиілік әр бекетте әртүрлі болуы мүмкін.

7.3-сурет. Ауа райының пайдалану есебі

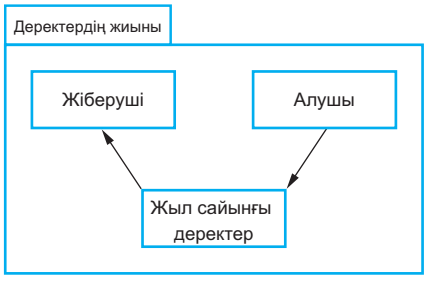
7.1.2. Құрылымдық жобалау

Жүйенің және бағдарламалық қамтамасыздандыру жүйесінің қоршаған ортасы арасындағы қатынас анықталады, бұл ақпаратты жүйенің құрылымдық зерттеме негіздері ретінде қолдануға болады. Ол үшін, әрине, өзіңіздің құрылымдық жобалау қағидаттар туралы біліміңіз арқылы біріктіру қажет. Сіз жүйені құрайтын басты компонентермен олардың қатынастарын анықтап, сосын құрылымдық үлгіні қолданатын клиент-сервер немесе қабатты үлгілер секілді компоненттерді құрай аласыз.

Метеобекет үшін бағдарламалық қамтамасыздандырудың құрылымдық жобалаудың жоғары деңгейі *7.4-суретінде* бейнеленген. Бұл метеобекет кеңжүйелік хабарлар жалпы инфрақұрылымға қатынасатын *7.4-суретіндегі* байланыс желісі ретінде көрсетілгендей тәуелсіз қосалқы жүйелерден тұрады. Әр қосалқы жүйе хабарды инфрақұрылымға жіберіп, оларға бейімделген хабарларды таңдап алады. *6-тарауда* айтылып кеткендей, осы құрылымдық жобалауға қосымша болады.



7.4-сурет. Ауа райы бекетінің жоғарғы денгейдегі сәулеті



7.5-сурет. Деректерді жинайтын жүйенің сәулеті

Мысалы, қосалқы басқару жүйесін алған кезде, дұрыс жолмен баратын басқа әрбір қосалқы жүйелер жиналады. Басты құрылым түрлі қосалқы жүйелердің кескін үйлесімін қолдайды, себебі хабар жіберуші нақты қосалқы жүйесіне хабарын жібермесе де болады. *7.5-суретте* қосалқы жүйелердің құрылымдық *7.4-суретінде* бейімделгендей деректер жиыны көрсетілген. Хабарлағыш пен қабылдағыш объектлері басқару жүйесімен байланысқан және WeatherData объектісін ақпаратты инкапсуляция барысында өткізеді, ол құралдардан жинақталады да, ауа райы туралы ақпаратты жібереді.

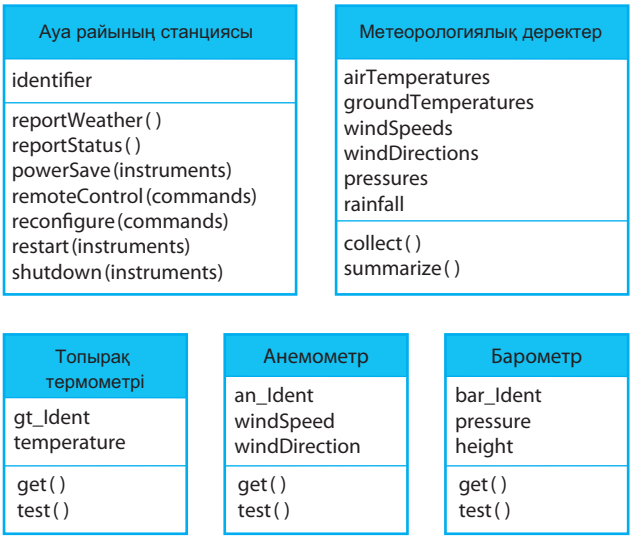
7.1.3. Объект топтамасын сәйкестендіру

Осы зерттеме кезеңінде сізде біршама жүйеге қажетті объектілер туралы көрсетілім болуы керек. Сіздің ойыңыз арқылы әрленім әрі қарай дамиды, сіздің көрсетіліміңіздің көмегі тиеді. Нұсқаның қолдану сипаттамасы объектілерді сәйкестендіруге және жүйенің түрлі операцияларына көмектеседі. Weather Report нұсқаның қолдану сипаттамасынан объектілер, ауа райы туралы мәліметтерді жинайтын жабдықтар көрсетілімі көрінеді. Сіз тағы сәйкестендіру жүйесінің қатынасын анықтайтын жоғары деңгейдегі жүйені немесе объектіні қажет етесіз. Бұл объектілермен жүйенің объект топтамаларына жіктеуді бастай аласыз.

Объектілі-бағытталған жүйедегі объект топтамасын анықтайтын бірнеше ұсыныстар бар:

1. Табиғи тілдегі жүйенің анықтайтын грамматикалық талдаудағы қолданысын жасау. Объектімен атрибуттар зат есім, операциямен қызметі етістік болады (Abbott, 1983).
2. Жалпы есімдерді қолданыңыз, мысалы, ұшақ, тікұшақ, мәндерді қолданыңыз, басқарушы немесе дәрігер, оқиғалар, мысалы, сұраныстар, арақатынастар, мысалы, кездесулер, орындар, мысалы, офис немесе компаниялар және т.б. (Coad және Йордан, 1990; Shlaer және Меллор, 1988; Wirfs-Брок т.б., 1990).
3. Талдау негізінде сценарий қолданыңыз, түрлі қолданылатын сценарийлер жүйеде анықталады да, өз алдына талданылады. Әр сценарий талданады, талдау атқарушы қажетті объектілерді, атрибуттарды және операцияларды анықтау керек (Бек және Cunningham, 1989).

Объект топтамаларын анықтау үшін сіз бірнеше дерекнамаларды қолдануыңыз керек. Білімнің қолданылуы қосымша ақпарат болады. Бұл ақпарат талап етілген құжаттардан, пайдаланушымен сұхбаттан немесе бар жүйені талдау нәтижесінен жиналады.



7.6-сурет. Ауа райы бекетінің нысандары

Метеобекет шөлдаласында объектінің сәйкестендіруі шынайы сезінетін жүйенің құралдарынан негізделген. Жүйенің барлық объектілерін қосатын орын жоқ, сол себептен *7.6-суретінде* бес объект топтамасын көрсеттім. Бірінші термометр, анемометр, барометр және объектілер заттар саласына жатады, тағы ме-

теобекет пен WeatherData объектілері жүйенің баяндалуынан және сценарийден анықталған:

1. WeatherStation объект топтамасы қоршаған орта мен метеобекет жүйесінен негізгі интерфейсті құрайды. Оның операциялары *7.3-суретінде* көрсетілген қатынастарды бейнелейді. Бұл жағдайда, барлық қатынастардың сәйкестендіру үшін мен тек бір ғана топтаманы қолданамын, бірақ басқа құрылым арқылы бірнеше түрлі топтама секілді сіз жүйенің интефейсін өңдей алатын едіңіз.
2. WeatherData топтама объектісі ауа райы болжауының команда өңдеумен үйлеседі. Ол метеобекет құрылғылардан ауа райы туралы мағлұмат жүйесіне жалпылама ақпараттарды жібереді.
3. Бірінші термометр, анемометр, барометр, объектілер топтамасы жүйенің құжаттарымен байланысады. Олар жүйенің құралдары арқылы елеулі жүздермен бұл жабдықтаудың басқаруымен байланысты объектілерді қамтиды. Бұл объектілер берілген жиілікте деректерды жинау үшін және жиналған мәліметтерді сақтау үшін автономдық түрде жұмыс істейді. Деректер WeatherData объектісіне сауал арқылы жіберілген.

Сіз заттар саласындағы біліміңізді басқа объектілерді, аттрибуттарды анықтау үшін қолдана аласыз. Метеобекеттер көбінесе алыста орналасқанын білеміз, олар түрлі құралдарды өздеріне енгізеді. Автоматтық түрде құралдар өздерінің сәтсіздіктерін хабар етеді. Сізге құралдардың дұрыс жұмыс істейтінін көрсететін атрибуттардың қажеттілігін білдіреді. Алыстағы метеобекеттер өте көп, сол себептен, әр метеобекеттің өзінің сәйкестендіргіші болуы керек.

Бұл жобалау үдерісінде сіз объектілердің жүзеге асқанын ойламай, объектілердің өзіне көңіл бөлуіңіз керек. Объектілерді анықтап алғаннан кейін, оның әрленімін ойластыру керек. Сіз жалпы белгілерін тауып алып, сосын мұра нысандарын құру керексіз. Мысалы, сіз сәйкестендіргіш секілді суперкласс құралдарын анықтап, әрекеттерді тестілеп аласыз. Тағы деректер жинайтын жиілігін ұстанатын атрибуттар секілді жаңа атрибуттарды және әрекеттерді суперкласқа қоса аласыз.

7.1.4. Әрленім үлгісі

Жүйе немесе әрленім үлгілері, мен *5-тарауда* талқылағандай, объектілер немесе класстарды жүйеде көрсеттім. Олар тағы объектілер мен класстар арасындағы қатынас пен арақатынасты көрсетеді. Бұл үлгілер жүйелік талаптар мен жүйенің енгізуіне көмектеседі. Олар жалпылама болып келеді, сол себептен жүйенің кейбір бөлшектері жүйенің талаптары мен арақатынастарды жасырмайды. Сонымен қатар олар бағдарламашыға жеткілікті қорытындыны бұлжытпай жүзеге асыру керек. Ереже бойынша, үлгіні өңдеу арқылы түрлі деңгейлерде мұндай шиеленістерді атап кету керек. Талаптар мен өңдеушілер, әрленушілер мен бағдарламашылардың

арасындағы қатынас бар болған кезде, барлығы жалпылама үлгілер болып табылады. Нақты жобаның қорытындысы жүйенің орындалуы секілді болып, мәселелер бейресми түрде шешіледі. Жүйенің спецификаторлар арасындағы қатынас әрленушілер мен бағдарламашылардың жанамасы болып келеді.

Жобалау үдерісінің маңызды қадамы қажеті бар талдап тексеретін үлгіде зерттеме үлгісінің шешімі болып саналады. Жүйенің түріне байланысты, қазіргі таңда осы үдеріс өңделеді. Зерттеменің ретті жүйесі деректерді өңдеуі болады, басқаша кіріктірме жүйенің шынайы уақытта орындалғаннан сізге түрлі әрленім үлгілері керек болады. UML 13 әртүрлі әрленім үлгілерін қолдайды, бірақ *5-тарауда* айтылып кеткендей, сіз оны сирек қолданасыз. Үлгі мөлшерінің барынша азаюы өңдеу мен уақыттың шығарымдарына кететін жобалау үдерісінің соңында қажет болады.

UML қолданатын әрленімді өңдеу үшін сіз екі түрлі әрленім үлгісін тарқатадыз:

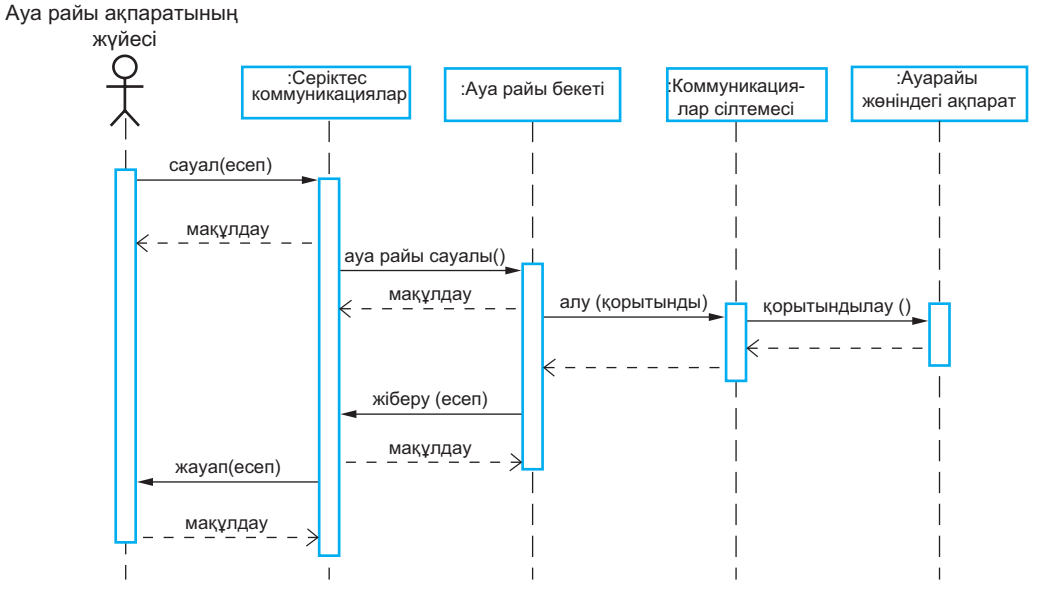
1. Құрылымдық үлгілер, қозғалмайтын жүйенің құрылымы объекттің топтарын және олардың арасындағы қатынасты қолданады. Берілген қадамдағы қатынастар маңызды түрде талдап қорытылып құжатталады және қатынастар арқылы қолданылады.
2. Серпінді үлгілер, жүйенің серпінді құрылымы арқылы бейнеленіп, жүйенің объектілер арасындағы қатынасты көрсетеді. Құжатталған қатынастар жүйелілік сауалдар қызметі өзіне кіреді.

Жобалау үдерісінің алғашқы қадамдарында, менің ойымша, бөлшек қосындысының үш үлгісі бар:

1. Қосалқы үлгілер қисындық объект топтарымен көрсететіледі. Топ кестесі бойынша әр қосалқы жүйелер жабық объектілер ретінде қолданылады.
2. Үлгінің жүйелілігі объектілер қатынасының жүйелілігін көрсетеді. Жүйелілікті қолданып, UML диаграмма ретінде ұсынылады. Қарқынды үлгілер жүйелілік үлгісі болып табылады.
3. Жалпы машина үлгісі бөлек объектілердің өзгеру күйін көрсетеді. Жалпы машина үлгісі қарқынды үлгі болып табылады.

Қосалқы жүйе статикалық үлгі болып, әрленім логикалық объект топтарымен байланысты екенін көрсетеді. *7.4-суретінде* бұл үлгі түрін бейнелеп, ауа райы бейнесі қосалқы жүйеде байқалады. Қосалқы жүйенің үлгісі секілді жүйедегі барлық объектілермен олардың қосындысын көрсетіп, объектінің бөлшек үлгілерін өңдей аласыз. Сонымен қатар, көп үлгілеудің қаупі бар. Мысалы, жүзеге асыратын тура шешімді көрсетпей қалуы.

Серпінді үлгі жүйелік үлгіге әрбір тәртіптемеге қатынасы болып, объект арасындағы жүйелілік орын алып бейнеленеді. Әрленімді құжаттаған кезде, әр маңызды қатынас үшін үлгі жүйелілігін жасау керек. Егер сіз прецендент үлгісін өңдесеңіз, әрбір қолданған сайын анықталған үлгінің жүйелілігі болмау керек.

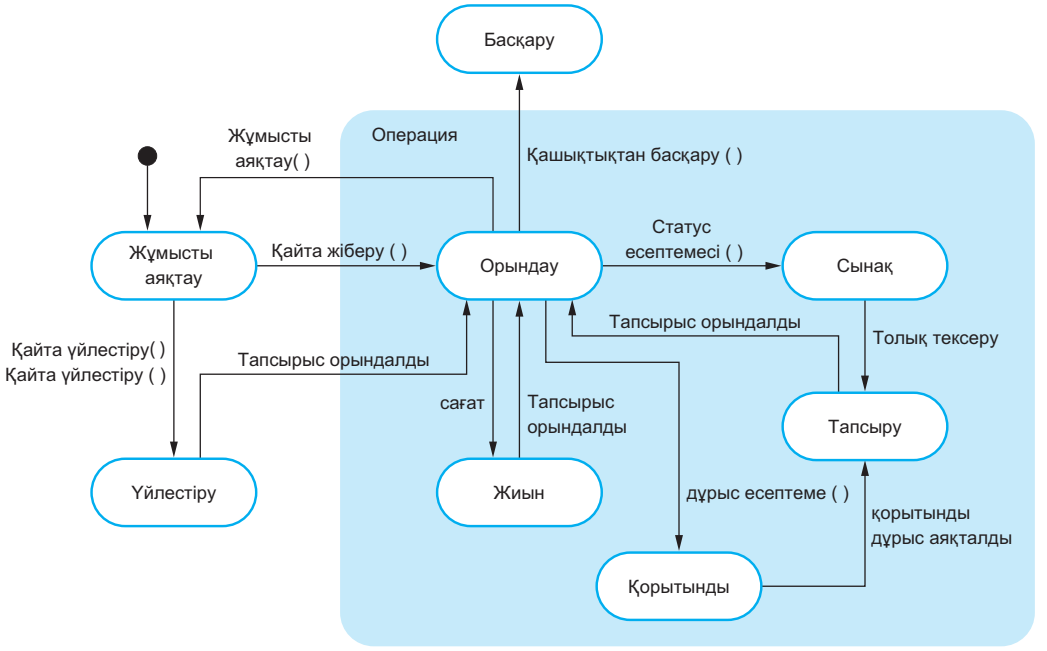


7.7-сурет. Деректер жиналуын суреттейтін тізбекті кескін

7.7-суретте UML жүйелілігі арқылы үлгінің жүйелілігі сызба түрінде келтірілген. Бұл кесте сыртқы сауалдармен метеобекеттің жалпылама деректер арақатынас жүйелілігін көрсетеді. Төбеден астыға қарай жүйеліліктің сызбасын оқисыз:

1.1. SatComms объектісі сауалнаманы жүйеден алып, метеобекеттен ауа райы туралы ақпаратты жинайды. Бұл осы сауалнаманы қабылдағанын мәлімдейді. Таяқша тілі жіберілген хабар мен сыртқы жүйе жауапты күтпейтінін білдіреді, бірақ басқаша түрде өңделеді.
1.2. SatComms WeatherStation-ға жалпылама жиналған деректер үшін хабарлама серік арқылы жібереді. Тағы да, таяқша тілі SatComms өзін тоқтатпайтынын көрсетеді.
1.3. WeatherStation Commslink объектісіне ауа райы туралы жалпы мәліметті хабарлама ретінде жібереді. Кейін жауабын күтеді.
1.4. Commslink WeatherData объектісінің қорытынды тәсілін атап, жауабын күтеді.
1.5. Commslink объектісі арқылы WeatherStation қайта оралып, ауа райы шолуын деректерде санайды.
1.6. SatComms объектісі WeatherStation кейін ауа райы туралы ақпараттық жүйеде спутник байланысы арқылы жақын деректер үшін шақырылады.

SatComms объекті мен WeatherStation объектісінің жасалған жұмысы тоқтатылуы және жаңартылуы қос үдеріс арқылы өңделеді. Мысалы SatComms объектісі сыртқы жүйеден хабарды алып, метеобекетке бұл хабарламаларды код-сыздандырады.



7.8-сурет. Ауа райы бекетінің тұрақты кескіні

Жүйелілік сызбасы құрамдастырылған объект топтарын қолданып, объектінің тәртібін жалпылауларыңызға немесе хабарламаға жауап ретінде қосалқы жүйеде болады. Соңғы автомат үлгісін қолданып, объект данасының күйінің өзгерісін көрсетеді. (1987) жылы алғаш Харель ұсынған машина үлгісінің түрі UML сызбасында көрсетіледі.

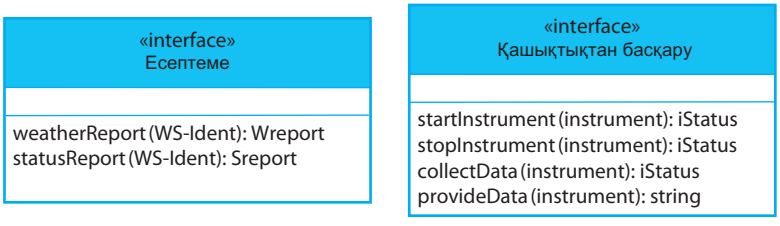
7.8-суретте метеобекет жүйесіне арналған сызба түрлі қызметтердің көрсетіліміне қалай әрекеттесуін бейнеленеді.

7.1.5. Интерфэйсты сипаттайтын құжат

Әрленімдегі құрамдас бөлшектер арасындағы интерфэйсті сипаттайтын құжат кез келген жобалау үдерісінде маңызды болып табылады. Интерфэйсті көрсету қажет, себебі объектілермен қосалқы жүйе қатарлас өңделуі мүмкін. Интерфэйсті анықтап алғаннан кейін, басқа объектілерді өңдеп, интерфэйс жүзеге асты деп ойлауға болады.

Интерфэйстің әрленімі объектіге арналған бөлшекке немесе объект топтарына байланысты. Объектімен немесе объект тобымен берілген қызметтің семантикасын және сигнатурасын анықтауды білдіреді. Класс диаграммалары арқылы UML қолданылған түрде интерфэйс берілуі мүмкін. Сонымен қатар UML берілгенінде ешқандай атрибут жоқ, интерфэйс бір бөлігінің атауына жату керек. Object Constraint (OCL) тілі арқылы интерфэйстің семантикасы анықталуы әбден мүмкін.

Бағдарламалық қамтамасыздандырудың компоненттік өңделуін *17-тарауда* талдап беремін. UMLде интерфэйстің баламалы тәсілін көрсетемін.



7.9-сурет. Ауа райы бекетінің интерфейстері

Интерфэйс әрленіміне деректер көрсетілімі туралы мәліметтерді қоспау керек, өйткені интерфэйсті сипаттайтын құжатта атрибуттар анықталмаған. Сонымен, сіз мүмкіндік үшін және деректердің жаңартылуы үшін операцияларды қосуыңыз керек. Деректер көрсетілімі жабық болғаннан, бұл деректер қолданылған объектілер оңай өзгертіледі. Өз бетімен оңай қызмет ету әрленімге әкеліп соқтырады. Мысалы, массивтегі стэк көрсетілімі тізім көрсетіліміне ауысуы мүмкін, бірақ стэк қолданатын басқа объектілерді тиіспейді. Басқа жағынан, статикалық өңдеу үлгісінде атрибутты көрсетудің көбінесе, зор мәні бар. Бұл объектілердің негізгі сипаттамасының тұтас тәсілі болып табылады.

7.2 Жобалаудың үлгілері

Жобалаудың үлгілері, Александр Кристофер (1997) ұсынған ойлардан алынды. Ол жобалаудың тиімді және жағымды істері болғанын болжады. Мәселенің сипаттама көрінісі және оның шешімінің мәні ерітіндіні әртүрлі шарттарда қайтадан пайдалануға болатынын көрсетті. Көрініс толық спецификация емес. Дегенмен, сіз оны жиналған даналықпен тәжірибені ортақ мәселенің сенімді шешімі деп ойлай аласыз.

Hillside (http://hillside.net) веб-группасының, инкапсуляциялаған үлгілер туралы ақпарат сақтау үшін пайдалануға арналған дәйексөзі:

> *Үлгілер және үлгі тілдері жақсы тәжірибелерді сипаттаудың тәсілі, жақсы әрлендіру, сонымен бірге мүмкін басқалар үшін бұл тәжірибені қайтадан пайдалануы болып табылады.*

Үлгілер бағдарламалық жасақтамаларға зор әсер етті. Тысқары ортақ мәселелер сыннан өткеннен кейін, олар әрлендірудің талқылауы үшін сөздік болды. Сіз пайдаланатын өзініздің әрлендіруіңізді үлгінің сипаттамасымен түсіндіре аласыз. Бұл әсіресе, бастапқыда суреттелген жобалаудың белгілі үлгілерінің еңдері үшін дұрыс (Гамма, 1995). Үлгінің сипаттамаларының басқа маңызды ерекшеліктері Siemens авторларының кітап топтамасында, технологияның ірі еуропалық компаниясында

шығарылған (Бушман, 1996; Бушман, 2007a; Бушман, 2007b; Кирхер және Джейн, 2004; Шмидт, 2000).

Үлгінің әрлендіруі нысанға-бағытталған жобалауға байланған. Жарияланған объектінің сипаттамасында мұрагерлік полиморфизммен қамтамасыз етіледі. Дегенмен, үлгінің инкапсуляция тәжірибесінің ортақ қағидаты бағдарламалық жасақтаманың кезкелген түрінде қолданылатын тең дәреже болуын көрсетеді. Ендеше, сіз балалар кереуеттерінің жүйесі үшін үлгінің теңселімін алар едіңіз. Үлгі білімдердің қайтадан пайдалануының тәсілі болып табылады.

Болған істің үлгісін әрлендірудің төрт негізгі элементтері анықталынған. Олар:

1. Үлгіге мәнді сілтеме болып көрінетін атау.
2. Суретті қолдану мүмкіндігін ұғындыратын мәселе облысының сипаттамасы.
3. Қарым-қатынастың және функцияның жобалық шешімінің бөліктері сипаттамасының шешімі. Бұл құрылымның нақты емес сипаттамасы. Бұл жобалық шешімді әзірлеу үшін үлгі құралады. Бұл жиі график түрінде байқалады және объектілердің арасындағы қарым-қатынасты көрсетеді.
4. Зардаптар туралы мәлімдеме – нәтиже және ымыра – үлгінің қолдануы. Бұл өңдеушінің түсінуіне көмектеседі.

Сызба аты: Бақылаушы

Баяндама: Нысанның көрінісін өзінен алыстатады және басқа нысандардың көріністерін қамтамасыз етеді.

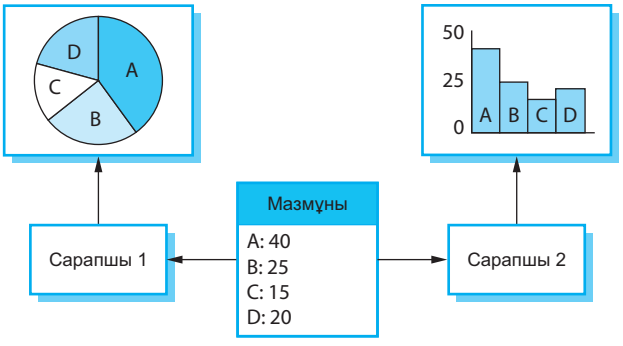
Мәселенің баяндамасы: Көптеген жағдайларда, сіз бір ақпараттың әртүрлі көріністерін суреттеуге мәжбүр боласыз. Мысалы, графикалық немесе кестелі көріністер. Ақпараттың анықталу кезінде олардың барлығы белгілі бола бермейді. Барлық суреттер интерактивті әрі ақпараттың өзгеруіне байланысты өзгермелі болу керек. Бұл сызба суреттеу жағдайларының бірден көп болған кезінде қолданылады.

Шешімнің баяндамасы: Бұл Пән және Бақылаушы деген негізгі екі тұжырымды қамтиды. Бұл тұжырымдар барлық жағдайларға келетін қарапайым операцияларды қамтиды.

Салдар: Әр пән тек қана өзінің бақылаушысы жөнінде хабардар болады. Берілген ақпараттың жеткіліксіз болғандығынан суреттердің өнімділігін жақсарту үдерісі тәжірибесіз қалады. Пәннің өзгерісі кейбір керек емес байланыстарға алып келеді.

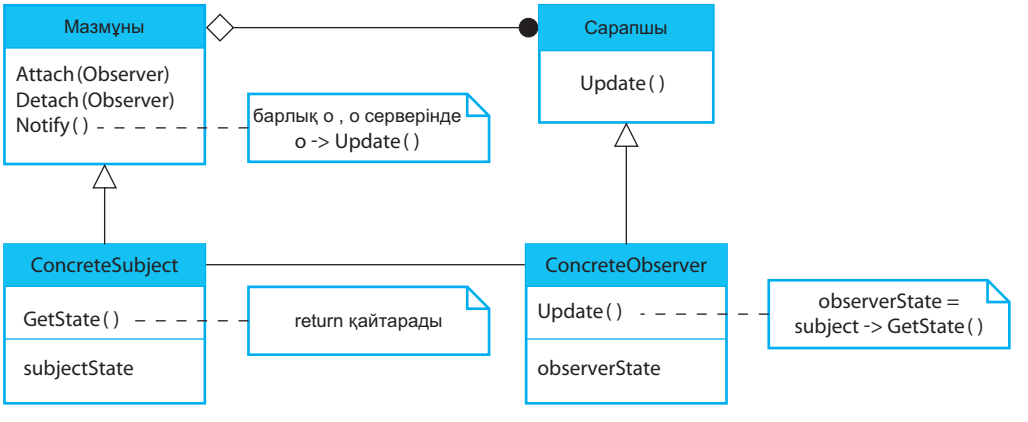
7.10-сурет. Бақылаушы сызбасы

Гамма және оның екінші авторлары (сипаттама үлгісі іске жарайтын) себепте мәселенің сипаттамасын және (алгоритмға мүмкін жұмсалатын жағдайдың сипаттамасы) қолданушылығын үзеді. Шешім баяндаумен сәйкес келеді, олар құрылымның үлгісін, қатысушыларды, ынтымақтастықты суреттейді және өткізеді.



7.11-сурет. Көптеген әртүрлі көріністер

Үлгінің сипаттамасы үшін, мен мысал келтірдім. Гамма (1995) кітаптан алынған үлгі бақылаушысын пайдаланамын. Бұл *7.10-суретте* көрсетілген. Менің сипаттамамда, сипаттаманың төрт негізгі элементтері, сонымен бірге қысқаша мазмұндама бар. Бұл үлгі объектінің күй жағдайларын, әртүрлі презентацияларда пайдалануын талап етпейді. Ол көріністің әртүрлі пішіндерінде суреттеуі керек болған объект бөледі. Бұл деректерді теру екі график түріндегі көріністер *7.11-суретте* көрсеткен.



7.12-сурет. Бақылаушы үлгісінің UML кескіні

Үлгіде объектілерінің талаптары көркемдеуі және қарым-қатынас үшін пайдаланатын олардың график түрінде көрсетеді. Бұл үлгінің сипаттамасын толық және сипаттаманың ерітіндіге бөліп қосуы болып табылады. *7.12-суретте* үлгі бақылаушы UML көрінісі көрсетілген.

Сіздің әрлендіруіңізде үлгіні пайдалану үшін сіз мәселенің кезкелген әрлендіруі мүмкін қолданылған немесе байланған үлгі ие болуын мақұлдауыңыз керек. Мұндай тапсырмалардың мысалдарымен кітапта сипатталған:

1. Басқа объектінің күйін (үлгі бақылаушы) өзгертетін бірнеше объектілерді айтыңыз.
2. Болған іс жиі (үлгінің фасады) біртіндеп жасалынып байланған объектілерді қатарға интерфейстерді жинастыру.
3. Топтаманың элементтеріне қол жеткізудің үйреншікті тәсіліне ілігеді, тәуелсіз жиын (үлгінің итераторы) жүзеге асырылады.
4. Орындау уақытына (декоратордың үлгісі) кластардың функционалды кеңейту мүмкіндігін ескеру.

Үлгі биік деңгейде, қайтадан пайдаланудың тұжырымдамасында қолданылады. Талпынғанда қайтадан құрауыштардың орындаушыларына арналған жасалған істің толық жобалық шешімдері шектеулі. Олар нақты алгоритмдарының объектілерге құрауыштарды өткізу үшін пайдаланылған істерді және құрауыштың интерфейстерінің түрлерін өзгертеді. Құрауыштарды сіздің талаптарыңыз қайтадан пайдаланумен бұл жанжалдың жобалық шешімдерінің болуы мүмкін емес немесе сіздің жүйеңізге тиімсіздікті енгізеді. Үлгілерді пайдалану сіз қайтадан пайдаланатыныңызды білдіреді, бірақ сіз өндейтін жүйемен сәйкестікте өткізу мүмкіндігі дамиды.

Егер сіз белгілі үлгіні керексінсеңіз алдын ала болған жүйенің жобалауын бастаңыз. Қорыта келгенде, үлгілерді, мәселе сынайтын әрлендіру көбінесе, дамуға қатысты жобалаудың үдерісінде пайдаланумен, ал үлгі жұмсалатынын содан соң мақұлдауы мүмкін. Бұл, әрине, сіз іздеген үлгінің жалпы 23 тағайындауында өрнектердің біртума кітабында сипатталуы мүмкін. Алайда, егер сіздің мәселеңіз қиын болса, сіз оны ұсынылған істің әр түрлі үлгілерін, жүздіктерін арасындағыларын табуға қосылады.

Үлгі өте жақсы ой болып көрінеді, оларды тиімді пайдалану үшін қажетті бағдарламалық жасақтама тәжірибе әзірлейсіз. Сіз үлгіні қолдануға керек болғанда мақұлдауыңыз мүмкін. Егер олар кітаптан өрнек оқыса, тәжірибесіз бағдарламалаушылар қиын шешеді, үлгіні олар қайтадан алуы керек немесе арнаулы шешімдер жасауға керек.

7.3. Іске асырудың сұрақтары

Бағдарламалық қамтамасыз етулерді әзірлеу өзіне бағдарламалық қамтамасыз етудің бастапқы жүйелік талаптарыннан техникалық қызмет көрсетуге дейінгі аралықты қосып жатыр. Бұл үдерістің дағдарыстық кезеңі, бағдарламалық қамтамасыз етулерді іске асыратын жүйелерді енгізумен тікелей байланысты. Іске асыру өзіне бағдарламаларды әзірлеуде жоғарғы деңгейдегі немесе төмен деңгейдегі программалық тілді қосады.

Менің болжамым бойынша, бұл кітапты оқыған оқырмандардың көпшілігі программалаудың қағидаларын түсініп және программалауда кейбір тәжірибе алады. Мен сұрақтарға немесе программалық мысалдарға тоқталмаймын. Бұлардың

орнына, мен, бағдарламалық қамтамасыз етуге маңызды программалық тілде қаралмаған мәтіндерді ұсынамын. Оларға жатады:

1. *Қайтадан қолдану.* Қазіргі программалық қамтамасыз етулердің көпшілігі компоненттердің және бар жүйелерді қайтадан қолданылуымен құрылған. Программалық қамтамасыз етуді әзірлеуде, сіз кодта қолдануға болатын барлық мүмкіндіктерді пайдалануыңыз қажет.
2. *Кескінмен басқару.* Әрбір программалық компонентті әзірлеуде, көптеген әртүрлі болжамдар жасалады. Егер сіз кескіндерді басқаруларды жүйеде қадағалап отырмасаңыз, сіз міндетті түрде жүйеңізде бұл компоненттердің бұрыс болжамдарын қосуыңыз керек.
3. Дамытулардағы хост- нысаналы параметрлер. Программалық қамтамасыз етулер бір компьютерде орындалмайды. ІШындап келгенде, сіз (хост-жүйеде) бір компьютерде оның жетілдіріп және (нысаналы жүйеге) жеке компьютерде оны орындайсыз. Сонымен қатар, нысаналы жүйелер кейде бір және сол тип сияқты, бірақ, олар жиі әртүрлі болып келеді.

7.3.1. Қайтадан қолдану

1990 жылдан 1960 жылға дейін жаңа программалық қамтамасыз етулер басынан бастап, биік деңгейдегі программалық тілдегі кодта жазылған. Қайтадан қолданудың жалғыз маңызды мәні программалық қамтамасыз етудегі программалау тіліндегі функциялардың және объектілердің библиотекаларын қайтадан қолдануы еді. Дегенмен, шығыстар мен кестелер коммерциялық және Интернет-Жүйелер үшін жарамсыз екенін көрсетіп тұр.

Программалық қамтамасыз етуді қайтадан қолдану бірнеше деңгейлерден кейін ғана мүмкін, олар:

1. *Абстракциялық деңгей.* Бұл деңгей программалық қамтамасыз етулерді қайтадан қолданумен тікелей байланысты емес, бұл деңгейде программалық қамтамасыз етулерді әзірлеуде тиімді абстракциялық білімдерді қолдану керек. Жобалаулар мен сәулеттік үлгілерді қайтадан қолдануы үшін абстрактілі білімдердегі әдістерді қолдану қажет.
2. *Объекттің деңгейі.* Бұл деңгейде, сіз қолдан жасамай-ақ объектілердің библиотекаларын тікелей қолдана аласыз. Қайтадан қолдану типін іске асыру үшін, сіз қажет библиотекаларды табуға тиістісіз және бұл объектілер және әдістер функционалдық жағынан қажеттігін түсінуіңіз қажет. Егер сізге поштаны Java деген бағдарламада өңдеу қажет болса , сізге JavaMail деген библиотекадағы объектілер мен әдістерді қолдана аласыз.
3. *Компоненттердің деңгейі*. Лайықты функцияны және қызметті қамтамасыз ету үшін, бірге жұмыс істейтін объектілердің және жалғыз объектілердің, коллекциялардың компоненттерін қолдану қажет. Компонентті бейімдендіру және кеңейту жиі кейбір меншікті кодты қосу арқылы жасалады. Қайтадан

компоненттерді қолдануға мысал ретінде қолданбалы интерфейсті алуға болады. Оқиғалардың өңдеуді іске асырумен ортақ жалғыз объектілердің жиыны – басқарулар, дисплей, және т.д. Сіз дисплейдің нақты бөлшектерін анықтау үшін, атап айтқанда, экранды орналастыру мен түсті өзгерту үшін кодты іске қосып жазуыңыз керек.

4. *Жүйелік деңгей.* Бұл деңгейде, сіз жүйелердегі қосымшаларды қайтадан қолданасыз. Бұл жүйелерге әлдеқандай кескінді қосады. Бұл кодқа қосымша немесе өзгеріс енгізумен немесе жүйедегі кескіндеудің меншікті интерфейсінің көмегімен жасалынады. Коммерциялық жүйелердің көпшілігі осы жолмен жасалған, ортақ COTS (коммерциялық бар адаптерленген) арқылы. Кейде сондай тұрғы өзіне бірнеше жүйелерді қолдануды және жаңа жүйені жасаудағы кірігуді қоса алады.

Қазіргі программалық қамтамасыз етуді қайтадан қолданудың арқасында, сіз тез жаңа жүйелерді жасай аласыз және шығындарды азайта аласыз. Қайтадан программалық қамтамасыз ету басқа бағдарламаларда тестіленген болатын, жаңа программалық қамтамасыз етулерге қарағанда сенімді болуға тиіс. Қайтадан қолдануымен сабақтас шығындар бар, олар:

1. Қайтадан қолдануы үшін программалық қамтамасыз етуді іздейсіз және ол сіздің қажеттіктеріңізге жауап бере ме жоқ па соны анықтайсыз, бұл сіздің уақытыңызды алады. Мүмкін, көз жеткізу үшін, сізге программалық қамтамасыз етуді тексеруге тура келер.
2. Көп біржолғы программалық қамтамасыз етулерді сатып алуға тура келеді. Бұл шығындар өте көп қаржыны қажет етеді.
3. Бірнеше рет программалық қамтамасыз етулердегі компоненттерді қолданулар мен бейімделуге және күйге келтіруге кеткен шығындар сіз өндейтін жүйемен байланысты.
4. Қайтадан қолданатын кодтағы элементтер шығындарды қажет етеді. Қалай олардың программалық қамтамасыз етуі қайтадан қолданыла алатыны әртүрлі жабдықтаушылардан программалық қамтамасыз етулерді алу қиын және қымбат.

Бағдарламалық қамтамасыз етулерді әзірлеулер жобасын бастаған кезде, қалай бағдарламалық қамтамасыз етуді қолдануды бірінші ойлауға тиістісіз. Сіз бағдарламалық қамтамасыз етулерді қайтадан қолданудағы барлық мүмкіндіктерді бағдарламалық қамтамасыз етуді әзірлеуге дейін қайтадан қарап, өз дизайныңызды бейімдендіре аласыз.

Қолданбалы жүйелердің үлкен саны, шын мәнінде, бағдарламалық қамтамасыз етулерді әзірлеу бағдарламалық қамтамасыз етулерді қайтадан қолдану дегенді білдіреді. Сондықтан мен бұл тақырыпқа байланысты “Технологияда бағдарламалық қамтамасыз ету” деген бөлімде арнайы тоқталдым (*16 , 17 , 19-бөлімдерде*).

7.3.2. Кескінмен басқару

Бағдарламалық қамтамасыз етудің әзірлеуінде бүкіл уақыт өзгеріс жанында болады, дегенмен, өзгерістермен басқару абсолютті қажетті болып көрінеді. Адамдарды басқарып бағдарламалық қамтамасыз ету кезінде, сіздің тобыңыздың мүшелері бір-бірінің жұмысына кедергі келтірмегенін қол жеткізуіңіз керек. Яғни егер екі адам өзгеріс үйлестіруімен шұғылданса, құрамдас топ болғандары жөн. Басқа жағдайда, бір бағдарламашы өзгерісті киргізіп, екіншісі оны қайта жазып жіберуі мүмкін. Сіз әр бағдарламалық құрамдас бөліктердің қазіргі нобайларына енгеніне қол жеткізу алатынын білуіңіз керек, басқа жағдайда өңдеуші ретінде жасалған жұмысты қайта істеуі мүмкін. Жүйенің жаңа болжамымен бірдеңе дұрыс емес болса, сіз дәл осылай бар жүйенің жұмыс нобайына қайтару жағдайды немесе құрамдас бөлікті көтеруіңіз керек.

Кескінмен басқару бағдарламалық жүйені қамтамасыз етудің өзгеріс үдерісінде көрсетілген атауы болып көрінеді. Кескіннің басқару мақсаты жүйенің ықпалдасу үдерісін қолдайды, компилятор және жүйенің жасауы үшін құрамдас бөліктерді жеткізеді, бақыланатын бейненің құжаттары болған ісінің жасалған өзгерісін айқындайды. Қорыта келгенде, кескіннің үш негізгі шаралары бар:

1. Нобайлармен басқару, көмек беру жағдайында программалық құрамдас бөліктер әртүрлі болжамдарды зерттеп отырады. Басқару жүйесінің нобайы бірнеше бағдарламашыларды дамудың үйлестіруі үшін қосады.
2. Жүйелік ықпалдасу, көмек беру жағдайында өңдеушіні анықтауға, жүйенің әр нобайын жасау үшін пайдаланады. Сипаттама бұл құрастырма жүйенің құрылысы үшін автоматты түрде пайдаланады және қажетті құрамдас бөліктерін құрастырады.
3. Көмек беру жағдайындағы мәселелерді зерттеп отырып, өңдеушілер мәселелердің үстінде жұмыс істегенін көрсетіп, қолданушыға тағы басқа мәселенің қатесін хабарлауға рұқсат етілуі керек.

Бағдарламалауды кескінмен басқару жоғарыда айтылған кез-келген құралдармен жұмыс істей алады. Бұл құралдар өзгеріс кешенді басқару жүйесінде бірлескен жұмыс үшін арнала алады (Белладжио және Миллиган, 2005). Кескіндермен, нобайларды басқару, жүйелік ықпалдасу, интегралданған басқару бірге жасалды. Кодтың біртұтас қоймасы арқылы олар қолданбалы интерфейстің стилін бөледі.

Одан басқа, әзірлеу интегралдалған орта орнатылған жеке құралдар қолданыла алады. Нобайларды басқару Subversion (Pilato et al., 2008) жүйесін қолдана алады. Бұл кезкелген алаңда және кезкелген топтармен жұмыс жасауға мүмкіндік береді. Аңду жаңылысы немесе зерттеп отыру бөліктің шығаруы, тағы басқа сұрақтар қатені хабарлауға пайдаланады және бұл болған істерінің түзетілгенін шынымен зерттеп отырады.

Бағдарламалық қамтамасыз етудің кәсіби әзірлеуінде өз маңыздылығын және өзгерістерді басқаруды мен толығырақ *25-бөлікте* талқылаймын.

7.3.3. Мақсатты даярлау

Кейде, әзірлеу және бағдарламалық қамсыздандыру платформалары бірдей. Сондықтан оны бір компьютерде әзірлеп және дереу тексеріп алуға болады. Бірақ кейде олар ерекшеленеді, сізге бағдарламаңызды тестілеу үшін бағдарламалық қамтамасыз етуді ауыстыру немесе өңдеушінің компьютеріндегі симуляторды іске қосу керек.

Программалық әзірлемелерді көпшілік мақсаттық үлгі негізделеді. Бағдарламалық қамтамасыз ету (хост) бір компьютерде жасалады, бірақ (мақсат) жеке компьютерде жұмыс істейді. Негізінен, біз әзірлеудің тұғыры және тұғырдың орындауы туралы сөйлей аламыз. Тұғыр қарапайым жабдыққа қарағанда артығырақ. Ол басқару жүйесінен, сонымен бірге басқа басқару программаларынан, жүйе дерекқор немесе әзірлеудің тұғырларынан құралады.

Симулятор енгізілген жүйелерді әзірлеуде жиі пайдаланады. Сіз аппаратты құрылғыларды сылтауратсаңыз, көрсеткіштері және жүйеге жазғызған ортада оқиға ашылады. Жүйеге симулятор енгізген әр өңдеушінің мақсатты – бағдарламалық жабдықты қамтамасыз ету, жүктеуді орындаудың тұғырын қажеттіліксіз өз меншікке ие болған жүйелердің әзірлеу үдерісін нығайту. Алайда, тренажер үшін бағдарлама әзірлеу қымбат болып көрінеді және де бұл әйгілі аппаратты архитектуралар үшін түсінікті.

UML орналастыру диаграммалары

UML орналастыру диаграммалары бағдарламалық жасақтама компоненттерінің процессорларда физикалық түрде орналасу жолдарын көрсетеді.; бұл дегеніңіз, орналасу диаграммасының жүйедегі жабдық пен бағдарламалық жасақтаманы және жүйедегі түрлі компоненттерді жалғауға қолданылатын байланыстырушы бағдарламалық жасақтаманы көрсету дегенді білдіреді. Іс жүзінде, сіз орналастыру диаграммаларын мақсатты ортаны анықтау және құжаттау жолы ретінде қабылауыңыз мүмкін.

http://www.SoftwareEngineering-9.com/Web/Deployment/

Егер мақсаттық жүйеде аралық орнаса немесе сіз пайдаланатын басқа бағдарлама болса, онда сіз бұл бағдарламалық қамтамасыз етуді көмекші жүйеге келтіру арқылы көтеруіңіз керек. Егер ол мақсаттық тұғыр сондай болатын болса, бұл сіздің компьютеріңіздегі бағдарламалық қамтамасыз етуді қондыру үшін құнсыз болуы мүмкін, лицензиялық шектеулер артынан қосылады. Мұндай жүйенің тестеуі үшін игерілген код тұғырды орындау мақсатында көшіреді.

Бағдарламалық қамтамасыз етудің әзірлеуі тұғыр бағдарламалық қамтамасыз ету үдерістерін қолдау үшін, төменде көрсетілген кең спектрлі құралдарды қамту керек. Олар:

1. Интегралданған компилятор және кодты әзірлейтін редактор сізге кодыңызды әзірлеуге, тексеруге және дұрыстауға мүмкіншілік беруге тиіс.
2. Тілді дұрыстау жүйесі.
3. Редакциялаудың график түріндегі құралдары. Мысалы, UML үлгілері үшін қолданатын құрал.
4. Құралдарды тестілеу. Жаңа программа тесттерді автоматты түрде орындайды (Massol, 2003) JUnit.
5. Дамудың әртүрлі жобалары үшін сіздерге көмектесетін құралдарды қолдаудың жобасы құрастырылады.

Осы стандартты құралдармен қатар, бағдарламаларды жасақтау жүйесінің құрамына статикалық талдау әдістері секілді (15-тарауда талқыланған) бірнеше арнаулы құралдардың қамтылуы мүмкін. Әдеттегі жағдайда, мамандар ұжымына арналған жасақтау шарттарының мазмұнында өзгерістер мен конфигурацияларды басқару жүйесін жүргізетін ортақтасқан сервер де кездесіп жатады және мұнда техникалық талаптарды басқару орталығына қолдау көрсетуші жүйенің де қамтылуы ықтимал.

Бағдарламалық жасақтаманы жасақтаушы құралдар Бағдарламаларды жасақтаудың біріктірілген ортасын (БЖБО) құру мақсатында жиі жағдайда топтасқан түрде келеді. БЖБО бірнеше ортақ арқаулық жүйе мен пайдаланушының интерфейсі арқылы бағдарламалық жасақтаманың бірнеше жасақталу аспектілерін қолдайтын бағдарламалық жасақтама құралдарының жиынтығы болып табылады. Жалпы айтқанда, БЖБО-лар бағдарламалық жасақтамаларды Java сияқты арнаулы бағдарламалық тілде жасақтау жұмыстарын қолдау мақсатында жасалып шығарылған. БЖБО арнаулы түрде жасақталуы мүмкін немесе ол арнаулы тілдерді қолдау құралдары бар жалпы мақсаттағы БЖБО бірнеше элемент құру амалы болуы да мүмкін.

Жасақталынып отырған бағдарламалық жасақтама жалпы мақсаттағы БЖБО құралдардың бірігіп жұмыс жасауына мүмкіндік береді және интеграция механизмдеріне арналған деректерді басқару құралдарымен қамтамасыз ететін орналастыру ортасындағы бағдарламалық жасақтаманың арқаулық жүйесін береді. Кеңінен таралған жалпы мақсаттағы БЖБО Тұтылу ортасы болып табылады (Карлсон, 2005 ж.). Бұл орта түрлі тілдер мен қолданба домендері үшін арнаулы түрде жасақталатын қосылатын модульдер архитектурасына негізделеді (Клейберг Рубель, 2006 ж.). Осылайша, Тұтылу ортасын орнатуға мүмкіндігіңіз пайда болады және оны қосылатын модульдерді қосу арқылы арнаулы қажеттіліктеріңізге сай етіп белгілі бір мақсаттар үшін жасақтап шығаруыңызға мүмкіндік бар. Мысалы, C-ті пайдалана отырып, Java немесе кіріккен жүйелердің техникалық жұмыстарын желілік жүйелерде қолдау көрсету мақсатында қосылатын модульдер жиынтығын қосып алуыңызға болады.

Жасақтау үдерісінің бір бөлігі ретінде мақсатты платформада жұмыс жасау үстінде болатын бағдарламалық жасақтаманың жасақталу амалдарына қатысты шешімдер шығаруыңыз қажет болады. Бұл әдетте, дара компьютер нысан ретінде қарастырылатын жағдайдағы ендірілген жүйелері үшін қарапайым болып табыла-

ды. Алайда, үлестірілген жүйелерде арнаулы платформалардағы компоненттердің жұмыс жасау үстіндегі орнын белгілеу туралы шешім шығару қажеттілік орын алады. Осы шешімді жасау барысында қарастырып отырған мәселелеріңіз мыналар:

1. *Компоненттердің жабдық және бағдарламалық жасақтама талаптары.* Егер компонент арнаулы жабдық архитектурасына арналып жобаланған болса, немесе ол басқа да бағдарламалық жасақтамаға сенім артатын болса, онда ол талап етілген жабдыққа немесе бағдарламалық жасақтамаға қолдау көрсету қызметімен қамтамасыз ететін платформада жұмыс жүргізілуін қажет етуі ықтимал.
2. *Жүйенің қолжетімділік талаптары* Жүйенің жоғары дәрежедегі қолжетімділігі компоненттердің бірнеше платформаларда жұмыс жүргізуін талап етуі мүмкін. Бұл платформа қатесі орын алған жағдайда компоненттердің альтернативті түрдегі іске қосу әрекеті қолжетімді болады деген мағынаны білдіреді.
3. *Компонент коммуникациялары* Егер компоненттер арасындағы коммуникациялардың тасымалдануы жоғары деңгейде болса, онда ол әдетте аталмыш платформада немесе басқасына физикалық түрде жақын орналасқан платформаларда жұмыс жүргізілуіне ықпалын тигізетін болады. Бұл коммуникациялардың күту кезеңінің деңгейін төмендетеді және хабарламаның бір компонент арқылы жіберіліп, келесі компонент арқылы қабылдануы арасындағы уақыт кідірісі орын алады.

Жабдық пен бағдарламалық жасақтаманың мазмұнына қолданбаларын ендіру жөніндегі шешіміңізді UML топологиясының сызбадеректері көмегімен құжаттауыңызға болады. Ол жерде бағдарламалық жасақтаманың жабдық платформасы арқылы үлестірілу жолдары көрсетіледі.

Егер ендірілген жүйені жасақтап жатқан болсаңыз, физикалық көлем, қуат күші, оқиғаларды сезінуге бағытталған нақты уақыттағы жауаптар, дискіенгізгі жетектері, физикалық сипаттамалары мен оның нақты уақыттағы амалдық жүйесі сияқты мақсатты сипаттамаларды есепке алуыңыз қажет болуы мүмкін. Мен ендірілген жүйелердің техникалық жұмыстары туралы 20-тарауда талқылап өттім.

7.4. Бастапқы ашық мәтіндерді жасақтау

Бастапқы ашық мәтіндерді жасақтау бағдарламалық жасақтаманың бастапқы коды ашық түрде жарияланатын және ерікті тұлғалардың жасақтау үдерісіне қатысуға шақырылатын бағдарламалық жасақтаманы жасақтау әрекетіне деген қатынас болып табылады. (Реймонд, 2001 ж.). Бұның тамыры тегін бағдарламалық жасақтама қорының негізінде жатыр (http://www.fsf.org), мұнда бастапқы кодтың меншіктік болып кетпеуін қорғайды, бірақ пайдаланушылардың өз қалаулары бойынша тексеріс жасауға немесе түрлендірулер жүргізуге әрдайым қолжетімді болуын да

қарастырып отырады. Мұнда кодтар аталмыш кодты пайдаланушыларға қарағанда, шағын орталық тобы арқылы басқарылады және жасақталады деген тұжырым бар.

Бастапқы ашық мәтіндердің бағдарламалық жасақтамасы ерікті жасақтаушылардың айтарлықтай ірі көлемге жеткізуге септігін тигізетін интернетті пайдалану арқылы бұл ойды кеңейте түсті. Олардың көбі кодтың пайдаланушылары. Негізінде, кем дегенде бастапқы ашық мәтіндерінің жобасына үлес қосушы кез келген тұлға баяндамалар жасап қана қоймай, сондай-ақ, ақаулықтардың алдын алу әрекеттерін және де жаңа мүмкіндіктер мен қызметтердің ауқымын болжамдай алады. Алайда, тәжірибе жүзінде, бастапқы ашық мәтіндерінің сәтті жүйелері бағдарламалық жасақтамаға енгізілетін өзгертулерді басқарып отыратын жасақтаушылардың орталық тобына әлі де зор сенім артып келеді.

Linux амалдық жүйесі сервер жүйесі ретінде кең қолданысқа ие және қарқынды түрде жұмыс үстелінің ортасы ретінде қолданылатын бастапқы ашық мәтіндердің кеңінен танымал өнім болып табылады. Басқа да маңыздылығы жоғары болып саналатын өнім қатарына Java, Apache веб сервері және mySQL дерекқормен басқару жүйесін енгізуге болады. IBM және Sun сияқты компьютер өнеркәсібіндегі қатысушылардың басым көпшілігі бастапқы ашық мәтіндердің қозғалысын қолдайды және олар өздерінің бағдарламалық жасақтамаларын сол бастапқы ашық мәтіндердің өнімдеріне негіздеп отырады. Сонымен қатар, танымалдылық деңгейі төмен болып келетін бастапқы ашық мәтіндердің жүйелері мен компоненттерінің мыңдаған түрлері бар және оларды да пайдалануға болады.

Мұндай бастапқы ашық мәтіндердің бағдарламалық жасақтамаларды әдетте өте арзан бағаға сатып алуыңызға болады немесе оларға тегін қол жеткізе аласыз. Сіз қалыпты жағдайда осы бастапқы ашық мәтіндердің бағдарламалық жасақтамаларды еш төлемсіз жүктеп алуыңызға болады. Алайда, егер сізге құжаттамалар және қолдау көрсету қызметі қажет болған жағдайда, біраз төлем жасауыңызға тура келуі мүмкін, дегенмен төлем жасау көлемі әдетте мардымсыз көлемде болады. Бастапқы ашық мәтіндердің өнімдерін пайдаланудың басқа да негізгі пайдасы деп жетік жасақталған бастапқы ашық мәтіндердің жүйелеріне деген тұтынушылардың зор сенімін айтуға болады. Мұндай болудың себебі туындаған мәселелерін жасақтаушыға мәлімдегеннен кейінгі жүйенің жаңа шығарылған нұсқасын күту машақатынан гөрі сол мәселелерін өздері шешуге ниет білдірген пайдаланушылардың саны өте көп. Ақаулықтарға қатысты мәселелері меншіктік бағдарламалық жасақтамаларда орын алуы ықтимал мәселелерге қарағанда, аса жылдам зерттеледі және жөндеуден жылдам өткізіледі.

Бағдарламалық жасақтаманы жасақтау әрекетіне жұмылдырылған компания үшін талқылауды қажет ететін бастапқы ашық мәтіндерге байланысты екі мәселе бар, олар:

1. Жасақталып отырған өнім бастапқы ашық мәтіндердің компоненттерін пайдалану қажет пе?
2. Бастапқы ашық мәтіндердің қатынасын бағдарламалық жасақтаманы жасақтау үдерісіне пайдалану қажет пе?

Осы сұрақтарға берілетін жауаптар жасақталып отырған бағдарламалық жасақтаманың түріне және өндігіне, сондай-ақ, жасақтаушы мамандар ұжымының тәжірибесіне байланысты болады.

Егер сіз бағдарламалық жасақтамаңызды сату үшін жасақтап жатсаңыз, онда маркетинг жүргізу уақыты мен шығындарды қысқартуға қатысты жағдайыңыз сыни болып келеді. Егер мазмұнында жоғары сападағы бастапқы ашық мәтіндердің жүйелері қолжетімді болып келетін доменде жасақтап жатқан болсаңыз, сол жүйелерді пайдалана отырып, уақытыңыз бен қаражатыңызды үнемдеп қалуыңызға мүмкіндігіңіз бар. Алайда, егер бағдарламалық жасақтаманы ұжымдық талаптардың арнаулы жиынтығы үшін жасақтап отырсаңыз, онда ол жерде бастапқы ашық мәтіндердің компоненттерін опция ретінде пайдалана алмайсыз. Сізге өзіңіздің бағдарламалық жасақтамаңызды қолданыста бар жүйелермен біріктіруіңізге тура келуі мүмкін, бірақ бұл қолжетімді бастапқы ашық мәтіндердің жүйелерімен үйлеспейтін болады. Тіпті солай болған жағдайда да, бастапқы ашық мәтіндер жүйесінің түрлену әрекеті өзіңізге қажетті қызмет түрін қайта жасақтауға қарағанда, жылдам және арзан түсуі мүмкін.

Жасақтау әрекеттеріне қатысты бастапқы ашық мәтіндердің қатынастарын пайдаланушы компаниялардың саны артып келеді. Олардың бизнес үлгісі өнімге арналған сатылымды қолдауға қарағанда, сату үшін аса сенімділікке ие емес. Олардың сенімі бойынша, бастапқы ашық мәтіндердің қауымдастығы бағдарламалық жасақтаманың арзанырақ әрі жылдамырақ жасақталуына және бағдарламалық жасақтамаларға арналған пайдаланушылардың қауымдастығын құруға мүмкіндік береді деп біледі. Тағы бір айта кететін жайт, бұл арнаулы ұжымдық қолданбаларға қарағанда, тек жалпы бағдарламалық жасақтама өнімдері үшін ғана шын мәнінде қолданыс табады.

Көптеген компаниялардың тұжырымы бойынша бастапқы ашық мәтіндердің қатынастарын қолдану әрекеті олардың бәсекелес компанияларына құпиялы бизнес ақпаратын әшкерелеп береді және осы себептен олар осы жасақтау үлгісін қабылдауға ынтасы өте төмен болады. Алайда, егер сіз шағын компанияда жұмыс жасап жүрген болсаңыз және өзіңіздің бағдарламалық жасақтамаңызды бастапқы ашық мәтіндеріңізді ашық күйде ұстасаңыз, компанияңыз бизнестен тыс кеткен кезде тұтынушыларыңыздың үміті қайта оянуы мүмкін, осылайша олар бағдарламалық жасақтаманы жасақтауға қолдау көрсету мүмкіндігіне ие бола алады.

Жүйенің бастапқы кодын жариялау әрекеті ауқымы кеңірек болып келетін қауымдастыққа мүше адамдардың жасақтаманы жасақтауға қажетті көмек көрсетеді деген мағынаны білдірмейді. Айтарлықтай сәтті болып саналатын бастапқы ашық мәтіндерінің өнімдері қолданба өнімдеріне қарағанда, платформа өнімдері болып табылады. Мұнда арнаулы қолданба жүйелеріне қызығушылық танытатын жасақтаушылардың шектеулі саны берілген. Мысалы, бағдарламалық жасақтаманы бастапқы ашық мәтіндері ретінде жасау әрекеті қауымдастықты өзіне баулу кепілдігін бермейді.

7.4.1. Бастапқы ашық мәтіндерді лицензиялау

Дегенмен, бастапқы ашық мәтіндерді жасақтаудың фундаменталды негізі сол ашық мәтін коды тегін қолжетімді болуы қажет, бірақ бұл дегеніңіз кез келген адамның кодпен қалағанын жасай алу деген мағынаны білдірмейді. Заң жүзінде, кодты жасақтаушы тұлға (не компания, не жеке тұлға) кодтың иесі болып қала береді. Олар бастапқы ашық мәтіндерінің бағдарламалық жасақтама лицензияларының мазмұнында заңды түрде міндеттелген шарттарға енгізу арқылы пайдалану жолдарына шектемелер түрін белгілеуге құқылы болып табылады. (Снт. Лорент, 2004 ж.). Кейбір бастапқы ашық мәтіндерінің жасақтаушылары жаңа жүйені жасақтауға қолданылатын бастапқы ашық мәтіндерінің компонеттерінің де бастапқы ашық мәтіндерінің болу мүмкіндігіне сенім артады. Басқа жасақтаушылар өздерінің кодтарын ешқандай шектемелерсіз қолданылауын қалайды. Жасақталған жүйелер меншіктік болуы мүмкін және олар жабылған бастапқы мәтін ретінде де сатыла беруіне болады.

Бастапқы ашық мәтіндерінің лицензиялардың басым көпшілігі жалпы үш үлгілер алынады:

1. GNU Стандартты қоғамдық лицензия (СҚЛ). Бұл сондай-ақ, ‘кері шама’ лицензиясы деп те аталады, қарапайым тілмен айтқанда, бұл егер сіз СҚЛ лицензиясы арқылы лицензияланған бастапқы ашық мәтіндерінің бағдарламалық жасақтамасын пайдаланып отырсаңыз, онда сіз қолданып отырған сол бағдарламалық жасақтамаңызды бастапқы ашық мәтіндері етіп жасауыңыз тиіс.
2. GNU Қолданысы шектеулі стандартты қоғамдық қоғамдық лицензия (ҚШСҚЛ). Бұл сол компоненттердің бастапқы мәтінін жариялау қажеттілігінсіз бастапқы ашық мәтіндерінің кодына байланыстыратын компоненттерді жазу мүмкіндігін беретін СҚЛ лицензиясының бір нұсқасы болып табылады. Алайда, егер сіз лицензияланған компонент үшін төлем жасаған болсаңыз, онда сіз оны бастапқы ашық мәтіндері ретінде жариялауыңыз керек.
3. Берклей стандарттық үлестірілімі (БСҮ) лицензиясы. Бұл кері шамаға жатпайтын лицензия түрі болып саналады. Ол сіздің бастапқы ашық мәтіндеріңіздің кодына енгізілетін кез келген өзгертулер мен түрлендірулерді қайта жариялауға қатысты міндеттемелеріңізді жоқ деген мағынаны білдіреді. Сіз кодыңызды меншіктік жүйелер құрамына енгізіп қоюыңызға болады, ол меншіктік жүйелер сатылады. Егер сіз бастапқы ашық мәтіндерінің компоненттерін пайдаланатын болсаңыз, сіз сол кодтың жасап шығарған бастапқы тұлғасы туралы біліміңіздің болуы тиіс.

Лицензиялау мәселелеріне қатысты сұрақта маңызды болып келеді, себебі, егер сіз бастапқы ашық мәтіндерінің бағдарламалық жасақтамасын бағдарламалық жасақтаманың бір бөлігі ретінде пайдаланатын болсаңыз, онда сіз өзіңіздің өніміңізді бастапқы ашық мәтіндерін жасауыңыз үшін лицензияның шарттарын орындауға міндеттеме алатын боласыз. Егер сіз өзіңіздің бағдарламалық жасақтамаңызды сатуға ниеттенсеңіз, онда сіз бұл шешіміңізді құпияда сақтауды

жөн санауыңыз мүмкін. Бұл дегеніңіз өзінің жасақтауындағы СҚЛ арқылы лицензияланған бастапқы ашық мәтіндерінің бағдарламалық жасақтамасын пайдаланудан аулақ болу ниеті деген сөз.

Егер сіз Linux сияқты бастапқы ашық мәтіндерінің платформасында жүргізілетін бағдарламалық жасақтама құрып отырсаңыз, онда лицензияларға қатысты ешқандай мәселе туындамайды. Алайда, бағдарламалық жасақтамаңызға бастапқы ашық мәтіндерінің компоненттерін енгізу жұмыстары жаңадан басталғандықтан, сізге алдымен жұмыстың орындалу жолы мен олардың лицензия шарттарын сақтап отыруға қатысты үдерістер мен дерекқорларды орнатып алу қажет. Байерсфорфер (2007 ж.) бастапқы ашық мәтіндерінің қолданып отырған жобаны басқарушы компаниялардың келесі әрекеттерді орындау қажеттілігін ұсынады, олар:

1. Жүктеліп алынған және қолданыс үстіндегі бастапқы ашық мәтіндерінің компоненттері туралы ақпарат жасап шығаруға арналған жүйе орнату. Сіз қолданылып отырған уақыт ішінде пайдалану мерзімі жарамды компоненттің әрқайсысы үшін берілетін лицензия көшірмесін сақтап отыруыңыз қажет. Лицензиялар өзгертіліп отыруы мүмкін, сондықтан, келісіміңізді беріп отырған шарттар туралы білуіңіз қажет.
2. Лицензиялардың түрлі типтерін ескеріңіз және пайдалану алдындағы компоненттердің лицензиялану жолдарын түсінуге тырысыңыз. Сіз бір компонентті басқа емес тек бір жүйеде ғана қолдануды ұйғаруыңыз мүмкін. Себебі, сіздің жоспар бойынша бұл жүйелерді әртүрлі жолдармен пайдаланасыз.
3. Компоненттерге арналған даму бағытын ескеріңіз. Сіз компоненттердің болашақта орын алуы ықтимал өзгеріс жолдарын ұғынуға арналып жасақталған компоненттері бастапқы ашық мәтіндерінің жобасы туралы қандай да бір біліміңіздің болуы қажет.
4. Адамдарды бастапқы ашық мәтіндер туралы біліммен қамтамасыз етіңіз. Лицензия шарттарымен үйлесімді болуына көз жеткізу үшін орынды іс шараларын жүргізу жеткілікті емес. Сіз сонымен қатар, жасақтаушыларды бастапқы ашық мәтіндері мен бастапқы ашық мәтіндерінің лицензиялау туралы біліммен қамтамасыз етуіңіз қажет.
5. Іске қосылған тексеріс жүйелерінің қолданыста болу шарты. Жасақтаушылар қатаң түрде белгіленген аяқталу мерзімінде лицензия шарттарын бұзуға итермеленеді. Егер мүмкіндік болса, бұл жағдайды анықтау және тоқтату үшін іске қосылған бағдарламалық жасақтамаңыздың бар болуы қажет.
6. Бастапқы ашық мәтіндерінің қауымдастығына қатысу. Егер сіз бастапқы ашық мәтіндерінің өнімдеріне сеніміңіз берік болған жағдайда, сіздің аталмыш қауымдастыққа қатысуыңызға және олардың жасақтау жұмыстарына қолдау көрсетуге көмегіңізді тигізгеніңіз жөн.

Бағдарламалық жасақтамалардың бизнес үлгілері өзгеріп тұрады. Бұл орайда арнаулы бағдарламалық жасақтама жүйелерін сату арқылы бизнес жасау қиындығы өсіп келеді. Көптеген компаниялар өздерінің бағдарламалық жасақтамаларын бастапқы ашық мәтіндерін жасақтап шығаруды қалайды және солай жасағаннан

кейін олар бағдарламалық жасақтамалардың пайдаланушыларына қолдау көрсету қызметі мен кеңес беру қызметтерін сатумен айналысады. Бұл бағыт өз қарқынын күшейте түскенге ұқсайды. Себебі, бастапқы ашық мәтіндерінің бағдарламалық жасақтамаларды пайдаланушылар саны артып келеді және осы пішіндегі қолжетімді бағдарламалық жасақтама да көбею үстінде.

НЕГІЗГІ ТҰЖЫРЫМДАР

- Бағдарламаның әрленімі және әзірленуі кезектеседі. Әрленімдегі бөлшектердің деңгейі әзірленіп жатқан жүйеге және қолданылатын бағдарламалық жасақтаманың үдерісіне байланысты болады.
- Нысанға бағытталған әрленім жүйенің сәулетін әзірлеуден, жүйедегі нысандарды анықтаудан, әрленімді суреттейтін әртүрлі нысандардан және мүшелердің байланыстарын құжаттандырудан құралады.
- Нысанға бағытталған әрленім көптеген әртүрлі үлгілерді шығарады. Мысал ретінде бірқалыпты үлгілерді (топ үлгілері, жалпы үлгілер, байланысқан үлгілер) және айнымалы үлгілерді (қатар үлгілері) қарастыруға болады.
- Мүшелердің байланыстары басқа нысандарға жетімді болуы үшін тиянақты анықталуы керек. Пайдаланушыға арналған жүйелерді әрленімдегі байланыстарды анықтау үшін қолдануға болады.
- Бағдарламаны әзірлеу барысында сол бағдарлама негізіндегі басқа бағдарламаларды әзірлеу мүмкіндігін есте сақтауға тиіс.
- Құрылымды басқару үдерісі - бағдарламалық жасақтаманың өзгерістерін басқаратын жүйе. Бұл бірнеше адамдар бағдарламаны әзірлеуге қатысқанда өте пайдалы.
- Күрделі бағдарламалық жасақтама бөлшек-бөлімді болады. Сіз өзіңіздің бөлшегіңізде даярлаған бағдарламаңызды негізгі бөлімге салып, орындауыңызға болады.
- Ашық бастамадағы жасақтама даярлаған кодыңызды барлық адамдарға қолжетімді етеді. Бұл жасақтамаға кезкелген адам өзгерістерді енгізіп, оның сапасын жақсарта алады.

ҚОСЫМША ӘДЕБИЕТТЕР

Design Patterns: Elements of Reusable Object-oriented Software. This is the original software patterns handbook that introduced software patterns to a wide community. (E. Gamma, R. Helm, R. Johnson and J. Vlissides, Addison-Wesley, 1995.)

Applying UML and Patterns: An Introduction to Object-oriented Analysis and Design and Iterative Development, 3rd edition. Larman writes clearly on object-oriented design and, as well as discussing the use of the UML. This is a good introduction to using patterns in the design process. (C. Larman, Prentice Hall, 2004.)

Producing Open Source Software: How to Run a Successful Free Software Project. His book is a comprehensive guide to the background to open source software, licensing issues, and the practicalities of running an open source development project. (K. Fogel, O'Reilly Media Inc., 2008.)

ЖАТТЫҒУЛАР

7.1. *7.3-суретке* қарап, бекеттің Хабарламалы немесе Қайта-құрылымды регистрлерін қолданатынын анықтаңыз. Бұл жерде қолданылған функционал жөнінде күрделі талқылау жүргізуіңіз қажет.

7.2. MHC-PMS жүйесі нысанға бағытталған әрленіммен әзірленіп жатыр деп қарастырайық. Кемінде алты регистрді көрсететін регистрлер диаграммасын салыңыз.

7.3. Нысан топтарына арналған пайдаланушылар кестесін қолдана отырып, төменде көрсетілген:

- телефон
- жеке компьютерге арналған принтер
- жеке стерео жүйесі
- кітапхана каталогі сияқты нысандардың топтарын даярлаңыз

7.4. *7.6-суреттегі* ауа райы бекетінің анықталған нысандарын қолдана отырып, осы жүйеде қолдана алатын қосымша нысандарды анықтаңыз. Анықталған нысандардың байланыстарының құрылысын анықтаңыз.

7.5. Деректердің жиынының жүйесінің және ауарайы деректерін жинайтын аспаптардың өзара байланысын көрсететін ауарайы бекетінің әрленімін әзірлеңіз.

7.6. Топтық жүйедегі күнделікті кездесетін адамдардың байланысын көрсететін тізбектік диаграммасын әзірлеңіз.

7.7. Топтың күнделігінде немесе толтыратын бекет жүйесіндегі өзгерістерді көрсететін UML диаграммасын әзірлеңіз.

7.8. Осы мысалдарды қолдана отырып, топтық бағдарлама әзірлеудің маңыздылығын түсіндіріңіз.

ПАЙДАЛАНЫЛҒАН ӘДЕБИЕТТЕР

Abbott, R. (1983). 'Program Design by Informal English Descriptions'. *Comm. ACM*, **26** (11), 882–94.

Alexander, C., Ishikawa, S. and Silverstein, M. (1977). *A Pattern Language: Towns, Building, Construction*. Oxford: Oxford University Press.

Bayersdorfer, M. (2007). 'Managing a Project with Open Source Components'. *ACM Interactions*, **14** (6), 33–4.

Beck, K. and Cunningham, W. (1989). 'A Laboratory for Teaching Object-Oriented Thinking'. *Proc. OOPSLA'89* (Conference on Object-oriented Programming, Systems, Languages and Applications), ACM Press. 1–6.

Bellagio, D. E. and Milligan, T. J. (2005). *Software Configuration Management Strategies and IBM Rational Clearcase: A Practical Introduction*. Boston: Pearson Education (IBM Press).

Buschmann, F., Henney, K. and Schmidt, D. C. (2007a). *Pattern-oriented Software Architecture Volume 4: A Pattern Language for Distributed Computing*. New York: John Wiley & Sons.

Buschmann, F., Henney, K. and Schmidt, D. C. (2007b). *Pattern-oriented Software Architecture Volume 5: On Patterns and Pattern Languages*. New York: John Wiley & Sons.

Buschmann, F., Meunier, R., Rohnert, H. and Sommerlad, P. (1996). *Pattern-oriented Software Architecture Volume 1: A System of Patterns*. New York: John Wiley & Sons.

Carlson, D. (2005). *Eclipse Distilled*. Boston: Addison-Wesley.

Clayberg, E. and Rubel, D. (2006). *Eclipse: Building Commercial-Quality Plug-Ins*. Boston: Addison Wesley.

Coad, P. and Yourdon, E. (1990). *Object-oriented Analysis*. Englewood Cliffs, NJ: Prentice Hall.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, Mass.: Addison-Wesley.

Harel, D. (1987). 'Statecharts: A Visual Formalism for Complex Systems'. *Sci. Comput. Programming*, **8** (3), 231–74.

Kircher, M. and Jain, P. (2004). *Pattern-Oriented Software Architecture Volume 3: Patterns for Resource Management*. New York: John Wiley & Sons.

Massol, V. (2003). *JUnit in Action*. Greenwich, CT: Manning Publications.

Pilato, C., Collins-Sussman, B. and Fitzpatrick, B. (2008). *Version Control with Subversion*. Sebastopol, Calif.: O'Reilly Media Inc.

Raymond, E. S. (2001). *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. Sebastopol. Calif.: O'Reilly Media, Inc.

Schmidt, D., Stal, M., Rohnert, H. and Buschmann, F. (2000). *Pattern-Oriented Software Architecture Volume 2: Patterns for Concurrent and Networked Objects*. New York: John Wiley & Sons.

Shlaer, S. and Mellor, S. (1988). *Object-Oriented Systems Analysis: Modeling the World in Data*. Englewood Cliffs, NJ: Yourdon Press.

St. Laurent, A. (2004). *Understanding Open Source and Free Software Licensing*. Sebastopol, Calif.: O'Reilly Media Inc.

Wirfs-Brock, R., Wilkerson, B. and Weiner, L. (1990). *Designing Object-Oriented Software*. Englewood Cliffs, NJ: Prentice Hall.

8.

Бағдарламалық сынақ

Мақсаттары

Бұл тараудың мақсаты – бағдарламалық сынау және бағдармалық сынаудың үдерістерін ұсыну. Тарауды оқығаннан кейін, сіз:

- сынақтан сынақтың кезеңдерін, клиенттер жүйесіндегі қабылдау сынауларының дамуын түсінесіз;
- бағдарламаның ақауларын табуға арналған әдістемелерімен таныс боласыз;
- бас негіз сынаудың дамуын түсініп, кодты жазып және сынауларды автоматша қосудан бұрын сіз сынауларды жобалайсыз;
- компоненттің, жүйенің және сынау шығуының арасындағы өзгешеліктерді және қолданбалы сынау үдерістері мен әдістемелері туралы хабардар боласыз.

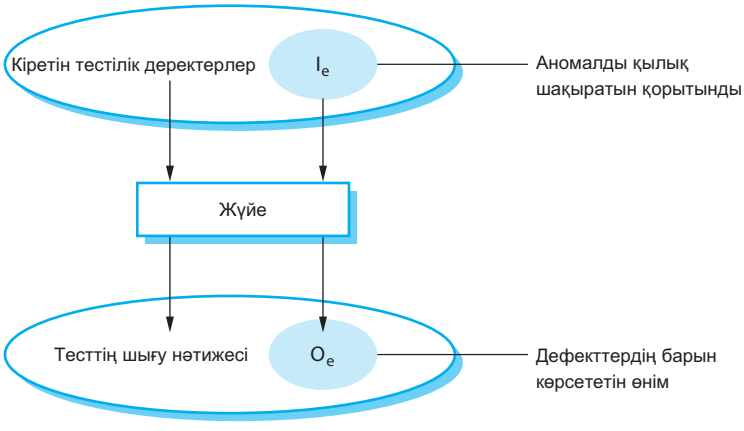
Мазмұны

Сынау бағдарламаның не істегісі келгенінін көрсетуге және ақауларды тауып алдын алуға арналған. Сіз бағдарламаны тексеру кезінде, бағдарламаның жасанды деректерін орындайсыз. Қателер, ауытқушылықтар немесе ақпарат туралы бағдарламаның функциялық емес сипаттары үшін сынаудың нәтижесін тексересіз.

Сынау үдерісінің екі түрлі мақсаты бар:

1. Бағдарламалық қамтамасыздандырудың әзірлеуші мен тапсырыс берушінің талаптарына сай екенін көрсету. Бағдарламалық қамтамасыздандыруға тапсырыс берушінің талап етілген құжатында кем дегенде әр талапқа бір сынау болу керек. Әмбебап бағдарламамен қамтамасыздандыру өнімдері үшін барлық жүйенің өзгешеліктері және оның комбинациялары өнімнің шығуына сынаулар қосылуы тиіс.
2. Бағдарламалық қамтамасыздандырудың тәртібі дұрыс емес, лайық емес немесе талабына келмейтін жағдайларды табу. Олар бағдарламалық қамтамасыздандыру ақауларының салдары болып табылады. Сынаудың ақауы – жүйенің бұзылуы, сыртқы жүйелермен байланысуы, дұрыс емес есептеулер және деректердің тұтас бұзылуы сияқты лайық емес жүйе тәртібімен байланысты.

Бірінші мақсат сіз күтетін жүйенің берілген терімін дұрыс колдануына әкеледі. Егер сынау ақауларды табуға арналған болса, екінші мақсат сынау ақаулары болып табылады. Сынау ақауындағы сынақ құрамдары жүйенің қалыпты қолданылуын көрсетпей, түсініксіз болады. Әрине, бұл екі сынау әдістерінің арасында ешқандай белгілі айырмашылық жоқ. Сынау тексеру кезінде, сіз жүйеден ақау табасыз; ақау сынағы кезінде, кейбір сынаулар бағдарламаның талаптарына сай екенін көрсетеді.



8.1-сурет. Бағдарламаны тестілеудің енгізу-шығару үлгісі

8.1-суретінде көрсетілген диаграмма, бізге сынау тексеру мен ақау сынауының арасындағы айырмашылықты көрсетуге көмектеседі. Сыналатын жүйені қара жәшік деп ойлаңыз. Жүйе кейбір кірер I жиынынан кіретін және шығар O жиы-

нынан шығатын дыбыстар қабылдай алады. Кейбір шықпа жиындар қате болады. Жүйе арқылы кірме I жиынына жауапты O шығу жиыны құрылады. Сынау ақауының басымдылығы кірме I_e жиынын табу, себебі ол жүйенің қатесін табады. Сынауды тексеру өзіне қажетті кірме I_e жиын сынауын өзіне қосады. Олар жүйені күтпелі дұрыс шығуды алуға ынталандырады.

Сынау бағдарламалық камтамасыздандырудың ақауларын көрсете алмайды немесе ол өзін көрсетілген жағдайдағыдай ұстайды. Бұл әрқашанда мүмкін, өйткені сіз байқамаған жүйедегі мәселелерді сынау тауып бере алады. Бағдарламалық қамтамасыздандыруға үлес қосқан, Эдсгер Дейкстра (Dijkstra et al., 1972) айтқандай:

Сынау тек қана қателердің жоқтығын емес, сондай-ақ барын көрсете алады.

Сынау верификация және валидация үдерісінің үлкен бөлігі болып табылады. Верификация және валидация бірдей емес, бірақ оларды жиі шатастырады. Бағдарламалық қамтамасыздандыру саласындағы пионері Барри Боэм (Boehm, 1979) екеуінің арасындағы айырмашылықты:

- Валидация: Біз дұрыс өнім шығардық па?
- Верификация: Біз өнімді дұрыс шығардық па? – деп жинақы айтады.

Верификация және валидация үдерістері, адамдардың бағдарламалық қамтамасыздандыруға, төлеміне арналған, бағдарламалық қамтамасыздандырудың спецификациясы мен оның фунукционалдық қабілетін тексерумен байланысты. Талаптар қолжетімді бола бастап және барлық үдеріс деңгейлерінде жалғасқаннан кейін бұл үдерістер тексеруден басталады.

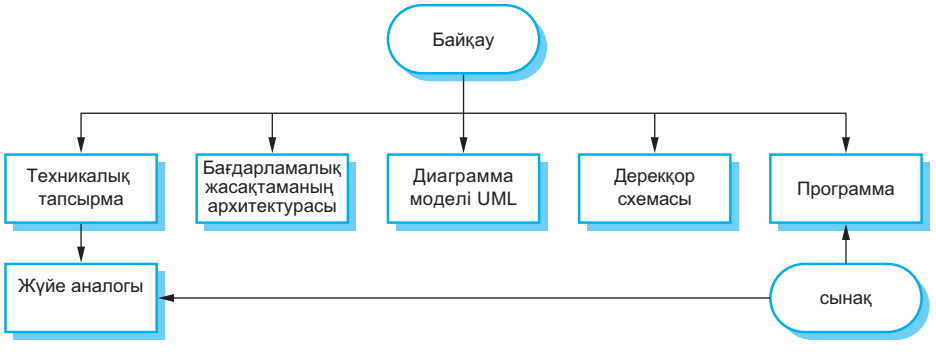
Тексеру мақсаты бағдарламаларға берілген функционалды және функционалды емес талаптарға сай тексеру болып табылады. Демек, валидация ортақ үдеріс екенін көрсетеді. Тексеру мақсаты бағдарлама мен қамтамасыздандыруға тапсырыс берушінің көңілінен шығуы болып табылады. Ол жай ғана спецификациямен, сәйкестіктерімен тексеру ауқымынан шығып, бұл бағдарламалық қамтамасыздандыруға тапсырыс берушінің күткенің істеп көрсетеді. Тексеріс қажет, өйткені, мен *4-бөлімде* айтқандай, техникалық талаптар тапсырыс берушілердің және пайданушылардың шынайы қалауын және қажеттіліктерін көрсетпейді.

Верификация және валификация үдерістерінің соңғы мақсаты, бағдарламалық қамтамасыздандыруды мақсатқа талапты етіп, сенімділікті туындату болып табылады. Жүйенің талап бойынша жеткілікті жақсы дәрежеде болуы дегенді білдіреді. Керекті сенімділік деңгейі жүйенің қолданушыларының мақсатына байланысты және қазіргі маркетингтік ортадағы жүйе үшін:

1. *Бағдарламалық қамтамасыздандырудың мақсаты.* Бағдарламалық қамтамасыздандыру қатерлі болған сайын, сенім көбірек болу тиіс. Мысалы, жүйенің қауіпсіздігіне маңызды, бағдарламамен қамтамасыздандыруға

арналған сенім деңгейі жаңа өнімнің идеяларын көрсетуге ойлап табылған прототиптен маңыздырақ.

2. *Қолданушының күтуі.* Көп қолданушылар бағдарламамен қамтамасыздандырудың сапасына сенбейді, себебі ол сенімсіз және олар қатемен көп кездескен. Олар бағдарламамен қамтамасыздандырудың қондырылмауына таңғалмайды. Жаңа жүйе қондырылғанда, қолданушылар қателер жібереді, сондықтан қолданудың артықшылығы қайта қалпына келтіруден асып түседі. Бұл жағдайда сіз бағдарламамен қамтамасыздандыруды сынауға көп уақыт бөле алмайсыз. Бұған қарамастан, бағдарламамен қамтамасыздандыру жетіліп, қолданушылар оның сенімді болуын күтеді.
3. *Маркетингтік орта.* Жүйе сатылымға шыққанда, сатушылар басекелес тауарларға, тапсырыс берушілердің жүйе үшін төлей алатын сомасына және осы жүйе бойыншы жұмыс істеу кестесіне мән беру керек. Қатал шарттарда, сатылымда бірінші болғысы келгендіктен, бағдарламалық қамтамасыздандыру компаниясы бағдарламаны тексермей тұрып, оны шығаруға шешім қабылдай алады. Егер де бағдарламалық өнім өте арзан болса, қолданушылар сенімділіктің төмен деңгейіне көшеді.



8.2-сурет .Тексеру және тестілеу

Сондай-ақ, бағдарламалық қамтамасыздандыру сынауы, верификация және валидация үдерістері ішіне бағдарламалық қамтамасыздандыру тексерістері мен қорытындысын кіргізе алады. Тексеру мен қорытындылау анализінен өтеді және жүйенің талаптары, модель дизайны, бағдарламаның бастапқы коды тексеріледі, тіпті, жүйе сынағы ұсынылады. Бұл, сіз бағдарламаны қоспай-ақ тексере алатын, статикалық В & В әдістемесі деп аталады. *8.2-суретте* В & В бағдарламалық қамтамасыздандырудың әзірлеуін, үдерістің әртүрлі кезеңдеріне тексерулер және қолдаудың сынауын бағдарламалық қамтамасыздандыру көрсетеді. Қолданылуы мүмкін әдістерде, бағыттар үдеріс кезенін көрсетеді.

Тексеріс бірінші кезеңде, бастапқы код жүйесіне көп мән береді, бірақ әр талап немесе модель дизайны сияқты, бағдарламалық қамтамасыздандырудың көрсетілімі тексеріле алады. Жүйе тексеру кезінде, сіз қателерді табу үшін, жүйе

туралы ақпаратты, оның заттық ауданын және бағдарламалау немесе модельдеу тілін қолдана аласыз.

Сынақ жоспарлау

Сынақ жоспарлау әрекетіне сынақ үдерісіндегі кесте құру және барлық қажетті іс-шараларымен қамтамасыз ету амалдарын жатқызуға болады. Оның құрамына сынақ жүргізу үдерісін анықтау, қолжетімді адамдар мен уақытты есепке алу әрекеттері кіреді. Әдетте, сынақ жоспары сынақ жүргізілетін нысанды, болжалды сынақ кестесін және сынақ жазбасын жүргізу жолдарын анықтайтындай етіп жасалады. Бағдарламалық жасақтамаға жүргізілетін сынақ жоспарларының мәліметтері де маңыздылығы жоғары жүйелерге арналған сынақ жоспарына кіруі мүмкін.

http://www.SoftwareEngineering-9.com/Web/Testing/Planning.html

Бағдарламалық қамтамасыздандыру арқылы сынаудың үш артықшылығы бар:

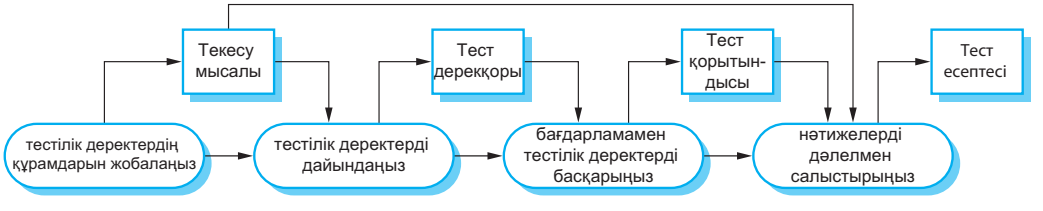
1. Сынау кезінде, қателер басқа қателерді жасыруы (көрсетпеуі) мүмкін. Егер қате күтпеген шығуға әкеліп соқса, кешірек шығу ауытқушылықтарын жаңа қателеден немесе бастапқы қатенің қосымша әсер көрсететіне ешқашан сенімді бола алмайсыз. Бақылау статикалық үдеріс болғандықтан, қателер арасындағы байланыста ісіңіз болмайды. Сонымен, тексерудің бір сеансының ішінде жүйеден көп қате табуға болады.
2. Жүйенің толық емес нұсқасы қосымша шығынсыз тексеріле алады. Егер бағдарлама толық емес болған жағдайда, жетім бөлшектерді тексеру үшін сізге арнайы сынау ойлап табу керек. Бұл жүйенің құрастыруының бағасын көтеретіні айқын.
3. Ақау іздеу бағдарламасы сияқты, тексеру, сондай-ақ бағдарламаның шамаларға сәйкестігі, мобильділік және жөндеуге келетіндігі сияқты кең сапа белгілерін қарастыруы мүмкін. Сіз тиімсіздікке, ретсіз алгоритмдерге және жаңарту жасай алмайтын жүйедегі программалаудың кедей стильдеріне қарай аласыз.

Бағдарлама инспекцияның көне идея және бірнеше зерттеулер, программадағы ақауларды табу үшін, тиімдірек болып табылғанын көрсеткен тәжірибе жүргізілді. Фаган (1986) 60%-дан астам қателерден бағдарламаның тексерулерін үйреншікті емес көмекпен білдіруге болатынын хабарлады. Тазарту бөлмесінде (Prowell et al., 1999) 90%-дан астам ақаулардан тексерулерді бағдарламаға білдіре алғанын бекітті. Дегенмен, инспекция бағдарламалық қамтамасыздандырудың сынағын өзгерте алмайды.

Әйтсе де, инспекция бағдарламалық қамтамасыздандыру сынауын алмастыра алмайды. Уақытша мәселелер немесе жүйе өнімділігінің мәселесі

арасындағы күтпеген әрекеттесуде пайда болған ақауларды табу үшін инспекция жақсы емес. Одан басқа, шағын серіктестіктерінде немесе даму топтарында, сонымен қатар бәрі потенциалдық топ мүшелерін, әсіресе, бағдарламалық қамтамасыздандырудың өндеушілерін біріктіру қиын және қымбат болуы мүмкін. Мен (сапа менеджментінің жүйесі) *24-бөлімде* пікірлерді және тексерулерді толығырақ талқылаймын. Бағдарламалық түпнұсқаның аномалиясын білдіру үшін автоматты түрде талдау *15-бөлімде* ұғындырылған. Мен осы тарауда үдерістерді сынау және сынауға жұмылдырамын.

8.3-сурет сынаудың «дәстүрлі» үдерістерінен әзірлеу жоспарында пайданылатын абстрактілі моделін көрсетеді. Сынаққа кіре берісте сынақтық деректердің жиынтығын және сынақтан өтетін туралы (тестердің нәтижелері) жүйеден шығу, сонымен бірге мәлімдеме күтіледі. Сынақтық деректер іс жүйенің кіре берістерін тексеруі үшін жасалған. Сынақ деректері кейде автоматты түрде шығара алады, өйткені адамдар түсінгендіктен жүйе күтілген сынақ нәтижелерін тарту керек, бірақ сынақтың автоматты генерация болуы мүмкін емес. Алайда, тестің орындауы мүмкіндігі автоматталған болуы мүмкін. Күтілген нәтижелер болжанған нәтижелермен автоматты түрде теңестіріледі, дегенмен, адамдарға сынақ орындауынан қате іздеудің қажеті жоқ.



8.3-сурет. Бағдарламаны тестілеу үдерісінің үлгілері

Әдеттегідей, жүйенің коммерциялық бағдарламалық қамтамасыздандыруы сынақтың төменде көрсетілген үш кезеңінен өтуі керек:

1. Жүйені зерттеу үдерісінде қателерді және ақауларды табудағы сынау дамуы. Сірә, жүйе дизайнерлері және бағдарламашылары сынақтың үдерісіне тартылған болады.
2. Сынақтың шығаруы, сынақтың жеке топ пайдаланушылар үшін оның шығаруына дейін жүйенің толық нобайын тексереді. Сынақ шығаруының мақсаты, жүйе, мүдделі тараптардың талаптарға жауап беретінін тексеру.
3. Пайдаланушылық сынақ, пайдаланушылар немесе потенциалды пайдаланушылардың бір ортада болуы. Егер бағдарламалық қамтамасыздандыру сатылса, бағдарламалық қамтамасыздандырудың қарым-қатынасын шешетін ішкі маркетинг тобы “пайдаланушыны” азат етеді. Егер жүйеден қабылдануы керек болса немесе жеткізуші одан әрі ғылыми әзірлемелер талап етсе, қабылдау сынақтары үстірт жүйені шешуге тексеретін пайдаланушыларға сынау түрлерінің бірі болып көрінеді.

Сынақ үдерісі тәжірибеде әдетте, қол және автоматталған сынақты қосып алады. Қол сынағында, тексеруші кейбір сынақтық деректермен бағдарламасын іске қосады және олардың нәтижелерді теңестіреді. Олар белгі салады және сәйкессіздікті өңдеушілердің бағдарламасында хабарлайды. Автоматталған сынақта, сынақ әрбір жасалатын жүйе тексеруінде орындалған программада код-талады. Регрессиялық сынақ сөз болғанда, бағдарламаға өзгеріс енгізу үшін ол қол сынағына қарағанда тезірек.

Автоматталған сынауды пайдалану соңғы бірнеше жылда едәуір өсті. Алайда, сынақ ешқашан автоматтандырылған болмау керек, тек қана бағдарламаның жұмысын тексере алады. Автоматтандырылған сынауды жүйені тексеруге қолдану мүмкін емес.

Себебі олар мысалға пайдаланушының графикалық интерфейсінің қалай көрінетіндігіне немесе қосымша эффектілері болмауын тексеруіне тәуелді.

8.1. Сынақты дамыту

Сынақтың дамуы сынаудың командамен жүзеге асырылатын барлық түрін өзіне қосады. Әдеттегідей бағдарламалық қамтамасыздандырудың тексерушісі, бағдарламалық қамтамасыздандыруды жасаған бағдарламашы болып табылады, бір жағынан әрқашан бұлай емес. Кейбір даму үдерістері бағдарламашы/тексеруші жұбын (Cusamano and Selby, 1998) қолданады және сынақтың үдерісінде көмектеседі. Сын көзге маңызды жүйелер үшін, үстірт үдеріс, салушының командасының құрамында, сынақтың жеке тобымен қолданылуы мүмкін. Олар сынақтардың әзірлігіне жауап береді және сынау нәтижелерінің толық есептеуін жүргізеді.

Әзірлеу кезінде, сынақты уақыт тәптіштеудің үш деңгейінде өткізе алады:

1. Модульдік сынақ, жеке программалық модульдерді немесе класс объектілерін сынайды. Модульдік сынақ объектілердің немесе әдістердің функционалдығын сынауға түйілігуі керек.
2. Құрама құрауыштарды жасау үшін интегралданған бірнеше жеке бірліктерінің құрауыштарын тексеру. Құрауыштарды тексеру құрауыштардың интерфейстерінің сынауына жұмылдыруы керек.
3. Жүйенің кейбір немесе барлық құрауыштары интегралданған болса және сынақтың жүйесі бүтін болса,сынақ жүйесі деп аталады. Жүйенің сынағы өзара құрама іс-әрекеттерді сынауға бағытталу керек.

Сынақтың дамуы – ең алдымен, бағдарламалық қамтамасыздандырудан ақауларды табу мақсаты болып табылатын ақауларды сынау үдерісі.

Күйін келтіру

Күйін келтіру сынақ барысында анықталған қателер мен мәселелердің алдын алу үдерісі болып табылады. Бағдарламалық сынақтардан алынған ақпаратты пайдалана отырып күйін келтіруші мамандар бағдарлама қатесінің орнын анықтау мен қатенің алдын алу үшін сынақ нәтижелерінің нұсқаулығы бойынша бағдарламалау тіліне қатысты білімдерін пайдаланады.

http://www.SoftwareEngineering-9.com/Web/Testing/Debugging.html

8.1.1. Модульдік сынау

Программалық құрауыштарды сынаудың үдерісін, әдісін немесе объектілерін модулдік тестілеу дейді. Жеке функциялар немесе әдістер құрауыштың ең оңай түрі болып көрінеді. Сынақ әртүрлі ену параметрлері бар, бағыныңқы программалардың шақыруларынан болуы керек. Сіз әрлендіру жағдайын сынаққа тәсіл етіп пайдалана аласыз, функцияның әзірлеуі үшін сынақтар әдісі *8.1.2-бөлімде* талқыланады.

Сіз класс объектілерін сынау кезінде, барлық ерекшеліктерін қамтумен қамтамасыз ету үшін, сынақты өндеу керек. Сіз ол үшін:

- объектіге қатысты операциялардың бәрін тексеруіңіз;
- объектіге қатысты барлық төлсипаттардың мағынасын тексеру және орнатуыңыз;
- барлық мүмкін күйлерде объектіге енгізуіңіз керек. Күйдің өзгерісін шақырған барлық оқиғаларына сіз үлгі жасауыңыз керек екендігін білдіреді.

Мысалы, мен *7-бөлімде* айтқан метеобекетінің объектісі туралы мысалды қарастырайық. Бұл объектінің интерфейсі *8.4-суретте* көрсетілген. Ол оның идентификаторы болып көрінген жалғыз төлсипаты болып табылады. Бұл орнатылған, метеобекетінің жанында бекітілетін, тұрақты шама. Қорыта келгенде, егер ол дұрыс болса, сіз тексеретін сынаққа мән бергеніңіз жөн. Сізге барлық затқа қатысты әдістерді анықтау керек, мысалы, reportWeather, ReportStatus және т.б. Сіз оқшаулаған әдістерді тексеруіңіз керек, ал кейбір жағдайларда кейбір сынақтық тізбектер қажет. Мысалы, (ажырату) құралдарды метеобекетте аяқталған әдістің тестілеуі үшін, сізде қайта іске қосу әдісі атқарылған болуы керек.

Жалпылау немесе мирас, класс объектісін сынауды күрделірек етеді. Сіз анықталынған класстың жұмыс істеуін қарапайым тексере алмайсыз және мирас ететін операциялар класс тармақтарында күтілген жұмыс істегенін көрсетеді. Басқа операциялар және төлсипаттар туралы мұра ететін операциялар тұспалдауы мүмкін. Олар мұра ететін операциялардың кейбір класс тармақтарында нақты бола

алмайды. Қорыта келгенде, сіз оны пайдаланатындығынан барлық контекстерде мұра етілген операцияны тексеруіңіз керек.

8.4-сурет. Ауа райы бекетіні нысандарының интерфейсі

Метеобекеттегі күйді тексеруі үшін, сіз алдыңғы бөлімдегі *7.8-суретте* көрсетілгендей, күйдің үлгісін пайдаланыңыз. Тексеруі керек болған күйлердің арасындағы өткелдерді тізбектеп анықтау керек, ол үшін бұл үлгіні қолданасыз. Қағидатта, сіз мүмкін өтпелі күйдің тізбектерінің бәрін тексеруіңіз керек, бір жағынан, бұл тәжірибе тым қымбат болуы мүмкін. Метеобекетте тексеруі керек күйлердің тізбектерінің мысалдары:

Сөндіру → Қосу → Сөндіру;
Баптау → Қосу → Сынау → Жіберу → Қосу;
Қосу → Жинау → Қосу → Жинақтау → Жіберу → Қосу;

Әрқашан мүмкін болғанда, модульдік сынауды автоматтандыру керек. Автоматтандырылған блок сынауында, сіз автоматтандырылған сынау жақтауына (мысалы, JUnit) жазуды пайдаланыңыз және программаның сынақтарын орындаңыз. Жақтаудың модульдік сынауы нақты сынақтардың жасауына тарайтын әмбебап сынақ кластарын қамтамасыз етеді. Олар кейбір графикалық интерфейске, жетістікке немесе сынақтардың құлауына енгізілген барлық сынақтарды іске қоса алады. Сынақтардың барлық теруін бірнеше секундтарда іске қосуға болады, дегенмен, сынақтың бәріне өзгеріс енгізгеннен кейін де орындауға болады.

Автоматталған сынақ:

1. Сынақта жүйенің инициализациялауында, кіру және шығу нәтижелерінің, бөліктің орнатуы.
2. Сіз объект немесе сынақ әдісі деп атаған бөлікті шақыруы.
3. Сіз күтілген нәтижемен шыққан нәтижесі салыстырғаныңызды бекіту бөлігі сияқты үш бөліктерден тұрады. Егер бекіту шынайы болса, тест табысты келеді; егер жалған болса, онда ол сәтті болмайды.

Егер олар пайдаланылса, кейде сіз сынақта жаза алмаған басқа объектілерден тәуелділікке ие болған объектіні немесе тестілеудің үдерісін баяулатады. Мыса-

лы, егер сіздің объектіңіз баптаудың ақырын үдерісінен өзімен еліктірген дерекқор шақырса, ол бұрынғыдан көп жұмсалу мүмкін. Сіз бұл жағдайларда жалған объектілерді пайдаланып шеше аласыз. Жалған объектілердің функционалдығы пайдаланылатын сыртқы объект дәл келтірген интерфейсіндей объект болып көрінеді. Қорыта келгенде, деректерді пішіндеу ашық объектісінің макеті массив түрінде ұйымдастырылған деректердің бірнеше элементтері мүмкіндігіне ие болады. Демек, үстеме шығыс пен дискілерге қол жеткізбей, деректерді тез шақыруға болады. Одан басқа, жалған объектілер дұрыс емес жұмысқа немесе сирек оқиғаларға пішіндеу үшін жұмсала алады. Мысалы, егер сіздің жүйеңіз мөлшерлі уақытқа қабылдауға арналса, сағаттағы нақты уақытқа тәуелсіз, сіздің объектіңіздің макеті қарапайым сол уақытты қайтара алады.

8.1.2. Модульдік сынау жағдайларын таңдау

Сынау қымбат және сыйымды болып көрінеді, сондықтан сіз тиімді сынақтық құрылғыларды таңдағаныңыз маңызды. Бұл жағдайда тиімділік екі болжамды білдіреді:

1. Сынақтардың күтілген сапасын пайдалануда көрсететін құрауыш көрсету керек;
2. Егер құрауышта ақаулар болса, оларды сынақты анықтау керек.

Сондықтан сіз сынақтың екі түрін жазуыңыз керек. Біріншісі программаның қалыпты жұмысын бейнелеп көрсетуіңіз керек және құрауыш жұмыс істеп тұрғанын көрсетуіңіз керек. Мысалы, егерде сіз емделуші мен жаңа жазба аты-жөнін көрсететін құрауышты сынаудан өткізсеңіз, онда сіздің сынауыңыз дерекқорда жазба болғанын, айтылғандай орнатылғанын көрсетуі керек. Сынақтардың басқа түрі ортақ мәселелер пайда болған сынаудың тәжірибесіне негізделген болуы керек. Дұрыс емес кіре берістерді колданып, олардың керекті мөлшерде өңделгенін тексеруге пайдалануы керек.

Мен сізге сынақ таңдауға көмектесетін екі мүмкін тиімді болған стратегияларды талқылаймын. Оларға төмендегілер жатады:

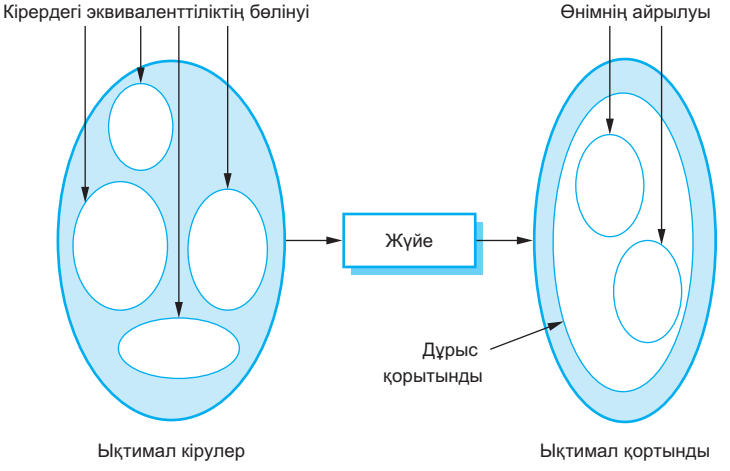
1. Сынаудың бөлімі, сізге ортақ сипаттамалар ие болған кіре берістер тобын анықтауға және осы сияқты өңдеулерге керек. Сіз бұл топтардың әрқайсысына сынақ таңдауыңыз керек.
2. Сынаудың негізіндегі басшылық деген сіз сынауды таңдауға кепілдемені пайдаланатыныңыз. Бұл басқарушы қағидалар құрауыштарды әзірлеудің жанында программалаушылар жиі жасаған қателердің түрлерін, алдыңғы тәжірибені бейнелеп көрсетеді.

Кіретін деректер және бағдарлама нәтижелерінің шығуы ортақ сипаттамалар мен әртүрлі кластарлы қатарларға жиі тап болады. Оң сандар, теріс сандар және

меню таңдау сол кластардың мысалдары. Программа кластың барлық мүшелері үшін салыстырылатын жолда әдетте, өзін-өзі ұстайды. Яғни егер сіз есептеу жасаған программаны сынақтан өткізсеңіз және екі оң сандарды талап етсе, онда осы сияқты барлық оң сандар үшін өзін-өзі ұстайды.

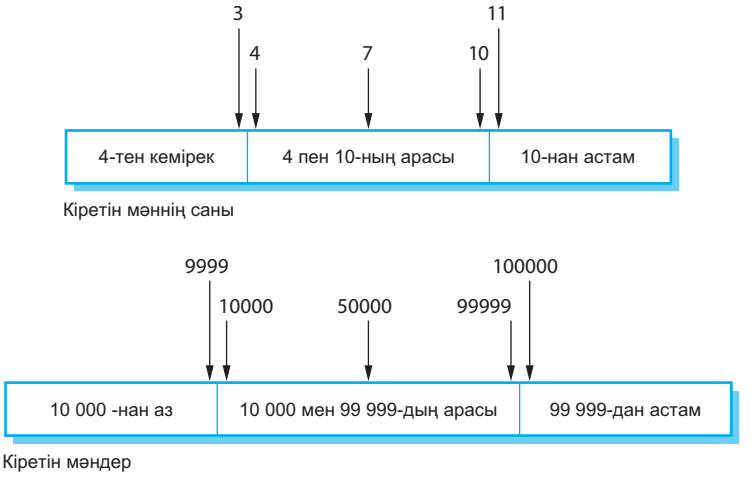
Эквивалентті тәртіп үшін, бұл кластар артынан эквивалентті бөлімдер кейде облыстар (Bezier, 1990) деп аталады. Әрлендіру жағдайдың сынаққа бір жүйелік көзқарас үшін, бөлімді анықтауда барлық кіріс және шығыс құрауыштар негізделген. Сынақтық сценарий кіру және шығу бөлімдерінің шектеулерімен жасалған. Сынаудың бөлімі екі жүйе үшін сынақтың әзірлеуі және құрауыштары үшін жұмсала алады.

8.5-суретте үлкен көлеңкелі эллипс барлық мүмкін сынақ программаға кіретін деректерін сол жағында таныстырады. Кем штрихталмаған эллипстер эквиваленттілікті таныстырады. Сыналатын программа кіретін эквиваленттің мүшелерінің бәрін жұмыстандыруы керек. Шығу эквиваленттілігі шығу бәріне бірдеңе ортақ болған бөлімдерге ие болады. Кейде кіру мен шығудың эквиваленттілігінің арасын 1:1 бейнелеу бөледі. Алайда, әрқашан бұлай емес, мүмкін,басқа кіру эквиваленттілігін табу керек. Сол эллипсте облыс болып көрінетін ретсіз кіре-беріс таныстырылады.



8.5-сурет. Ұқсастықтарды бөлу

Сіз кейін бөлімдерді теріп болғасын, сынақ бөлімдерінің біреуін таңдаңыз. Сынақтың таңдауы үшін жағдайдың жақсы эмпирикалық ереже бөлімдерінің шектерінде сынаудың таңдауы және бөлімнің жақын орта нүктеге жағдайлары болып көрінеді. Бұның себебі дизайнер және жүйенің әзірлеуінің жанында кіре-берістердің типті мағыналары есептеген бағдарламалаушылар болып көрінеді. Сіз олардың бөлім орта нүктесін таңдап тексересіз. Шекаралық мәндер жиі ұқсамайды (мысалы, басқа теріс емес сандар қарағанда, нөл өзін хабарлауы мүмкін), сондықтан өңдеушілер оларды кейде ұмытып кетеді. Бас-сирақтардың бағдарламасы бұл ұқсамаушылық мағыналарды өңдеудің жанында жиі болады.



8.6-сурет. Ұқсастықтар бөліктері

Сіз қателерді болжайтын, кластардың кіру мағынасын бағдарлама мен спецификацияны немесе құжаттаманы қолданып бөлімдерді анықтай аласыз. Мысалға айталық, бағдарламаның спецификациялы ортасында, кіре-берісте 4 пен 8 аралығында қабылдайтын бес таңбалы 10000-нан артық бүтін сандар орналастырылған. Сіз кіретін бөлімдерді және мүмкін тестің кіретін мағыналарын анықтау үшін бұл ақпаратты пайдалана аласыз. Олар *8.6-суретте* көрсетілген.

Эквиваленттілікті анықтау үшін жүйенің спецификациясының пайдалануында "тестілеу қара жәшік" деп аталатынды ойрандайды. Мұнда, сізге жүйенің қалай жұмыс істейтіні туралы ешқандай білімдер қажет емес. Алайда, сіз бағдарламаның кодына қарағанда, «ақ жәшікті сынау» сынақтары қара жәшіктегі қосымшаға пайдалы болуы мүмкін, басқа сынақтарды табуыңыз мүмкін. Мысалға, сіздің кодыңыз дұрыс емес кіре-берістерді өңдеу ерекшеліктері үшін болады. Сіз бұл білімді "бөлімдерді шығару" ерекшеліктерін өңдеуде қолдану керек болған жағдайда әртүрлі ауқымдағыларды анықтау үшін пайдалана аласыз.

Эквивалентті бөліктеу сынауға тиімді тәсіл болып көрінеді, себебі ол, бөлімдердің шеттеріндегі кіру өңдеуінің жанында бағдарламалаушылар жиі жасаған қателерді есептеуге көмектеседі. Сіз сынақтарды таңдауға көмектесетін кепілдеменің сынауын пайдалана аласыз. Білімдердің инкапсуляциясының басқарушы қағидасы сынақтардың қателерін табу үшін тиімді болып көрінеді. Мысалы, тізбектермен бағдарламалық сынауда, массив немесе тізімдер, ақау анықтауға көмектесе алады және басқарушы қағидалардан тұрады:

1. Бір ғана мағынаға ие болған тізбектермен сынақтық бағдарламаларды қамтамасыздандыру. Программалаушылар тізбек туралы ойлайды, себебі бірнеше мағыналардан тұрады, ал кейде оларды өз бағдарламаларына

қояды. Демек, тізбектің бір мағынасымен кездескенде, бағдарлама керекті мөлшерде жұмыс істемейді.
2. Әртүрлі сынақтарда әртүрлі өлшемді, әртүрлі тізбектерді пайдаланыңыз. Бұл ақаулармен бағдарламаға кіретін сигналды, кейбір кездейсоқ сипаттамаларды, артынан дұрыс қорытындыны өндіретін ықтималдықты кішірейтеді.
3. Сынақтарды бірінші, ортасыншы және соңғы элементтердің тізбегінен алып шығару. Бұл тәсіл бөлім шектеріндегі мәселесін анықтайды.

Жолға сынақ жүргізу

Жолға сынақ жүргізу компоненттен немесе бағдарламадан өтетін әрбір жеке атқару жолын жаттықтыруға мақсатталған сынақ стратегиясы болып табылады. Егер жеке жол атқарылса, онда компоненттегі нұсқаулықтардың барлығы кем дегенде бір мәрте орындалып шығуы тиіс.Барлық шартты нұсқаулықтар шын және жалған жағдайларда сынақтан өткізіледі. Нысанға бағытталған өңдеу үдерісінде, жолға сынақ жүргізу әрекеті нысандарға байланысты әдістерді сынақтан өткізу кезінде орындалуы мүмкін.

http://www.SoftwareEngineering-9.com/Web/Testing/PathTest.html

Уиттакер (2002) кітабында сынау әрлендіруге жұмсалатын көп кепілдемелердің мысалдарын көрсетеді. Ол ұсынған:

- Барлық қателік туралы хабарларды алу үшін кіре-берісті таңдаңыз;
- Кіре-берісті әрлендіру кіре-беріс буферлерінің толып кетуін шақырады;
- Енгізуді немесе топтаманы бірнеше рет қайталаңыз;
- Құрылатын жарамсыз шығуларды қою;
- Есептеу күшінің нәтижелері өте үлкен немесе тым аз болуы сияқты кепілдеменің кейбірі белгілі болып көрінеді.

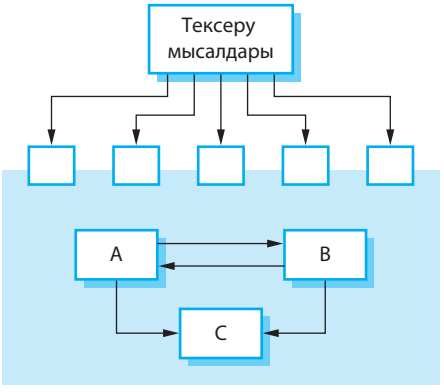
Сынаудың тәжірибесінің өлшем бойымен алуын, сіз тиімді сынақтарды таңдаған сияқты басқарушы қағиданы меншікті жасай аласыз. Мен осы бөлімнің келесі тарауында сынау бойымен кепілдеменің мысалдарын артық беремін.

8.1.3. Құрауыштарды тексеру

Программалық құрауыштар жиі өзара жұмыс істейтін құрама құрауыштардан, бірнеше объектілерден тұрады. Мысалы, реконфигурация құрауышы ауа райы станцияның жүйесінде, реконфигурацияның аспектісімен істейтін объектіге қосады. Сіз бұл объектілердің функционалдығына интерфейстің белгілі құрауышы арқылы рұқсат аласыз. Құрама құрауыштарды сынау интерфейстің құрауышы оның спецификациясымен сәйкестікте өзін-өзі ұстауын көрсетуге жұмылдыруы

керек. Бұл құрауышында сынақтар істерінің біткенін жеке объектілерінен болжауға болады.

8.7-сурет интерфейстің құрауыштарын сынаудың ойын суретімен көркемдейді.



8.7-сурет. Интерфейстерті тестілеу

Үлкен құрауыш немесе ішкі жүйені құру үшін, А, Ә және Б құрауыштарын топтастырып болжаймыз. Сынақ жеке құрауыштарға қолданылылмайды, ал композициялық материал құрауышын интерфейсіне бұл құрауыштарды біріктіру жолымен жасайды. Құрама құрауыштағы қатенің интерфейсі жеке объектілерді тексеруден табылуы мүмкін емес жолымен болады, өйткені олар – құрауыштағы объектілердің арасындағы өзара іс-әрекеттің нәтижесі.

Бағдарламалық құрауыштардың арасындағы өзара іс-әрекеттің әртүрлі түрлері болады, демек, әртүрлі интерфейс қателері пайда болуы мүмкін:

1. *Интерфейстердің баптаулары.* Бұл деректі интерфейстер, ал кейде функциядағы сілтеме басқаға бір құрауыштардан алып береді. Объектінің әдістерінде интерфейс баптаулары бар.
2. *Жадтың ортақ интерфейсі.* Бұл – барлық құрауыштардың арасындағы жад блок интерфейсі. Деректер бір ішкі жүйе бойымен жадта сыйғызылып салынады және ол жақтан басқа ішкі жүйелермен шығарылады. Бұл интерфейстің түрі кіріктірме жүйелерде жиі пайдаланылады, көрсеткіштері алынған деректерді жүйенің басқа құрауыштарымен өңдейді.
3. *Процедура интерфейстері.* Бұл – бір құрауышқа теруді инкапсуляциялаған, басқа құрауыштармен шақырыла алған процедура интерфейстері. Объект және қайтадан пайдаланылатын құрауыштар интерфейстің бұл пішініне ие болады.
4. *Хабар беру интерфейстері.* Бұл – бір құрауышқа хабар алу арқылы басқа құрауыштан қызмет беруге сұрау салатын интерфейстер. Жауап хабар қызметтің орындауының нәтижелерінен тұрады. Кейбір объектіге бағдарланған жүйелер интерфейстің бұл пішінін алады.

Қатенің интерфейсі қате пішіндерінің күрделі жүйеге кең таралғандардың (Lutz, 1993) бірі болып табылады. Бұл қателерді үш класқа бөлуге болады:

- Қиянатшы интерфейс. Шақыратын құрауыш кейбір басқа құрауыштарды шақырады және интерфейс оны пайдалануда қате жібереді. Бұл қатенің түрі ортақ баптаулардың теріс түрі бола алғандығынан интерфейстердің баптаулары болып көрінеді немесе берілген теріс ретінде болады немесе баптауларының қате саны берілген бола алады.
- Қателік интерфейсі. Шақыратын құрауыш шақырылатын құрауыш интерфейсінің спецификациясын теріс түсінеді және оның тәртібіне қатысты тұспалдайды. Құрауышы күтілгендей деп шақыратын құрауышта күтпеген жайт тәртіпке содан соң алып келмейді. Мысалы, іздеудің екілік әдісі реттелген емес массив болатын баптаулармен шақырылуы мүмкін.
- Уақытша қателер. Олар ортақ жадтарды пайдаланатын нақты жүйе уақыты немесе хабар беру интерфейсі болады. Деректер өндіруші және деректерді тұтынушы әртүрлі жылдамдықтарда жұмыс істей алады. Егер ерекше ілтипат интерфейстің әрлендіруіне бөлінбесе, тұтынушы әлдеқашанғы ақпаратқа қол жеткізу мүмкіндігін алады, себебі өндірушісі туралы ақпараттың жаңартпа интерфейсі таралған.

Ақауды интерфейске сынау қиын, өйткені ақаулықтың кейбір интерфейсі ғана ерекше шарттарды көрсете алады. Мысалы айталық, объект кезекті деректердің құрылымының белгіленген ұзындығын жүзеге асырады. Шақыратын объекті кезекті деректердің шексіз құрылымды түрде өткізілгенін элемент енгізілгенде есептеуге болады және кезектің толып кетуін тексеруі керек. Бұл шарт кезектің толып кетуіне мәжбүрлеген сынақтардың әзірлеуіне ғана уақыт табылуы мүмкін және жолымен табылатын объектілердің тәртібі кейбір дұрыс емес жолға түсірген толып кетуді шақырады.

Мәселе, тағы бір ақау әртүрлі модульдердің немесе объектілердің аралығында өзара іс-әрекетінен пайда болуы мүмкін. Бір объектідегі бұзылу басқа объект өзін күтпеген жайтта ұстаса пайда болуы мүмкін. Мысалы, объект басқа объектіні шақырып, кейбір қызметтерді алуы мүмкін және жауапты дұрыс деп болжаймыз. Егер ақаудың шақырылатын қызмет көрсетуі кейбір түрде, қайтарылатын мағынасы нақты болуы мүмкін. Бұл бірден табылмайды, кейбір есептеулер дұрыс жасалмағанда көріне бастайды. Интерфейстің сынауы бойымен кейбір ортақ кепілдемелер төмендегідей:

1. Тексерілген кодты зертеу және сыртқы құрауыштың барлық шақыруларын анық санап шығу. Сынауды терудің әзірленуі сыртқы құрауыштардың баптауларын мағынасының қарама-қарсы ауқымының соңдарында болатын көрсетеді. Бұл экстремальды мәндер, интерфейстің сәйкессіздігін анықтайды.
2. Нұсқағыш интерфейс арқылы алып беретін жерді, нұсқағышты нөлдік баптаулары бар интерфейсті әрқашан тексеру.

3. Құрауыш процедура интерфейсі арқылы шақырылса, сынақ әрлендіруі құрдымға кетеді. Жеткіліксіздіктің әртүрлі жорамалдары спецификацияның қателіктерінің кең таралған түрінің бірі болып табылады.
4. Хабарларды тарату жүйесінде сынау стресін пайдалану. Сірә, бұл хабарларды көп өндірген сынақ өндеу керектігін білдіреді. Бұл мәселенің анықтаудың уақытша тиімді тәсілі.
5. Егер бірнеше құрауыштар бөлетін жадпен өзара әрекет етсе, ретті өзгертетін конструкторлық сынақтар қосылады. Бұл сынақтар бағдарламалаушы жасаған деректерді бөліп, өндірген реттегі анықталмаған жорамалдары анықтай алады.

Инкементтік инеграция және сынақ

Жүйе сынағы сіз жасап шығарған біріккен жүйе сынағына қарағанда біріккен түрлі компоненттерді қамтиды. Сіз инеграция мен сынақ үшін (мысалы, сізге компоненттерді біріктіру, жүйеге сынақ жүргізу, басқа да компоненттерді біріктіру, қайтадан сынақ жүргізу және тағы сол сияқты әрекеттерді орындау қажет) әрдайым инкременттік тәсілді пайдалануыңыз қажет. Бұл дегеніңіз, егер мәселе туындаса, онда ол ең жуырда біріктірілген компонентпен ара қатынас салдарынан болуы мүмкін дегенді білдіреді. Инкременттік интеграция және сынақ әр уақытта жаңа инкементті біріктіріп отыратын регрессия сынағы бар XP сияқты жылдам және икемді әдістеріне фундаменталды болып келеді.

http://www.SoftwareEngineering-9.com/Web/Testing/Integration.html

Интерфейстің қатесін табылуға арналған сынауға қарағанда, тексеру кейде тиімдірек болуы мүмкін. Тексеру құрауыштарды интерфейстерге түйіткі болады және интерфейстің шамаланған тәртібі туралы сұрақ үдерістің тексеруіне уақыт сұрайды. Қатал типтелген тілдер Java интерфейстің көп қателеріне, компилятор байқап қалуға рұқсат береді. Статикалық анализатор (*15-бөлімді* қараңыз) интерфейс қатесінің кең диапазонын табуы мүмкін.

8.1.4. Жүйе сынағы

Даму уақытында жүйенің сынағы жүйенің түрін жасау үшін құрауыштарды ықпалдаса қосады, ал интегралданған жүйені содан соң тексереді. Жүйе сынауын тексеру үйлесімді құрауыштары дұрыс өзара әрекет етсе, интерфейс олар арқылы деректерінің құқығын алып береді. Анық, бұл сынаудың құрауышымен дәл келеді, бірақ екі маңызды өзгеліктер бар:

1. Жүйені сынау кезінде, тыс жүйесінің полктері интегралданып, игерілген құрауыштарымен қайта пайдаланылады. Барлық жүйе соңынан сыналады.

2. Құрауыштар топтардың әртүрлі мүшелерімен жасалады немесе топтар осы кезеңде интегралданудан өтеді. Жүйенің тестілеуі ұжымдық болып көрінеді, ал дара үдеріс емес. Кейбір компанияларда сынау жүйесі сынаудың әрлендірушісіз және бағдарламалаушыларсыз жеке командасын қосуы мүмкін.

Жүйе жасауда, құрауыштардың ықпалдасу кезінде, сіз пайда болатын тәртіптерді алыңыз. Құрауышты сіз салғанда, бұл жүйенің функционалдығының кейбір элементтерінің ғана анық болатынын білдіреді. Бұл пайда болатын жоспарланған тәртіп тексерілуі керек. Мысалы, сіз ақпарат жаңартатын құрауышы бар аутентификацияның құрауышын интегралдай аласыз. Содан кейін сізде қосылған пайдаланушылар үшін ақпараттың жаңарту функциясына шек қояды. Алайда, кейде пайда болатын тәртіптер жоспарланбаған және жағымсыз болады. Сіз жүйенің жасайтынын тексеретін сынақ жасауыңыз керек.

Сондықтан жүйенің сынауына ыңғайлануы үшін құрауыштардың жүйені құраған объектілермен арасындағы өзара іс-әрекеттерді сынайды. Сіз олардың қалай күтіліп, жаңа құрауыштармен интегралданғанын, жұмыс істегенін тексеру үшін, қайтадан пайдаланылатын құрауыштарды немесе жүйені тексере аласыз. Бұл сынаудың өзара іс-әрекеті құрауыштың басқа жүйеде қолданғанда пайда болатын қателерді табу керек. Сынаудың өзара іс-әрекеті құрауыштардың өңдеушілерімен жүйенің басқа құрауыштары туралы жасалған қателікті табуға көмектеседі.

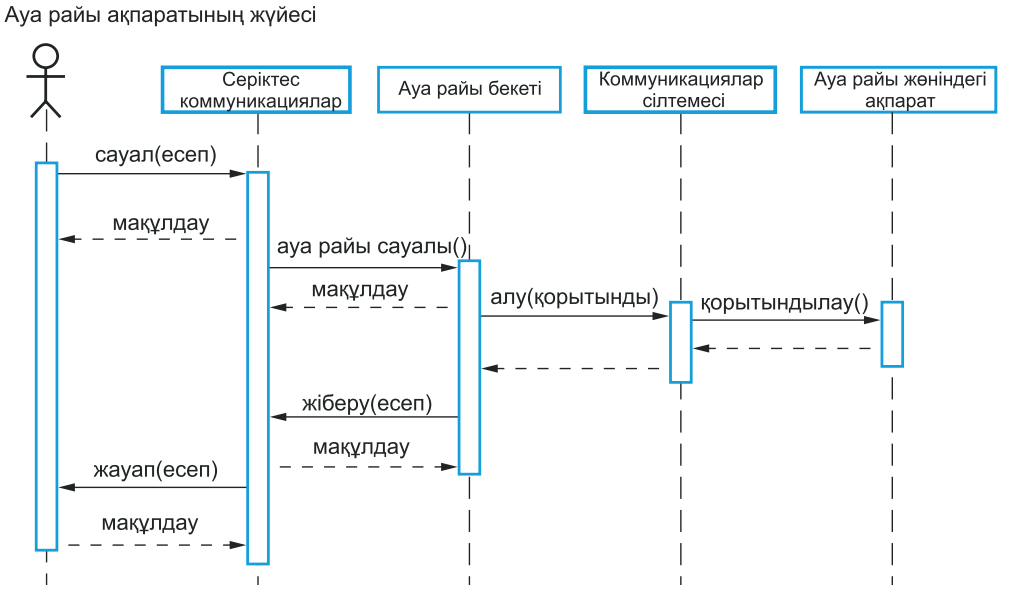
Сынақтың негізінде өзара іс-әрекетінің прецеңттеріне, оның акценті артынан жүйенің сынауына тиімді тәсіл болып көрінеді. Әдеттегідей, әр пайдаланудың нұсқасы бірнеше құрауыштарды немесе объектілерді жүйеде жүзеге асырады. Күш нұсқаларының пайдалануының сынауы арқылы өзара іс-әрекеттер болады. Егер сіз пайдаланудың нұсқасын өткізудің пішіндеуі үшін тізбектің диаграммасын жасасаңыз, сіз объектіні немесе өзара іс-әрекетте қатысқан құрауыштарды көре аласыз.

Бұл мысалды келтіру үшін, мен метеобекет алшақ компьютерге ауа райының қорытынды деректерлерін есептеп таныстыруға ұсынған жүйе туралы мысалды пайдаланамын. Ол үшін прецендент *7.3-суретте* таныстырылған (алдыңғы бөлімді қараңыз). *8.8-суретте* (*7.7-суреттің* көшірмесі болып табылады) метеобекеттегі операциялар тізбегі көрсетілген, жүйенің бейнелеуі үшін деректер жиын сауалға жауап береді. Сіз сынақтардың орындауы үшін, сынақтардың әзірлеуінде жәрдем берген операцияларды анықтау үшін бұл сұлбаны пайдалана аласыз. Қорыта келгенде, сауал берудің есебі әдістеріне:

SatComms:request → WeatherStation:reportWeather → Commslink: Get(қорытынды) → WeatherData:summarize желілерінің орындауына әкеледі.

Тізбектің диаграммасы сізге қажетті нақты сынақтарды жасауға көмектеседі, себебі бұның ақпарат енгізуі және шығуы үшін көрсетеді:

1. Есеп сауалды енгізуі үшін тиісті растау алуы керек. Түпкі есепте есеп сауалдан қайтарылуы керек. Сынау уақытында, сіз баяндамада дұрыс ұйымдастырылған тексеру үшін жұмсалынған, қорытылған деректер құруыңыз керек.
2. Құрама есептеу нәтижесіне баяндама WeatherStation нәтижелері үшін деректерді енгізуге сауалды өндіреді. Сіз оқшаулауда, сынақ SatComms үшін қорытындыға тиісті бастапқы деректер жасап тексере аласыз және WeatherStation объекті дұрыс қорытындыны өндіруін даярлайсыз. Бастапқы деректер WeatherData объекті тексеруі үшін пайдаланады.



8.8-сурет. Ауа райы деректер жинағын кескіні

Әрине, мен *8.8-суреттегі* циклограмманы ықшамдадым, сондықтан ол шығаруды көрсетпейді. Сынақтық сценарийдің толық нұсқасын назарға қабылдауы керек және объект шығаруды дұрыс жұмыс жасайтынына кепілдік беру керек.

Көптеген жүйелер үшін, жүйенің сынағы сынақты қашан тоқтатуыңыз керектігін көрсетеді. Әр мүмкін бағдарламаның орындауын тізбек жан-жақты тексеріледі. Демек, сынақ мүмкін сынақтардың ішкі жиынына негізделуі керек. Мұратта, компания өндеушілері бұл ішкі жиынның таңдауы үшін саясат алуы керек. Бұл саясат сынаудың ортақ саясатында негізделуі мүмкін, бағдарламасының операторлары бойымен атқарылған мәжбүрлік қажеттілік не бір рет болуы керек. Одан басқа, олар жүйенің пайдалануының тәжірибесіне негізделе алады және басқару жүйесінің ерекшеліктерін сынауда көрсетеді. Мысалы:

1. Меню арқылы қол жететін жүйелік функциялардың бәрі тексерілу керек;

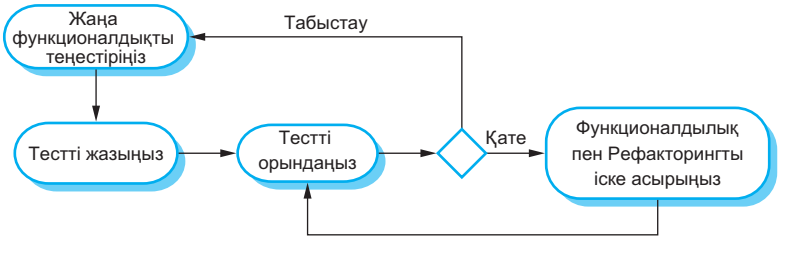
2. Функциялардың тіркесімдерінің меню арқылы тексерілуі қолжетімді болу керек;
3. Адрестерді қолмен енгізу кезінде, мәтін тексеру функцисы жұмыс істеу керек.

Бағдарламалық қамтамасыздандырудың негізгі өнімдері бар тәжірибе көрініп тұрғаннан, мәтіндік процессор сондай және ұқсас басқарушы қағидалар өнімнің сыналуында әдетте, пайдаланатын электрондық кестелер. Бағдарламалық қамтамасыздандырудың мүмкіндігінің жанында оқшауланып пайдаланады, олар қалыпты жұмыс істейді. Ол кең пайдаланылатын мәтіндік процессорға, көп бағаналы макет нұсқамасын қолдана отырып, мәтіннің теріс орналастырылуына алып келетін мысал келтіреді.

Әдеттегідей сынаудың автоматтандырылған жүйесі автоматталған торап немесе тестілеудің құрауышына қарағанда күрделірек. Құрылғының автоматталған сынау бағдарламада, содан соң кодтайтын болжаулардың шығуын болжауға міндеттеледі. Содан соң болжаммен нәтиже теңестіріледі. Сіз қорытынды зерттеу жағдайын көтере аласыз және оның абыройы міндетті түрде емес, оны алдын ала құруға тексересіз.

8.2. Сынақ арқылы әзірлеу

Сынақ арқылы әзірлеу – кодты және жасақтама әзірлеу тәсілдерін кезектесіп сынау үдерісі (Beck, 2002; Jeffries and Melnik, 2007). Мән бойымен, сіз кодты өсімше ретінде сынақпен бірге біртіндеп үдетесіз. Сіз жасаған кодыңыз сынақтан өтпегенше, сіз әзірлеудің келесі деңгейіне өтпейсіз. Әзірлеу иілгіш әдістердің жақтауларындағы сондай экстремалды бағдарламалау сынақ арқылы енгізілген. Алайда, ол да даму үдерісінде басқарылулардың жоспарына жұмсалуы мүмкін.



8.9-сурет. Тестілеумен әзірлеу

Сынақ арқылы әзірлеу негізгі үдерісі *8.9-суретте* көрсетілген. Үдерістің кезеңдері осындай:

1. Сіз талап еткен функционалдық өсімшесінің анықтамасынан бастаңыз. Әдетте, кодтың жүзеге асырылатын бірнеше жолдары бар.

2. Сіз функционалдық үшін сынақты жазасыз және автоматталған сынақта жүзеге асырыз. Бұл сынақтың атқарылғанын білдіруі мүмкін және ол сәтті қарастыпмауға да болады.
3. Сынақты, сонымен бірге болған іс өткізілген басқа сынақты сіз содан соң іске қосасыз. Бастапқыда, сіз функционалдықты жүзеге асырмай тұрып, жаңа сынақ құлайды. Бұл алдын ала сынақтың орнатылымын қоскандығын көрсетеді.
4. Сіз содан соң функционалдықты жүзеге асырасыз және сынақты қайтадан іске қосасыз. Бұл оны жақсарту үшін кодқа рефакторинг және жаңа код қосуы мүмкін.
5. Барлық сынақтар ойдағыдай өтілгеннен кейін, сіз функционалдықтың келесі бөлшегін жүзеге асыруға өтіңіз.

Сынаудың автоматтандырылған ортасында, Java (Massol және Husted, 2003) сынау бағдарламасы сүйейтін JUnit қоршаған орта сондай сынақ арқылы әзірлеу үшін маңызды мағынаға ие болады. Код өте кішкентай қадаммен жасалса, сіз әр сынақты іске қосу мүмкіндігіне ие боласыз. Қорыта келгенде, сынақ жеке бағдарламаға енгізілген және сынақ жүйенің шығуына алып келеді. Бірнеше секундтардың ішінде, ағысқа жеке сынақтардың жүздігін іске қосу мүмкіндігін қолдана аласыз.

Сынақ үшін, сізге не жазуына арналғанын түсінуіңіз керек, өйткені бұл түсінушілік оны қажетті кодқа жазуға жеңілдік жасайды. Әрине, егер толық емес білім немесе түсінушілік бар болса, онда әзірлеуге сынау көмектеспейді. Егер сіз әжептәуір білмесеңіз, сынақтар үшін, сізге жазуға қажетті кодтарды үдетпеңіз. Мысалы, егер сіздің есептеуіңіз бөлінуден тұрса, сіз нөлге сан бөлінуін тексеруіңіз керек. Ол үшін, егер сіз сынаққа жазуға ұмытсаңыз, онда кодты тексеруі үшін қосылған бағдарлама ешқашан болмайды.

Сонымен бірге мәселенің түсінушілігін жақсарту әзірлеудің, сынақтардың төмендегідей басқа артықшылықтары болады:

1. *Кодты толық сынау.* Қағидатта, сіз жазатын кодтың әр сегменті кемінде бір тиісті сынаудан өтуі керек. Нәтижесінде, сіз жазған кодтың әр бөлігі дұрыс орындалатынына сенімді боласыз. Бұл қателеріңізді кешіктірмей, әзірлеу үдерісінің барысында дұрыстап алуыңызға мүмкіндік береді.
2. *Регрессиялық сынау.* Бұл сынаулар бағдарламамен бірге әзірленеді. Бағдарламаға өзгеріс, жаңа қателер енгізбейтінін тексеру үшін, сіз регрессиялық сынауларды әрқашан өткізе аласыз.
3. *Ықшамдалған жөндеу.* Егер сынақ құласа, мәселе анық болу тиіс. Жақында жазылған және өзгертілген код тексерілуі керек. Сізге мәселені табу үшін дұрыстау құралды пайдалану керек. Әзірлеу сынақтары пайдалану баяндамаларында, әзірлеуде автоматталған жөндеуіштерді (Martin, 2007) сынау арқылы пайдалануға керектігін жориды.

4. *Құжаттама жүйесі.* Сынақтары суреттейтін құжаттаманың пішіні сияқты әрекет жасайды, онда негізгі код не жасауы керектігі жазылады. Сынақтардың оқуы кодтың түсінушілігі үшін оңай жасалуы мүмкін.

Әзірлеу сынақтарының ең маңызды артықшылықтары – ол регрессиялық сынаудың шығындарын арзандатылатыны. Регрессиялық сынау болған істің өзгерісінен кейін ойдағыдай орында жүйеге кіргізетін сынау кешендерін іске қоса тұрады. Регрессивтік сынақ бұл өзгерістер жүйеде жаңа қателер болған іс енгізетінін тексереді және код не жаңалық бар сияқты өзара әрекет болған кодпен күтіледі. Регрессиялық сынау өте қымбат және күш өте биік, жүйе жиі орамсыз, уақытпен сыналған. Мұндай жағдайларда, сіз қайтадан іске қосу үшін ең тиісті сынақтарды таңдап көруіңіз керек және маңызды сынақтарды жеңіл өткізуіңіз керек.

Алайда, регрессиялық сынаудың шығынын едәуір арзандататын сынау – автоматталған сынау. Сынақтар тез іске қосылады және арзан. Жүйеге өзгерістерді енгізгеннен кейін, бұл сынақтар қосылған сынаудың негізінде әзірлеуге арналған кезкелген функцияларды ойдағыдай орындауы керек. Бағдарламалаушы, сіз қосқан жаңа функционалдыққа немесе бұрынғы кодпен айқындалған мәселелерге алаң болмайды.

Әзірлеу арқылы жаңа бағдарламалық жасақтаманың функционалдығын әзірлеу сынақтарынан өткеннен кейін жаңа кодта жүзеге асырылған. Бұл жүйелер үшін сынақтарды бүтін жазу маңызды. Сынақ арқылы әзірлеу сонымен бірге көп тасқынды жүйелерде тиімсіз болуы мүмкін. Әртүрлі ағындар әртүрлі сынау жүгіріске уақытта әртүрлі сәттерге кезектесе алады және әртүрлі нәтижелер бере алады.

Егер сіз сынау арқылы әзірлеуді қолданып жатсаңыз, сізге жүйені тексеру үшін жүйелерді сынау үдерісін қолдануға тура келеді. Мысалы, жүйенің сынауы өнімділік, сенімділікті сынайды және жүйе дұрыс не бұрыс жасалғанын тексереді. Жағымсыз нәтижерелге әкелмеуге септігін тигізеді. Андреа (2007) сынау құралдарының сынау арқылы әзірлеу секілді жүйелерді тексеруде қалай кеңейетінін көрсетеді.

Сынау арқылы әзірлеу шағын және орта жобалар үшін тура келетіні дәлелденген. Әдеттегідей, бұл тәсілді қолданған бағдарламалаушылар онымен қанағаттандырылады (Jeffries және Melnik, 2007). Кейбір зерттеулерде, кодтың сапасын жақсартуға әкелетінін көрсеткен. Алайда, сынау арқылы әзірлеу кодтың жарамсыз сапасына алып келетіндігіне ешқандай дәлелдеме жоқ.

8.3. Релиз шығару

Өндеушілердің командасынан тыс пайдалану үшін арналған жүйенің нақты нобайын сынау үдерісі. Әдеттегідей, бұл тұтынушылар және пайдаланушылар үшін шығарылады. Күрделі жобада, алайда, ұқсас жүйелерді өндейтін командалар үшін

де болуы мүмкін. Кейбір бағдарлама әзірлейтін мекемелер релиз шығарылымын кейін сатуға көздейді.

Релиз шығарылымын әзірлеудің үдерісінде және жүйелік сынақтың аралығында екі маңызды айырмашылықтар бар:

1. Релиз шығарылымына жалпы бағдарлама әзірлеп жатқан топқа қатысы жоқ топ жауапты.
2. Бағдарлама әзірлеп жатқан топ жүйелік сынауға мән беруге тиіс. Релиздің мақсаты – жүйенің талаптарға сай болғанын және (тексеру сынақтар) сыртқа қолдануға жарамды екенін тексеру.

Релиздің түпкі мақсаты – тұтынушыны жүйенің пайдалануға жеткілікті болғанын сендіру. Егер солай болса, онда оның өнімінің сапасында немесе берілген тапсырыстарында күмән болмайды.

Шығарылым сынағы әдеттегі жағдайда, қара жәшік әдісі бойынша жүргізілетін сынақ болып табылады. Мұнда сынақтар жүйе спецификациясынан алынады. Бұл жүйе тек сол жүйенің кірістері мен қатыстық шығыстарын зерттеу арқылы ғана анықтау мүмкіндігіне ие қара жәшік ретінде қабылданады. Бұл әрекеттің басқа да атауы ‘функционалды сынақ’ болып табылады, оның осылай аталуы сынақ жүргізушінің бағдарламалық жасақтаманы іске қосылу жағдайларымен емес, тек жүйенің функцияларымен ғана жұмыс жүргізуімен түсіндіріледі.

8.3.1. Талаптарға негізделген сынақ

Талаптардың сынақтан өтуге жарамды болуы талаптарды өңдеу тәжірибесінің негізгі принципі болып табылады; бұл дегеніңіз, талаптар сол аталмыш талаптарға арналған сынақтың жобалануына мүмкіндік беретіндей етіп жазылуы тиіс. Сынақ жүргізуші талаптардың қанағаттандыру деңгейін тексереді. Сондықтан, талаптарға негізделген сынақ шығарып алынатын әрбір талап жиынын қарастырады, ал талапқа арналған сынақтар сынақтарды жобалауға қатысты жүйелі қатынас болып табылады. Талаптарға негізделген сынақ ақаулықтар сынағына қарағанда жарамдырақ келеді – сіз мұнда жүйенің қойылған талаптарға сай дұрыс жұмысын көрсетуге талпынасыз.

Мысалы, MHC-PMS (1-тарауда таныстырып өткен) үшін берілген дәрілік аллергиялар жағдайын тексеруге қатысты сәйкесінше талаптарды қарастырып өтейік:

> *Егер науқас кез келген белгілі бір дәрі-дәрмекке аллергиясы бар екені белгілі болған жағдайда, жүйе пайдаланушысы арқылы сол дәрі-дәрмек нұсқаулығы ескерту хабарламасында шығарылып тұруы тиіс.*
>
> *Егер нұсқаулық беруші маман аллерлияға қатысты хабарламаларды елемеуді қалайтын болса, неліктен елегісі келмегендігінің себебін түсіндіруге міндетті.*

Талаптардың қанағаттандырылуы туралы тексеріс жұмыстарын жүргізу үшін, төмендегі қатысты бірнеше сынақтарды жасақтап шығару қажет болуы мүмкін:

1. Науқастың жазбасын еш аллергиясы белгісіз деген мәнге орнатып қойыңыз. Пайда болуына белгілі болған аллергиялар үшін қажетті дәрі-дәрмекке нұсқаулық жазыңыз. Жүйенің ешқандай ескерту хабарламасын шығарып тұрмағандығын тексеріңіз.
2. Науқастың жазбасын өзіңізге белгілі болған аллергия түріне орнатып қойыңыз. Белгілі болған аллергия түрі үшін қажетті дәрі-дәрмекке нұсқаулық жазыңыз және жүйенің ешқандай ескерту хабарламасын шығарып тұрмағандығын тексеріңіз.
3. Науқастың жазбасын өзіңізге жазылған жазбалар арқылы белгілі болған екі немесе одан көп дәрілер түріне орнатып қойыңыз. Осы дәрі-дәрмектің екеуіне де жеке-жеке нұсқаулық жазыңыз және шығарылып тұрған әрбір дәріге берілген ескерту хабарламаның дұрыстығына көз жеткізіңіз.
4. Науқастың аллергиясы бар екі дәрі-дәрмек түріне нұсқаулық жазыңыз. Шығарылып тұрған ескерту хабарламаларының дұрыстығын тексеріңіз.
5. Ескерту хабарламасында көрсетіліп тұрған дәрі-дәрмекке нұсқаулық жазыңыз және сол ескерту хабарламасын қайтарып жіберіңіз. Жүйенің пайдаланушыдан оның ескерту хабарламасын қайтарып жіберу себебін түсіндіретін мәліметті талап етіп отырғандығын тексеріңіз.

Сіз осы үдерістерден көріп отырғаныңыздай, талаптарды сынақтан өткізу жеке сынақты жазумен ғана шектеліп қалмайтындығын білдіреді. Сіз қалыпты жағдайда, талаптардың қамтылуына ие екендігіңізге көз жеткізу мақсатында бірнеше сынақ түрлерін жазуыңызға болады. Сіз сонымен қатар, сынақтарды сынақтан өткізіліп отырған арнаулы талаптармен байланыстырушы талаптарға негізделген сынағыңыздың бағытын қадағалау жазбаларына да қол жеткізуіңізге болады.

8.3.2 Сценарий сынағы

Сценарий сынағы сіздің пайдалану сценариі мен жүйеге арналған сынақ деректерінің жиынын жасақтау үшін осыларды пайдаланудің типтік сценариін бөлуіңіздегі шығарылым сынағының қатынасы болып табылады. Сценарий дегеніміз жүйенің пайдалану жолдары сипатталатын әңгіме. Сценарий желісі шынайы болып келеді және ондағы нақты жүйенің пайдаланушылары олармен өзара қатынасқа түсе алады. Егер сіз сценарийді өңдеу үдерісі талаптарының бір бөлігі ретінде пайдаланатын болсаңыз, (4-тарауда сипатталған), онда сіз осы сынақ сценариін қайта пайдалана аласыз.

Сценарий сынағының қысқа парағында, Канер (2003 ж.) ұсынысына сәйкес, сценарий сынағы шағын әңгіме ретінде болуы тиіс, сондай-ақ ол шындыққа жанасатын сенімді кешен болуы қажет. Бұлай болу бірқатар мүдделі тұлғаларды ын-

таландыруы тиіс; бұл дегеніңіз, олар сценарийге тікелей қатысады және жүйенің сынақтан өтетіні маңызды екендігіне сенетін болады. Канер сонымен қатар, бұл сынақты бағалау әрекеті де оңай жүреді деп тұжырымдайды. Егер жүйеге қатысты проблемалар туындайтын болса, онда шығарылым сынағының мамандар ұжымы орын алған мәселелерді табатын болады. Пайда болуы ықтимал сценарийдің мысалы ретінде MHC-PMS жүйесін алуға болады, *8.10-суретте* жергілікті шолуда пайдалануға болатын жүйенің бір жолы сипатталған.

Бұл MHC-PMS жүйесінің бірнеше мүмкіндіктерін сынақтан өткізеді:

1. Жүйеге кіру арқылы жүргізілетін түпнұсқалық растама.
2. Арнаулы науқастардың жазбаларын қолкомпьютерге жүктеу және кері жүктеу.
3. Жергілікті шолу кестесі.
4. Ұшқыр құрылғыдағы науқастың жазбаларын шифрлеу және шифрынан шешу.
6. Теріс әсер туралы ақпарат алынатын дәрі-дәрмектер дерекқорымен байланыстар.
7. Қоңырау сүйемеліне арналған жүйе.

Егер сіз шығарылым сынағын жүргізуші болсаңыз, онда осы сценарий бойынша Катяның рөлін ойнай отырып, және жүйенің түрлі кірістерге жауап беру мәнерін бақылай отырып бағдарламалық жасақтаманы осы сценарий арқылы жүргізуіңізге болады. ‘Катя’, яғни сіз жазбалардың шифрынан шешу үшін қате кілт сөзін енгізу сияқты әдейі алдын ала ойластырылған қате жіберуіңізге болады. Пайда болған кез келген проблемаларға және оған қоса өнімділікке қатысты проблемаларға ерекше назар аударғаныңыз жөн. Егер жүйе тым баяу жұмыс жасай бастаса, бұл қолданыстағы пайдаланудың басқа жолына өзгереді. Мысалы, егер жазбаны шифрлау тым ұзын болып кетсе, онда уақыты қысқа болған басқа пайдаланушы келесі сатыға өтіп кетеді. Егер олар қолкомпьютерлерін жоғалтып алған болса, онда рұқсаты жоқ тұлғада науқастың жазбаларына шолу жасау мүмкіндігіне ие болады.

Сіздің сценарийге негізделген қатынас жасауыңыз барысында, сол бір сценарий бойынша бірнеше талаптарды қалыпты жағдайда сынақтан өткізе беруіңізге болады. Осылайша, жеке талаптарды тексерумен қатар, еш мәселе тудырмастан, бірнеше талаптардың тіркесімдерін де тексере беру мүмкіндігіңіз бар.

8.3.3. Өнімділік сынағы

Жүйе толығымен біріктіріліп біткеннен кейін, өнімділік пен сенімділік сияқты тәуелсіз сипаттарға сынақ жүргізу мүмкіндігі туады. Өнімділік сынақтары жүйенің өзіне тән жүктемелерді өңдеу мүмкіндігін тексеру мақсатында жасақталған. Бұл әдетте, жүйенің сынақ барысындағы жүк көлемін оның жүктемені қабылдауға

мүмкіндігі қалмағанша ұлғайта отырып жүргізетін бірқатар сынақтар тізбегін қамтиды.

Сынақтың басқа да түрлері сияқты, өнімділік сынағы да жүйенің талаптарына сай келетіндігін көрсетеді және жүйедегі проблемалар мен ақаулықтарды табады. Өнімділік талаптарының жеткілікті деңгейге жеткендігін тексеру үшін функционалды қима құруыңыз тиіс. Функуционалды қима (*15-тарауға* қараңыз) жүйе арқылы жүргізілетін жұмыстың нақты параметрін көрсететін сынақтар жиынтығы болып табылады. Осыдан кейін, егер жүйедегі бірізді амалдардың 90%-ы А түрінде; 5%-ы В түрінде; және қалғандағы C, D, және E түрлерінде, функционалды қиманы сынақтардың басым бөлігі А түрінде болатындай етіп жобалау қажет. Болмаған жағдайда, сіз жүйенің функционалдық өнімділігіне жүргізілген сынақтың тура нәтижесіне қол жеткізе алмайсыз.

Әлбетте, бұл қатынас ақаулар сынағына арналған үздік қатынас болып табылмайды. Тәжірибе нәтижелері көрсеткендей, ақаулықтарды табудың тиімді жолы жүйенің шектемелері айналасында сынақтар жобалап шығару болып табылады. Өнімділік сынағында, бұл дегеніңіз бағдарламалық жасақтамасының жоба шектемелерінен тыс талаптарды жасау арқылы жүктеме жүргізу деген сөз. Мысалы, сіз секундына 300-ге дейін бірізді амалдарды өңдеу жүйесін сынақтан өткізіп жатырсыз делік. Сіз осы жүйе сынағының басын секундына 300-ден төмен бірізді амалдарды өңдеуден бастадыңыз. Осыдан кейін сіз жүйенің максималды жобаланған жүктеме шегінен асып кеткенше және жүйе істен шыққанға дейін үдемелі түрде жүйедегі жүктеме мөлшерін секундына 300-ден асатын бірізді амалдарды өңдеу жүйесін жүргіздіңіз. Сынақтың бұл түрі екі функцияға ие, олар:

1. Бұл сынақ түрі жүйенің істен шыққандағы әрекетін сынақтан өткізеді. Жүйедегі жағдайдар жүйеде орын алған жүктеменің болжалды жүктеменің максималды мәнінен асқан оқиғалардың күтпеген жиынтығы арқылы пайда болуы мүмкін. Осындай жағдайларда, жүйенің істен шығуы жүйе мазмұнындағы деректердің бүлінуіне немесе пайдаланушы қызметінің күтпеген жоғалту жағдайларына себепші болып кетпеуі маңызды болып табылады. Жүктеме сынағы жүйенің жүктеме салдарынан қысылу жағдайына қарағанда, жүйедегі параметрлердің сатылы түрде нашарлауына себепші болуын тексереді.
2. Бұл жүйеге жүк түсіреді және соның салдарынан әдеттегі жағдайда анықтау мүмкін болмайтын ақаулықтар жарыққа шыға бастайды. Дегенмен, бұл ақаулықтары қалыпты қолданыс барысындағы жүйенің қателеріне себепші болатын сынақ әрекетін жүргізу қажетсіз деген талқылар бой көтергенмен, жүктеме сынағы қайталанатын қалыпты жағдайлардың ерекше тіркесімдері орын ала беруі мүмкін.

Жүктеме сынағы бөлшекті түрде процессорлардың желісіне негізделіп үлестірілген жүйелерге қатысы бар болып табылады. Бұл жүйелер жиі жағдайда жүктері ауыр жүктелген кезде қатты нашарлаған мәнге ие болады. Желі түрлі үдерістерді алмастыратын координация дерегімен толтырылады. Өңдеу әрекеттері

бара-бара баяу жүреді, себебі олар басқа үдерістерден талап етілген деректерді күтеді. Жүктеме сынағы нашарлау үдерісі орын ала бастаған сәтін анықтауға көмектеседі. Осылайша сіз осы нүктенің артындағы бірізді амалдарды қайтару үшін тексерістерді қосу мүмкіндігіне ие боласыз.

Кейт медбике дерлік. Оның жауапкершіліктерінің біреуі адамдарды үйлерінде кіріп шығып халдарын білу болсын.

Кіріп шығатын күндерінде ол MHC-PMS-ке кіріп аралайтын мекен-жайларды қағазға басып шығарып алады. Кіріп шыққан адамдардың сұранысстары өзінің жеке компьютеріне жазылып тұрады. Ол компьютеріндегі ақпаратты өзінің кілті арқылы шифрлайды.

Кіріп шығатын науқастардың бірі Джим. Ол депрессияға қарсы дәрі-дәрмек қабылдап жүр. Кейт ол туралы ақпараттарды сақтайды және әрқашан оны сұраған кезінде өзінің құпия кілтін енгізуге сұралады. Ол дәрі-дәрмектің Джимге келтіретін әсерін қадағалап отырады. Кейт Джимнің түнде нашар ұйықтайтыны жайлы хабардар, сондықтан ол Джимнің емханаға баруын талап етіп, осы ақпарат дереу компьютерге жазылады.

Кейін Кейт жұмысына оралғанда оның кіріп-шыққан науқастары жөніндегі ақпараттарды оқып шығып, уәде берген науқастарына дәрігермен келісіп, емханаға келетін уақытын телефон арқылы айтады.

8.10-сурет. MHC-PMS-ті пайдалану мысалы

8.4. Пайдаланушылық сынақ

Сынау пайдаланушысы немесе клиенті сынау үдерісінінің кезеңі болып табылады немесе клиенттік салымды және кеңестерді жүйенің сынауы қамтамасыз етеді. Егер бұл оларға жақса, онда оларға қажетті сыртқы жабдықтаушыда тағайындалған жүйенің ресми сынауы қосылуы мүмкін немесе пайдаланушыларға жаңа бағдарламалық өніммен эксперимент жүргізген үйреншікті емес үдеріс болуы мүмкін.

Жүйенің өңдеушісі үшін бұл іс жүзінде мүмкін емес. Бұл барлық жүйенің пайдалануына әсер етеді. Бірақ өңдеушілер оларды сынау ортасында қоса алмайды.

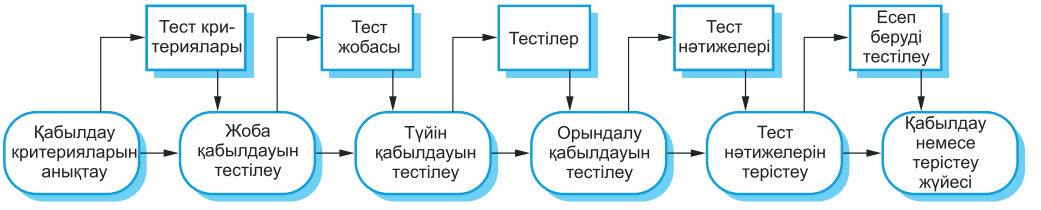
Альфа-сынауда, пайдаланушы және өңдеуші бірге жұмыс істеуі үшін, осы шақта жүйенің қалай жасалып тұрғанын тексереді. Бұл пайдаланушы мәселелері және дамудың сынақтық командасы үшін анық болып көрінбеген сұрақтар туралы біле алғанын білдіреді. Өңдеуші ғана талаптардан шынымен жұмыс істей алады, бірақ олар бағдарламалық жасақтаманың жаттығу пайдалануларына әсер еткен басқа факторларды жиі бейнелеп көрсетпейді. Демек, пайдаланушы әрлендірумен реалистік сынақтардан астам көмектескен тәжірибе туралы ақпаратпен қамтамасыз етуі мүмкін.

Альфа-сынау қорапты жүйе түрінде сататын бағдарламалық өнімдерді әзірлеудің жанында жиі пайдаланады. Бұл өнімдерді пайдаланушы альфа- сынау үдеріске араласуға даяр бола алады, өйткені бұл олар пайдалана алған

жүйенің жаңа функциялары туралы ерте мәліметтерін береді. Бұл да тәуекел, бағдарламалық жасақтамасына ойда болмаған өзгерістері, бизнес қиратушы әсері болғасын арзандатылады. Алайда, альфа-сынауда жұмсалуы мүмкін, осы шаққа арнаулы бағдарламалық жасақтама жасалады. Және даму үдерісінде Agile әдістері, сондай-ақ XP, пайдаланушының қатысуын қорғаушы, пайдаланушы жүйе үшін сынақтардың әзірлеуінде маңызды рөл атқаруы керек.

Бета-сынауда, кейде бітпеген бағдарламалық жүйені ерте шығару үшін оны клиенттерге және баға үшін пайдаланушыларға қолжетімді етеді. Бета- сынаушылар жүйеде лидерлері болып көрінген клиенттер тобы болып сайлана алады. Одан басқа, бағдарламалық жасақтама пайдаланылуы үшін жұрт алдында қол жететін барлық мүмкіндіктері қызықтырылған. Бета-сынау әртүрлі орталарда пайдаланатын программалық өнімдер үшін пайдаланады.

Қабылдау сынақтары тапсырыс бойынша істелген әзірлеу жүйесінің ажырамас бөлігі болып көрінеді. Бұл сынаудың босауынан кейін болады. Ол клиент үстірт жүйені сынайды, шешімі үшін, ол жүйенің өңдеушісімен қабылдауы керек. Қабылдауды өтем жүйе үшін жасалынуы керек болатынын білдіреді.



8.11-сурет. Қабылдауды тестілеу үдерісі

8.11-суретте көрсетілгендей қабылдау сынақтарының үдерісінде алты кезең бар. Олар:

1. *Қабылдаудың белгісін анықтау.* Бұл кезең тиісті жүйе үшін шарт жасауға дейін үдерістің басында тұрып қол қойылады. Қабылдаудың белгісі шарт жүйенің бір бөлігі болуы керек және тапсырысшының аралығында және өңдеушімен келісілуі қажет. Тәжірибеде үдерістің ерте кезеңдерінде оны анықтауға қиын екендігі көрсетілген. Толық талаптар қол жететін бола алмайды және талаптарды әзірлеудің үдерісінде елеулі өзгерістер бола алады.
2. *Қабылдаудың сынау жоспары.* Бұл – ресурстар туралы шешім қабылдау, уақыт және қабылдау-тапсыру сынағындағы бюджеті және тестілеудің кестесін анықтау.
3. *Қабылдау-тапсыру сынақтарын алып шығу.* Орнатылған белгіні қабылдау, сынақтардан кейін жасалатын жүйе үшін қолайлы болып көрінгенін тексеру. Қабылдау сынақтары функционалдық тексеру үшін ыңғайлануы керек, мысалы, өнімділік – функционалдық емес сипаттамалар. Олар жүйелік талаптармен толық қамтуды қамтамасыз етуге тиіс. Аргумент үшін мүмкіндіктер жиі болады.

4. *Қабылдау-тапсыру сынақтарын орындау.* Келісілген қабылдау сынақтары жүйеде орындалады. Жүйеде пайдаланған нақты ортада болуы керек. Қорыта келгенде, пайдаланушыларды сынаудың ортасында, бұл сынақтан шығару тура келеді. Бұл үдерістің автоматтандыруы өте қиын, бұл сынақ қабылдаудың және жүйенің бір бөлігі пайдаланушының арасындағы өзара іс-әрекеттің сынауын қоса алады. Кейбір оқу пайдаланушылары талап етілуі мүмкін.
5. *Сынау нәтижелерінің келісуі.* Бұл – белгілі қабылдау сынақтары өткен және ешқандай да мәселелер болмаған жағдай.
6. *Жүйемен қабылдамау/қабылдау.* Бұл кезең өндеушілердің арасындағы кездесуден тұрады және тапсырысшы туралы шешім қабылданады немесе жүйе қабылданудың керегі жоқ. Егер жүйені пайдалану әжептәуір жақсы болып көрінбесе, онда одан әрі даму айқындалған мәселелер түзетуде талап етеді. Қабылдау сынауын кезеңнің аяқтауынан кейін қайталайды.

ХР-дегідей, қабылдау-тапсыру сынағының мағынасы өзгерек. Қағидатта, ол пайдаланушы шешуі керек болатын пікірді анықтайды, жүйе қолайлы болып көрінеді. Яғни ХР, пайдаланушы өндеушілер командасының құрамына кіреді (немесе ол тексеруші альфа болып көрінеді) және қолданбалы әңгімелерді пікірден жүйеге талапты қамтамасыз етеді. Ол немесе ол да болуға бел байласатын сынақтардың анықтамасына жауап береді немесе бағдарламалық жасақтама жасалмаған әңгіменің пайдаланушысы сүйейді. Сынақ автоматталған және даму қабылдаудың әңгімесінің сынағына дейін жүрмейді. Қорыта келгенде, қабылдау сынақтары ешқандай да жеке қызмет болып көрінбейді.

Қабылдау-тапсыру сынақтары айқын келісім сұрағына болып көрінгенін ойлауға болады. Егер жүйе оның қабылдау сынақтарынан өтпесе, онда ол қабылдануы және төлеу жасалынуы керек. Клиенттер бағдарламалық қамсыздандыруды пайдалануын қалайды, олар өзінің жылдам ұңғылауының артықшылығының біледі. Олар үйретілген қызметшіні, жаңа жабдықты сатып алды және өз үдерістерін өзгертті. Олар бағдарламалық жасақтама қиындықтарын тәуелсіз қабылдауға даяр бола алады, мәселе шығын жұмысқа айналады, өйткені шығын бағдарламалық жасақтама артық пайдаланбайды. Қорыта келгенде, келіссөздерді анықтау – жүйенің шартты қабылдауының мүмкін болуы. Жүйе көкейкесті мәселені жөндеуге келісім береді және клиент жаңа болжамды қалай болса да тезірек жеткізеді.

НЕГІЗГІ ТҰЖЫРЫМДАР

- Сынау бағдарламадағы қателіктердің ғана бар болуын көрсетуі мүмкін. Оның ақаулықта қалмағанын көрсетуі мүмкін.
- Сынауды дамыту – бағдарламалық жасақтама өңдеушілері тобының міндеті болып көрінеді.

- Сынаудың дамуы сізге жеке объектілерді сынайтын модулдік сынауды қосады және сіз объектілердің байланған топтарын сынайсыз.
- Кейде, мүмкіндігінше, сіз автоматталған сынақтар жазуыңыз керек. Кіріктірме әрбір рет іске қосылған бағдарламаларға сынақ, жүйеге өзгерісті киргізуі мүмкін.
- Кодтың шағын өзгерісі және сынақ әзірше ойдағыдай орындайтын кодтың рефакторингін жасайды.
- Сынаудың сценарийі пайдалы болып көрінеді, себебі ол жүйе жаттығуының пайдалануын жаңғыртады. Ол пайдаланудың типті сценарийін ойлап табудан тұрады және сынақтардың алуы үшін пайдаланады.

ҚОСЫМША ӘДЕБИЕТТЕР

'How to design practical test cases'. A how-to article on test case design by an author from a Japanese company that has a very good reputation for delivering software with very few faults. (T. Yamaura, *IEEE Software*, **15**(6), November 1998.) http://dx.doi.org/10.1109/52.730835.

How to Break Software: A Practical Guide to Testing. This is a practical, rather than theoretical, book on software testing in which the author presents a set of experience-based guidelines on designing tests that are likely to be effective in discovering system faults. (J. A. Whittaker, Addison-Wesley, 2002.)

'Software Testing and Verification'. This special issue of the *IBM Systems Journal* includes a number of papers on testing, including a good general overview, papers on test metrics, and test automation. (*IBM Systems Journal*, **41**(1), January 2002.)

'Test-driven development'. This special issue on test-driven development includes a good general overview of TDD as well as experience papers on how TDD has been used for different types of software. (*IEEE Software*, **24** (3) May/June 2007.)

ЖАТТЫҒУЛАР

8.1. Бағдарламаның тұтынушыларға берілмей тұрып, неге ақуалдардан таза болу керек екендігін түсіндіріңіз.

8.2. Сынау қателердің бар болғанын тауып, ал жоқ болғанын таба алмайтынын түсіндіріңіз.

8.3. Кейбір адамдар өңдеуші өз кодты сынауға қатысу керек еместігіне сенеді, яғни сынау тыс команданың міндеті болуы керек. Осыған оң және теріс дәлел беріңіз.

8.4. Тармақ жақтауларындағы, бір бос белгімен бос орындардың тізбегін алмастыратын объект "тармақ" "catWhiteSpace" аталатын әдісті тексеруге сізден сұралған. Мысалы, ол үшін бөлім сынауды анықтаңыз және 'catWhiteSpace'әдісі үшін сынақтардың теруін алыңыз.

8.5. Регрессивтік сынау деген не? Автоматталған сынақтардың пайдалануын және JUnit регрессиялық сынауды ықшамдауын түсіндіріңіз.

8.6. MHC-PMS салынған. Сіз мұндай жүйені сынаудың аралығында айырмашылық қандай және тіл пайдаланумен объектіге бағдарланып жасалған бағдарламалық жасақтаманың сынауы неде?

8.7. Шөл даладағы ауа райы станциясының бағдарламасын сынауға сценарий жазыңыз.

8.8. «Стрестік сынау» деген терминді қалай түсінесіз?

Релиздің бағдарламалық жасақтаманы сынаудағы пайдаларын көрсетіңіз.

ПАЙДАЛАНЫЛҒАН ӘДЕБИЕТТЕР

Andrea, J. (2007). 'Envisioning the Next Generation of Functional Testing Tools'. *IEEE Software*, 24 (3), 58–65.

Beck, K. (2002). *Test Driven Development: By Example*. Boston: Addison-Wesley.

Bezier, B. (1990). *Software Testing Techniques, 2nd edition*. New York: Van Nostrand Rheinhold.

Boehm, B. W. (1979). 'Software engineering; R & D Trends and defense needs.' In *Research Directions in Software Technology*. Wegner, P. (ed.). Cambridge, Mass.: MIT Press. 1–9.

Cusamano, M. and Selby, R. W. (1998). *Microsoft Secrets*. New York: Simon and Shuster.

Dijkstra, E. W., Dahl, O. J. and Hoare, C. A. R. (1972). *Structured Programming*. London: Academic Press.

Fagan, M. E. (1986). 'Advances in Software Inspections'. *IEEE Trans. on Software Eng.*, **SE-12** (7), 744–51.

Jeffries, R. and Melnik, G. (2007). 'TDD: The Art of Fearless Programming'. *IEEE Software*, **24**, 24–30.

Kaner, C. (2003). 'The power of 'What If . . .' and nine ways to fuel your imagination: Cem Kaner on scenario testing'. *Software Testing and Quality Engineering*, **5** (5), 16–22.

Lutz, R. R. (1993). 'Analyzing Software Requirements Errors in Safety-Critical Embedded Systems'. RE'93, San Diego, Calif.: IEEE.

Martin, R. C. (2007). 'Professionalism and Test-Driven Development'. *IEEE Software*, **24** (3), 32–6.

Massol, V. and Husted, T. (2003). *JUnit in Action*. Greenwich, Conn.: Manning Publications Co.

Prowell, S. J., Trammell, C. J., Linger, R. C. and Poore, J. H. (1999). *Cleanroom Software Engineering: Technology and Process*. Reading, Mass.: Addison-Wesley.

Whittaker, J. W. (2002). *How to Break Software: A Practical Guide to Testing*. Boston: Addison-Wesley.

9.

Бағдарламалық жасақтама эволюциясы

Мақсаттары

Осы тараудың мақсаттарының бірі – бағдарламалық қамтамасыздандырудың эволюциясын түсіндіру, бағдарламалық жасақтаманың маңызды бөлігін және бағдарламалық қамтамасыздандыру үдерісін суреттеу. Осы тараудан кейін, сіз:

- Бағдарламалық қамтамасыздандырудың жүйесі және бағдарламалық қамтамасыз ету пайдасыз болған жағдайда, оны өзгерту шарасыз екенін түсінесіз.
- Бағдарламалық қамтамасыздандырудың үдерісін және осы үдерістерге әсер етуін түсінесіз.
- Бағдарламалық қамтамасыз етудің әртүрлі түрлерін қарастырасыз және техникалық қызмет көрсетуге әсер ететін факторлар жайлы түсініктеме аласыз.
- Мұра етілген жүйелермен жұмыс жасағанда, оларды орындатпау, қуаттау, қайтадан қайта өңдеу немесе алмастыру жайында мағлұмат аласыз.

Мазмұны

Бағдарламамен қамтамасыздандырудың дамуы тоқтамайды, бірақ жүйенің жинағы қызмет ету мерзімінің ағымына созылады. Қолданушылардың күтулеріндегі өзгертулер, қазіргі бизнесте бағдарламалық қамтамасыз етуге жаңа талаптарын туындатып отыр. Бағдарламалық қамтамасыз етуді жақсарту мақсатында, аппаратты және програмалық тұғырға өзгерту үшін, оны дағдыландырып,сонымен қатар өнімділігін немесе функционалдық сипаттамаларын жақсарту керек.

Ұйымдар бағдарламалық қамтамасыз етуге қаржылай көмек көрсеткендіктен, бағдарламалық қамтамасыздандыру эволюциясы ұйымдар үшін маңызды болып саналады. Олардың бұл жүйесі маңызды активті бизнес және бұл активтердің құнын қадағалау үшін жүйенің өзгеруін инвестициялау керек. Демек, ірі компаниялардың көпшілігі жаңа жүйеге қаржы бөлуге қарағанда, дайын жүйені қолдап отыр. Өнеркәсіптің бейресми сұрақтары негізінде, Эрлих (2000) көрсеткіші бойынша, бағдарламалық қамтамасыз етуге кеткен ұйымдардың 85-90% шығыны қазіргі кезде шығындардың эволюциясы болып табылғанын көрсетеді. Басқа да зерттеулердің көрсеткіші бойынша бағдарламалық қамтамасыз етуге кеткен шығындардың шамамен, 3\2 бөлігі эволюциялық шығын екенін көрсетіп отыр. Әрине, бағдарламалық қамтамасыз етудің өзгеруіне кеткен шығындар барлық компаниялардағы ИТ-бюджеттің үлкен бір бөлігі болып саналады.

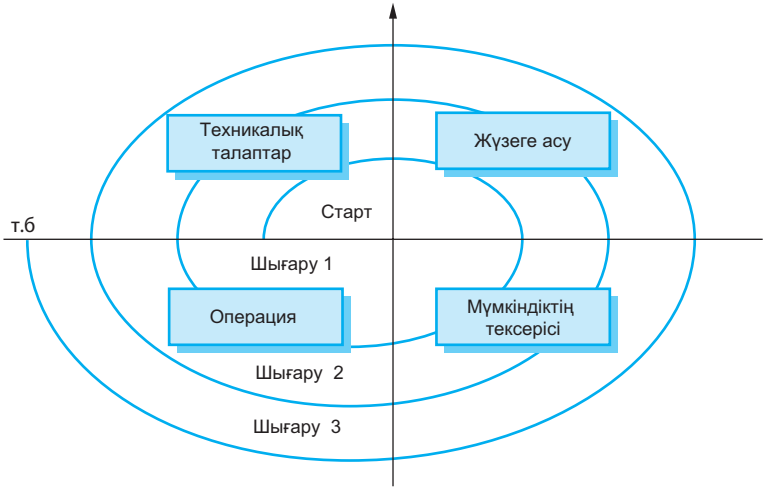
Бағдарламалық қамтамасыз етудің эволюциясы бизнес талаптарының өзгеруіне байланысты немесе бағдарламалық қамтамасыз етудің ақаулары негізінде туындауы мүмкін. Хопкинс және Дженкис (2008) жылы "қараусыз қалған бағдарламаны қамтамасыз ету" терминін енгізді, осы жағдайды сипаттау мақсатынды програмалық жүйе тек қана сол ортада, басқа да көптеген бағдарламалық жүйеге бағынышты жерде жасалуы және басқарылуы тиіс.

Қорыта келе, жүйенің эволюциясы өте сирек оңаша қарала алады. Қоршаған ортаның өзгеруі жүйенің өзгеруіне әкеліп соқтырады, кейіннен қоршаған ортаның өзгеруіне сылтау болып табылуы мүмкін. Әрине, жүйенің "бай жүйеде"дамуы қиындықтар мен шығын эволюциясын арттырып отыр. Сонымен бірге жүйеге ұсынылатын өзгерістердің әсерін түсіну және талдау, сонымен қатар бұл қалай операциялық ортада басқа жүйелерге ықпал тигізе алатынын бағалауыңыз қажет.

Пайдалы програмалық жүйелер өте ұзақ қызмет көрсете алады. Мысалы, әскери жүйелер немесе инфрақұрылымдар, әуе қимылын басқару жүйесі 30 жылдай немесе одан да көп қызмет көрсетеді. Бизнес жүйелері көбінесе, 10 жылдан астам қызмет көрсетеді. Бағдарламалық қамтамасыз ету өте көп қаржыны талап етеді, сондықтан бағдарламалық қамтамасыз ету жүйесін көп жылдар пайдалануы қажет. Қойылған жүйелердің талабы, қоршаған орта бизнесін және оның өзгеруін өзгерткені анық. Қорыта келе, жүйенің жаңа болжамы, сонымен қатар өзгертулер мен жаңартулар үздіксіз негізде құрылады.

Демек, сіз бағдарламалық жасақтаманың шиыршық тәріздес түрін, талаптарын, жобалауын, жүзеге асырылуын және тестілеуін, жүйенің қызмет ету мерзімінің ағымы туралы ойлануыңыз керек (*9.1-сурет*). Бір жүйенің релизін жасаудан бастаймыз. Жеткізуден кейін өзгеру мен дамудың дереу 2 шығарылымы ұсынылады. Шын мәнінде, эволюцияға деген қажеттілік анық көрініс таба алады

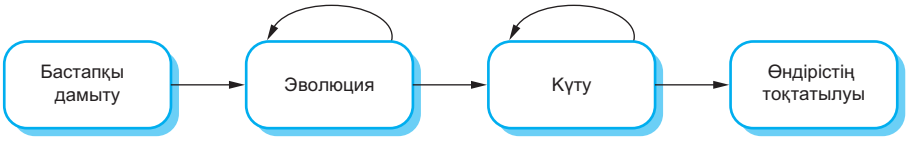
және бағдарламалық қамтамасыз ету келесі болжам жасалғанға дейін даму үстінде болады.



9.1-сурет. Әзірлеудің және эволюцияның спиральді моделі

Бұл жағдайда, үдерістің шиыршығында үзіліс болады. Талап пен жоба құжаттамасы бір компаниядан екінші компанияға беріле алмайды. Компания топтастырады немесе қайта ұйымдастырады және басқа бағдарламамен қамтамасыз етуін басқа компаниялардан мұра етеді, содан соң бұл өзгертілуі тиіс. Әзірлеудің эволюцияға орын ауыстыруы жапсарсыз емес, бағдарламалық қамтамасыз етудің өзгеруінен кейін көбінесе, "Бағдарламамен қамтамасыз етуді бақылау" деп атайды.

Раджлич және Беннет (2000) жылы бағдарламалық қамтамасыз етудің эволюциясының өмірлік оралымына баламалы көзқарасты ұсынды. Бұл үлгіде, эволюция мен қызмет ету аралығы туралы пікірлерін айтты (*9.2-сурет*). Эволюция архитектуралық және бағдарламалық қамтамасыз етудің функционалындағы түбегейлі өзгертулер істеліне алатын фаза болып табылады. Қызмет көрсету кезінде істелінген өзгертулер, маңызды өзгерістер болып саналады.



9.2-сурет. Өзгерістерді анықтау және эволюция үдерісі

Эволюция үдерісінде, бағдарламалық қамтамасыз етуді ойдағыдай пайдаланады және талаптардың ұсынылатын өзгерістерінде тұрақты ағыс бар. Алайда бағдарламалық қамтамасыз ету өзгертілген, оның құрылымының бұзушылық тенденциясы және өзгеруі бара-бара қымбаттырақ болып барады. Бұл жиі бола-

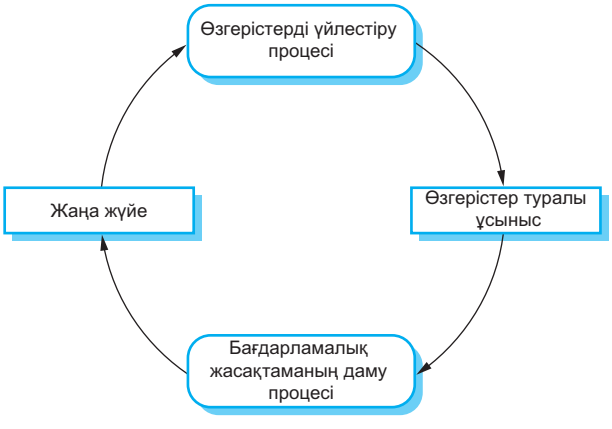
тын жағдай, бірнеше жыл қолданылғаннан кейін басқа экологиялық өзгерулер, жабдық және басқару жүйелері жиі өзгертуді талап етеді. Белгілі бір өмірлік кезеңде, бағдарламалық қамтамасыз ету белгілі бір нүктеге жеткенде, түбегейлі өзгерулерді, жаңа талаптарды енгізудің пайдасы аз болып қалады.

Бұл кезеңде бағдарламалық қамтамасыз ету эволюциядан қызмет етуге өтеді. Қызмет ету фазасында бағдарламалық қамтамасыз ету әлі де пайдалы және пайдаланады, бірақ тек қана шағын өзгертулер ғана жасалған. Бұл кезеңде, әдеттегідей компания бағдарламалық қамтамасыз етуді ескеру керек. Қорытынды кезеңдерде,бағдарламалық қамтамасыз етуді пайдалануға болады, бірақ одан әрі қарай өзгерулер жүзеге аспайды. Қолданушы кез келген мәселеден айналып өтуі тиіс.

9.1. Эволюцияның үдерістері

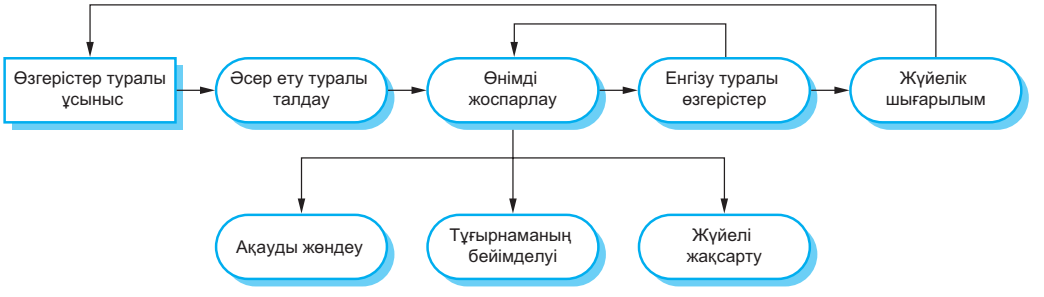
Бағдарламалық қамтамасыз ету эволюциясының үдерісі бағдарламалық қамтамасыз етуді бақылауға тәуелді. Кейбір ұйымдарда, эволюция биресми үдеріс болуы мүмкін. Өзгеруге деген сауал, жүйенің қолданушыларымен және өңдеушілердің аралығында болатын қатынастардан келетін үдеріс. Басқа компанияларда, үдерістің әр кезеңіне дайын тұрған жіктелген құжаттамамен бірге формализациаланған үдерістер болады.

Өзгеру жүйесі барлық ұйымдарда жүйенің эволюциясы үшін драйвер болып табылады. Ұсыныстың өзгеруі қазіргі талаптармен енгізіле алынады, жаңа талаптар, қатысушылардан қателік туралы хабар алу, сонымен қатар бағдарламалық қамтамасыз етуді жақсарту үшін жүйенің өңдеушілерінен жаңа идеялар ұсынылды. Өзгерістерді теңестіру үдерістері және жүйенің эволюциясы циклдік болып табылады және өмір бойы жалғасады (*9.3-сурет*).



9.3-сурет. Өзгерістерді үйлестіру және даму процесі

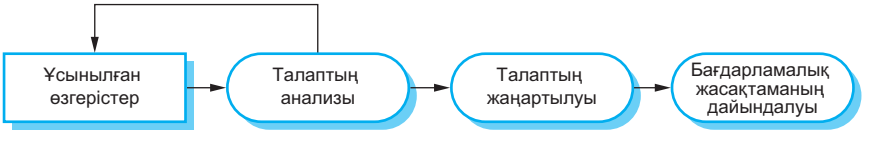
Ұсыныстың өзгеруі жүйенің компоненттерімен байланысты, яғни ұсыныстарды жүзеге асыру үшін түрлендірілген болуы тиіс. Бұл бізге құнды және бағаға әсер ететін өзгеріске мүмкіндік береді. Сол басқарманың ортақ үдерістерінің өзгертілген бөлігі, компоненттің дұрыс болжамдары, кепілдік беруі әр жүйелі жазбаға кіреді.



9.4-сурет. Бағдарлама эволюцияының үдерісі

9.4-суретте Артур (1988) ұсынған даму процесі бейнеленген. Үдеріс негізгі іс-шараларды, соның ішінде, өзгеріс анализін, жоспарларды, жүйенің енуін, сонымен қатар клиенттер үшін жүйе шығарылады. Өзгерту енгізу үшін неше жүйе өзгеріске тәуелді екенін, қанша тұратынын және өзгеріске ықпал етуі арқылы бағалауға болады. Егер ұсынылған жүйе қабылданса, жүйенің жаңа болжамының шығарылымы жоспарланады. Релиз жоспарлау барысында, барлық ұсынылған өзгерістер (ақаулықты жою, бейімделу және жаңа функционалдық) орындалады. Өзгерістер жүзеге асырылып, тексерілген соң жүйенің жаңа болжамы жасалынады. Кейін итерация үдерістерінің өзгерісі жаңа терімімен келесі релизге ұсынылады.

Сіз өзгерістің жүзеге асырылуы туралы, сонымен қатар зерттеме үдерісінің итерациясының қайда өзгертіліп, жүйеде әзірленуін, жүзеге асуын және сынақтан өтуі туралы мағлұмат аласыз. Алайда, өзгерістің орындалуының бірінші кезеңі бағдарламаны түсініп жүзеге асырылумен байланысты. Бағдарламаның бұл фазасында, сіз бағдарламаның қалай жасалғанын, қалай функционалдық қамтамасыз ететінін және ұсынылған өзгертулер бағдарламаға қалай ықпал тигізетінін түсінуіңіз керек. Сізге бұл өзгерулер жүзеге асырылып, жаңа мәселе туындатпайтынына көз жеткізуіңіз керек.



9.5-сурет. Өзгерістерді әзірлеу

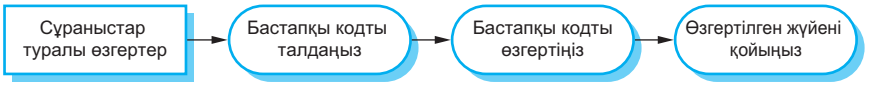
Шын мәнінде, осы үдерістің жүзеге асу сатысының ауысулары спецификациялық жүйеде өзгертілуі тиіс және *9.5-суретте* жүйеде өзгеруді бейнелеу үшін

жобалау және өткізу көрсетілген. Жүйеде өзгерулерді бейнелейтін жаңа талаптарға талдау жүргізіліп тексеріледі. Жүйелі компоненттер қайта өңделіп, жүзеге асады және бұл жүйе қайталанады. Қажет болған жағдайда, ұсынылып отырған прототиптің өзгерілуі анализ үдерісінің бір бөлігі болып табылуы мүмкін.Эволюция үдерісі, талаптарға толық талдау жүргізеді және өзгерістің зардаптары бұрын талдау үрдісінде байқалмағаны түсінікті болды. Егер клиенттер өзгертулерді талап етсе, бұл ұсынылып отырған өзгертулер енгізілуі мүмкін.

Сұраныстың өзгерісін кей-кезде шұғыл түрде шешу қажет. Осы жедел өзгерістердің туындауының үш себебі төмендегідей:

1. Жүйенің істен шығуы;
2. Егер қоршаған ортада өзгерістер болса, басқару жүйесінің жұмысын бұзатын күтпеген ақаулардың туындауы;
3. Егерде бизнесте,ойда болмаған, жүйеге әсер ететін өзгерістер орындалса, жаңа бәсекелестіктің пайда болуы немесе жаңа заңның енгізілуі.

Бұл жағдайда, өзгеріс анализінің формалдық үдеріс күйінде болуын, өзгерудің тез жасалуын білдіреді. Талаптарды және әрлендіруді өзгертудің орнына *9.6-суретте* көрсетілгендей шұғыл мәселелерді шешетін бағдарлама іске қосылды. Әйтсе де, бағдарламалық қамтамасыз етуге программа жасап шығару қауіпті болып тұр.



9.6-сурет. Төтенше жағдайындағы қалпына келу үдерісі

Апаттық жүйені жөндеуді тезірек аяқтау керек. Сіз шапшаң және тиімді шешім қабылдадыңыз, бірақ ең жақсы шешім емес,өйткені жүйенің құрылымында ақау бар. Бұл үдеріс бағдарламалық қамтамасыз етудің ескіруіне әкеп соқтырады, сонымен қатар келешекте өзгерту қиын болады және техникалық күтімге деген шығындар ұлғаяды.

Шын мәнінде, кодтың аппаттық жөнделуі өзгеріске сұраныс береді, кодтан кейін болған ақаулықтар қайтадан өңделеді. Содан соң, әр талдаудан кейін, қайтадан мұқиятырақ анықтай алады. Әрине, кодты жөндеу екінші рет қайталана алады. Талдауға көбірек уақыт бөлінген кезде, баламалы тиімді шешім табуға болады. Тәжірибеде, алайда, бұл жақсартулар төмен нәтиже беретіні сөзсіз. Жиі ұмытылатын жайт, егер жүйеге әрі қарай өзгертулер енгізілсе, онда апаттық жөндеуді қалпына келтіру мүмкін емес.

3-тарауда талқыланған Аджиль әдістері мен үдерістерін эволюциның бағдарламасы үшін және даму бағдарламасына қолдануға болады. Шын мәнінде,бұл әдістер қосымша дамуларға негізделген, әзірлеудің жылдамдығының ауысуы: эволюция жабдықтаудан кейін тігіссіз болуы керек. Жүйе өзгерген кезде, регрессияның

автоматталған сынағы пайдалы әдіс болып табылады. Өзгерістер, қолданбалы тарих және қолданушыларды еліктіру, басқару жүйесінде қажетті өзгеріс көрініс табады. Қысқаша айтқанда,эволюция икемді әзірлеуді жалғастырады.

Дегенмен, командадан өңдеушіге жұмыс берілген кезде,кейбір жағдайда мәселелер туындауы мүмкін. Екі ықтималды мәселенің ахуалдары бар:

1. Өңдеушілер команданың икемді тәсілін пайдаланды, бірақ таныс икемді әдістермен команданың эволюциясын және тәсілін қалайды. Команданың эволюциясынан эволюцияны қолдауы үшін толық құжаттама күтуге болады және бұл икемді өндірістік үдерістерде өте сирек өндіреді. Жүйеге өзгеріс енгізілу мақсатында, бұл жүйеге қойылған нақты талап емес.
2. Тәсілдеме жоспары даму үшін құрылған, бірақ команда эволюциясы икемді әдістерді қолдануды жөн санайды. Мүмкін, бұл жағдайда да, команданың эволюциясы қайта өңделген жүйеде автоматталған сынақтарды әзірлеуі қажет және кодты нөлден бастап іске қосып,икемді әзірлеуді күтеді. Бұл жағдайда да, кейбірі ширақ даму үдерісіндегі жұмсала алатын код қайтадан инженерлік жақсартуда өңделе алады.

Пул және Хюсмен (2001) бастапқы тәсілдеме жоспарының қолдануымен жасалған үлкен жүйенің сүйемелдеуі үшін экстремалды программалауды өз тәжірибелерінде қолданғандары туралы хабарлайды. Оның құрылымын жақсарту үшін жүйенің техникалық қайта жабдықтауынан кейін, үдерістің сүйемелдеуінде XP ойдағыдай қолданатын болды.

9.2. Эволюцияның серпінділік бағдарламасы

Эволюциясының серпінділік бағдарламасы өзгеру жүйесін зерттеу болып табылады. Белади және Лехман 1985 жылы бағдарламалық қамтамасыз етудің эволюциясы туралы түсініп, бар білім жүйесінің өзгеруін мақсат етіп, эмпирикалық зерттеулер қатарын өткізді. Лехман және басқалары да эволюциялық үдерістердің кері байланысын зерттеген (*9.7-сурет*).

Лехман және Белади бұл заңды құптайды, (E жүйе түріндегі) ол бағдарламалық қамтамасыз ету жүйелерінің барлық түрлері үшін шындық болып табылады. Жүйе де, бизнестің талаптарына байланысты өзгеруі мүмкін. Бизнесті құндылықпен қамтамасыз ету үшін, жүйенің жаңа релиздері жүйе үшін маңызды рөл атқарады.

Бірінші заң бойынша, жүйенің қызмет етуі, кері қайталанбайтын үдеріс болып табылғанын жариялайды. Өзгеру жүйесінің ортасына байланысты, жаңа талаптар пайда болады және жүйе жаңа талаптарға сай өзгертілуі керек.

Екінші заңда жүйенің қалай өзгергені және құрылымының нашарлағаны туралы айтылады.

Бұл тұғырықтан шығатын жалғыз ғана тәсіл бар, ол – жоспарлы-ескертпе инвестициясын жасау. Сіз бағдарламалық қамтамасыз ету құрылымының жақсартуына уақытты босқа өткізесіз, оның функционалдығына көңіл аудар-

майсыз. Анық, бұл қосымша шығындар, оның үстіне, қажетті жүйенің өзгерістерін өткізуді білдіреді.

Заң	Баяндама
Өзгерістерді жалғастыру	Шынайы ортада қолданылатын бағдарламаны міндетті түрде өзгерту, әйтпесе осы ортада пайдасы аздау болады.
Күрделіліктің үдерісі	Бағдарламаның өркендеуінің өзгерісі, ережеге сай оның құрылымы да күрделене түседі. Қосымша ресурстарды құрылымның сақталуына және жеңілдетуіне қарай бағыттау қажет.
Орасан зор бағдарламаның дамуы	Қосымшаның дамуы өзінен-өзі реттелетін үдеріс болып табылады. Жүйенің көлемі, түсінулер арасындағы уақыт және тіркелген қателердің саны секілді анықтауыштар әрбір жүйенің шамалы инварианты.
Ұйымдық тұрақтылық	Бағдарламаның барлық өмірі ішінде, оның өркендеу шапшаңдығы шамалы тұрақты болып келеді және оның жүйенің даму барысындағы ресурстарға қатысы жоқ.
Таныстықты сақтау	Жүйенің өмірі уақытында, әрбір шығару кезіндегі өсімше әрдайым біркелкі.
Жалғасатын өсім	Берілген жүйенің функционалы әрдайым пайдаланушының қанағаттанушылығын қолдау үшін артып отыру қажет.
Сапаның төмендеуі	Жүйедегі операциялық ортадағы өзгерістерді тойтару үшін өзгерістер енгізілмеген жағдайда, оның сапасы төмендейді.
Кері байланыс жүйесі	Үдерістердің дамуы өзіне көпагентті,көптүйінді кері байланыстарды қамтиды, және сіз оларға өнімнің сапасының ауқымды жақсаруына жетудегі кері байланыс ретінде қабылдауыңыз қажет.

9.7-сурет. Лехман заңдары

Үшінші заң өте қызықты және Лехманның өте даулы заңдары болып табылады.

Үлкен жүйелер даму үдерісінде динамикалық меншікке ие болады. Бұл жүйенің қызмет көрсету үдерісінің жалпы үрдісін анықтайды және жүйедегі өзгерістерге шектеу қояды. Лехман және Белади бұл заң жүйесінің өзгерулеріне шек қояды және эволюция үдерістеріне әсер ететін ұйымдастыру факторлары да, әсер еткен құрылымдық факторлар болып табылатынын болжады.

Үшінші заңда үлкен жүйелердің күрделіліктеріне әсер еткен құрылымдық факторлар туралы айтылады. Өзгерту және кеңейту бағдарламалары, оның құрылымын бұзушылыққа әкеліп соғады. Жүйелердің барлық түрлері үшін бұл

дұрыс (тек қана бағдарламалық қамтамасыз ету емес) сондықтан бұл құрылымды басқа мақсатта пайдалануға арналған. Бұл деградацияны тоқтатпаса, бағдарламаға қосымша программа енгізу қиындап кетеді. Шағын өзгерістерді енгізу, құрылымдық деградацияның дәрежесін арзандатады, жүйенің сенімділігінің маңызды мәселелерін қорыта келгенде құртады. Егер сіз үлкен өзгертулер жасайтын болсаңыз, бұл жаңа ақау туғызады. Содан соң олар бағдарламаның өзгерулеріне ингибитор жасайды.

Үшінші заңға әсер ететін ұйымдастыру факторлары, заң бойынша үлкен жүйелер ірі ұйымдармен өндіріледі. Бұл компанияларда әр жүйенің бюджетінің өзгеруіне шешім қабылдайды және шешім қабылдау үдерісін тексеретін ішкі бюрократиясы болады. Компания тәуекел етіп, өзгеру мағынасын және осымен байланысты шығындар туралы шешім қабылдау керек. Жүзеге асыруға қарағанда, өзгерту туралы шешім қабылдау үшін өзгертулер енгізілу керек, мұндай шешімдер көп уақытты талап етеді, кей кезде одан да көп уақыт керек. Ұйымдардың шешім қабылдау үдерістерін сондықтан жүйенің өзгеру жылдамдығы реттейді.

Лехманның төртінші заңының пайымдауынша, ірі программалық жобалардың көпшілігі "қанықкан" жұмыс істейді. Үшінші заңға сәйкес көптеген жағдайларда бағдарламаның дамуы басқарушылық шешімдерге бағынышты болады. Үстеме шығыстарда, команданың жұмысы үстем болатындықтан, бұл заңды бағдарламалық қамтамасыздандырудың өндеушілерінің үлкен командасы растайды.

Жүйеге жаңа функционалдық мүмкіндіктерді қосу, жүйеге жаңа ақаулықтар енгізетіні сөзсіз. Әр жаңа болжамда артық функционалдық болған сайын, қателік те арта түседі. Қорыта келгенде, шығарудың бір жүйесінде функционалдықтың үлкен өсімшесінің ар жағында жөндеу жүйесінің жаңа ақаулық релиз болғанын білдіреді. Лехманның бесінші заңы бойынша, жаңа шығарылымға аз ғана жаңа функция қосылуы керек.

Лехманның бастапқы бес заңы бас жағында аталып өтті, басқа заңдары әр жұмыстан кейін қосылып отырады. Нақты заң соңғы жұмыстардан көрінеді, бірақ бұл әлі айқын емес, өйткені бұл бағдарламалық қамтамасыз етуді әзірлеуде қалай қолданылатыны белгісіз.

Лехманның бақылауы жалпы тиімді көрінеді. Ол қызмет көрсету үдерісін жоспарлауында еленуі керек. Мысалы, бұл маркетингтің жағдайында, бір шығарылымда, жүйеге бірнеше өзгертулер енгізу керек. Бұл – жөндеудің арнаулы қателердің бір немесе бірнеше релиздері талап етпейтін ықтимал зардаптар. Сіз күнделікті өмірде кездесетін, әсіресе, дербес компьютерде, жаңа шығарылым болған кезде, артынан тез арада жаңарту болады.

9.3. Бағдарламалық қамтамасыз етуді бақылау

Бағдарламалық қамтамасыз етудің бақылап отыру, жүйенің өзгеруінің ортақ үдерісі болып табылады. Термин дамудың жеке топтарына дейін қатысады және бағдарламалық қамтамасыз етуді әзірлеуде қолданылады. Бағдарламалық қамтамасыз етуді жобалауда қатені түзету үшін түбегейлі өзгертулер жасау керек

немесе жаңа талаптары бар спецификацияны түзету үшін түбегейлі жетілдірулер керек. Өзгеру, жүйенің қазіргі компоненттерінің түрлендіруі жолымен өткізілген және қажеттіліктің жанында, сонымен қатар жүйеде жаңа компоненттердің қосымшасы арқылы жасалған.

Техникалық бағдарламалық қамтамасыз етудің үш әртүрлі түрлері бар:

1. Кодтағы қателікті жөндеу арзанырақ, жобалаудағы қателікті жөндеу қымбатырақ, себебі көптеген бағдарламалық компоненттерді қайтадан жазуды қажет етеді. Қатені қалпына келтіру өте қымбат болып табылады.
2. Қызмет етудің бұл түрі,бағдарламалық қамтамасыз етудің басқару жүйесінің өзгертілуін талап етеді. Жүйеге деген ұсыныс өзгертілуі тиіс.
3. Ауқымды өзгерістер басқа қызметтерге қарағанда, бағдарламалық қамтамасыз ету үшін аса қажет.

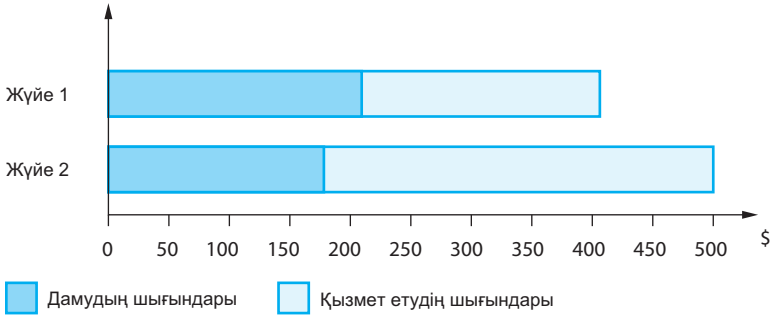
Тәжірибеде бұл қызмет етудің арасында айқын айырмашылық болмайды. Ақаулар бағдарламалық қамтамасыз етуде жиі болады, өйткені қолданушылар жүйені пайдаланғанда күтпеген жайтқа кездеседі. Жүйені өзгерту, жетіспеушілікті жөндеу үшін ең жақсы тәсіл болып табылады. Бұл қызмет етудің түріне адамдар әртүрлі есімдер береді. Техникалық күтімнің кемшіліктерін жою үшін барлық жерде “Жөндеу қызметтері” пайдаланылады. Әйтсе де, “Адаптивті бақылап отыру” кейде жаңа ортаға бейімделуді, кейде жаңа талаптарға бағдарламалық қамтамасыз етудің бейімделуін білдіреді. «Жетілген күту» жаңа талаптардың енуі – өзге жағдайлардағы бағдарламалық қамтамасыз етудің жетілдіру жолын, кейде жүйенің функционалдығының сақтауын білдіреді. Бұлардың нақты атаулары болмағандықтан, мен бұл терминдерді қолданбауға тырыстым.

Техникалық күтіммен және дамудың арасын бақылап отыратын бағдарламалық қамтамасыз етудің бірнеше зерттеулері бар. Терминологияда айырмашылық болғандықтан, бұл зерттеулерді толықтықтай салыстыра алмаймыз.

9.8-сурет. Жалғастыруға жұмсалған күштің бөлінуі

Технологияда өзгерулер және қосымшалардың әртүрлі домендері 1980 жылдан эволюцияның күштерін тарату кезінде өзгерулерге ие болғанын көрсетеді. Жаңа жетілулерге қарағанда, бюджетте бағдарламалық қамтамасыз ету үлкен орын алады (үштен екісі қызмет ету, үштен бірі даму). Қателерді түзетуге қарағанда, бюджеттің көптеген бөлігі жаңа талаптарды жүзеге асыруға шығындалатыны белгілі болды. *9.8-суретте* шамамен қызмет етуге кеткен шығындар бейнеленген. Нақты пайыз, басқа бір ұйымнан өзгереді, бірақ белсенділіктің қызмет ету жүйесінде ақаулықтар туындаған жағдайда сол ұйым арқылы қайтадан жөндеуден өтеді. Жаңа ортамен және жаңа өзгертілген талаптар жүйенің эволюциясын қолданып жұптаса қызмет жасайды.

Қызмет ету және дамуға деген салыстырмалы шығындар жаңа бір қолданбалы аймақтан басқаға түрленеді. Гимарайнш (1983) айтуынша бағдарламалық қамтамасыз етуге кеткен шығын жүйенің дамуына ықпал еткен. Нақты уақыттың ішінде жасалған жүйелердің шығындары, әзірлеуге қарағанда төрт есе артық өскен. Биік сенімділік және өнімділік үшін модуль тығыз байланыста болуы керек және өзгертілуі тиіс. Бұл бағалардың тұрғанына 25 жылдан астам уақыт болды, жүйенің әртүрлі түрлері үшін таратудың құны айтарлықтай өзгеретініне күмән бар.



9.9-сурет. Әзірлеу және жалғастыру шығындары

Бұл, әдеттегідей болашақ өзгеріс шығындарын азайту мақсатында және жүйенің өткізуінде күш салу үшін экономикалық тиімді болып отыр. Жаңа функцияларды қосу қымбаттырақ болады, себебі сізге жүйені зерттеуге және ұсынылатын өзгерістердің әсеріне уақыт жұмсауыңызға тура келеді. Қорыта келгенде, программалық эволюцияның програмаңы түсіну және өзгерту арқылы азайтуға болады. Кескінді әзірлеу және инженерлік әдісті басқару, сондай дәл спецификация қолдану, жақсы бағдарламалық қамтамасыз ету объектіге өңдеу, қызмет етудің шығындарының төмендетуіне мүмкіндік туғызады. Жүйеге талдаудың және сынақтың шығындарын азайту үшін мультипликация жасаудың әсері бар. Бірінші жүйе үшін, қосымша шығындар, жүйені басқарылатындай ету үшін инвестицияланады. Бұл жүйенің қызмет ету мерзімінің пайызды ұлғаюы жүйенің төмендетілген пайызымен дәлме дәл. Қызмет етуді жеңілірек ету экономикаға тиімді болып табылғандықтан, бұл бағалар болжамды болып есептеледі. Бұл икемді әзірлеуде қайтадан факторинг үшін дәлелдеу керек. Қайтадан екінші рет факторингсіз, кодты өзгерту күрделірек және қымбат. Әйтсе

де, программаға қосымша жақсартулар енгізулер сирек кездеседі Жөндеуге келетін инвестиция қысқаша шығынның өсуіне әкеліп соқтырады.

Мұрагерлік жүйелер

Мұрагерлік жүйелер әлі де қолданыста пайдалы және кей жағдайларда бизнес операциялар үшін маңыздылығы жоғары болып келетін ескі жүйелер болып табылады. Олар көнерген тілдерді және технологияларды немесе жүргізуге қымбат түсетін басқа да жүйелерді пайдалану арқылы орындалуы мүмкін. Жиі жағдайда олардың өзгеріс пен құжаттама арқылы нашарланған құрылымы жоқ болып немесе жарамдылық мерзімінен өтіп кеткен болып шығады. Қалай болғанда да, бұл осы жүйелерді алмастыруға еш әсерін тигізбеуі қажет. Бұл жүйелер тек жылдың белгілі бір уақытында қолданылуы мүмкін немесе спецификациясы жоғалатындықтан, оларды алмастыру да тым қауіпті болуы ықтимал.

http://www.SoftwareEngineering-9.com/Web/LegacySys/

Әдеттегідей, функционалдық жүйенің пайдалануынан кейін қосу қымбатырақ және де әзірлеудің үдерісі функционалды болып келеді. Бұның себептері:

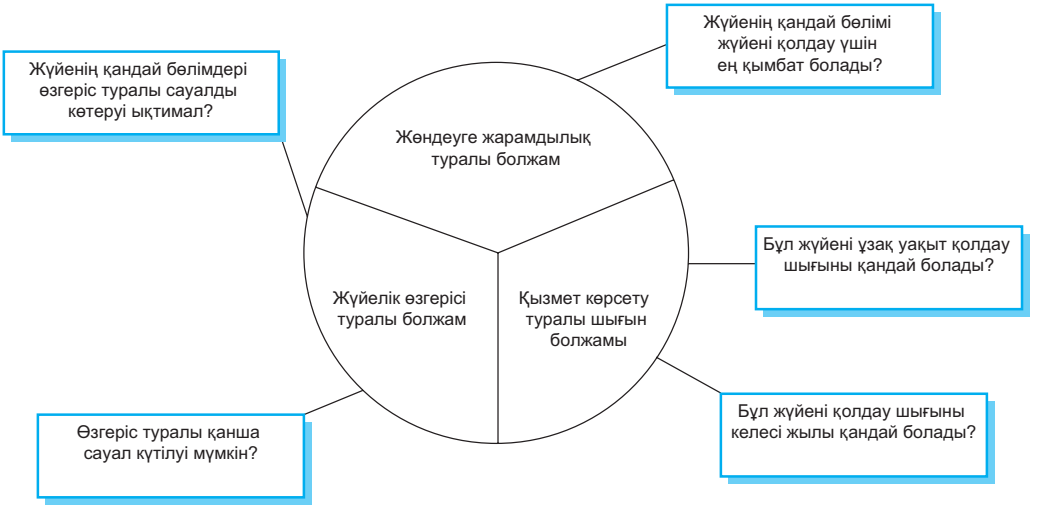
1. Тұрақтылық жүйе үшін қалыпты жағдай. Жаңа команда немесе тұлғалар, техникалық күтімдер жүйесіне жауаптылар жүйені түсінбейді. Жүйені енгізудің алдында қазіргі жүйені түсінуге уақыт жұмсау керек.
2. Әдеттегідей жүйені сақтаудың шарты жүйенің әзірлеу шартынан бөлек. Жүйенің өңдеуші түпнұсқасынан басқасын, келісімшарт арқылы басқа компанияға беруге болады. Бұл фактор, бағдарламалық қамтамасыз етуді бақылап отыратын команданың жоқтығын білдіреді. Егер бұл қиын бағдарламалық қамтамасыздандыру болса және де болашақта өзгертетулер енгізу керек болса, өңдеушілердің командасы үшін айрықша орын алулары керек.
3. Қосымшаның доменімен жұмыс жасау саласында қызмет көрсетуші тәжірибесіз. Бағдарламалық қамтамасыздандыру жаман бейнемен белгілі. Ол жүйе даму үдерісінде кіші қызыметшіге дейін жиі қаралады. Одан басқа, ескі жүйелер, кейде көнерген бағдарлама тілдерінде жазылады. Қызмет көрсетуші мүмкін, бұл тілдермен жұмыс жасауда тәжірибесіз және жүйемен жұмыс жасау үшін келесі тілдерді үйренуі керек.
4. Бағдарламаға өзгерістер енгізген кезде, бағдарламаның құрылымы бұзушылыққа тенденция алады. Демек, бағдарламалардың ескі болған сайын, өзгерту қиындай түседі. Кейбір жүйелер, ескі бағдарламалық қамтамасыз ету әзірлеуімен жасалды. Мүмкін олар жақсы жіктелмеген болар, мүмкін, түсіну үшін емес, тиімділік үшін оптимизицияланған болар. Құжаттама жүйесі қарама-қайшы да жоғалуы мүмкін. Ескі жүйелер, мүмкін, кескінді қатал басқаруға келмейді, уақытты жаңа дұрыс компоненттерге жұмсауға тура келеді.

Құжаттама

Жүйе құжаттамасы орындаушыларды жүйенің құрылымы мен ұжымы және жүйе пайдаланушысына ұсынылатын мүмкіндіктер туралы ақпаратпен қамтамасыз ету арқылы үдерісті орындау барысында көмегін тигізеді. Әйткенмен, XP сияқты жылдам және икемді тәсілдердің жақтаушылары кодты негізгі құжаттама деп тұжырымдайды, жоғары дәрежедегі жоба үлгілері мен тәуелділіктер мен шектемелер туралы ақпарат сол кодтарды түсінуді жеңілдетеді және оларды өзгерте алады. Мен жүктеп алуыңызға болатын құжаттама тақырыбына арналған бөлек тарау жазып шықтым.

http://www.SoftwareEngineering-9.com/Web/ExtraChaps/Documentation.pdf

Алғашқы үш мәселе, ұйымдардың дамуы мен сүйемелдеуін жеке шара ретінде қарауымен байланысты. Техникалық күтім белсенділік ретінде қарастырылады және жүйенің өзгеруіне ақша жұмсаған тиімсіз. Бұл мәселенің жалғыз ұзақ мерзімді шешімі жүйені белгілі бір мерзім аралығында пайдалануға болады. Мен басында атап өткендей, сіз жүйелер туралы ойлауыңыз керек, өмір бойы үздіксіз даму үдерісінде болуыңыз қажет. Төртінші сұрақ, азғындалған жүйе құрылымдары, бұл мәселенің оңтайлы шешімі. (Сипаттау бұдан арғы бөлімде) Қайтадан инженерлік әдістер бағдарламалық қамтамасыз ету жүйесінің құрылымын қолдануға болады. Архитектуралық өзгертулер жүйені жаңа жабдыққа өзгерте алады. Рефакторинг жүйенің кодын және оңай өзгертулер жасауды сапаландырады.



9.10-сурет. Жалғастыруды болжау

9.3.1. Болжаулар, техникалық қызмет көрсету

Жүйенің қандай бөліктерінде өзгерістер болатынын және қызмет көрсетуде қандай күрделілік туындайтынын болжап білуіңіз керек. Сонымен бірге осы уақыт аралығында жүйенің техникалық қызмет көрсетуге кеткен ортақ шығыстарын бағалауыңыз керек. *9.10-сурет* болжауды және сонымен байланысты сұрақтарды көрсетіп тұр.

Жүйенің өзгеруіне қажетті болжаулы сандар жасалған, жүйе мен сыртқы ортаның арасындағы өзара байланысты түсіну керек. Кейбір жүйелер төңіректегі сыртқы жүйемен және ортасымен өте күрделі қатынаста болады, бұл жүйенің өзгеруіне әкеліп соғады. Жүйемен және оның ортамен арасындағы байланысқа баға беру үшін, сіз төменде келтірілген талаптарды ескеруге тиістісіз:

1. Саны және интерфейс жүйелерінің күрделілігі, саны жағынан интерфейстан көп және бұл күрделі интерфейстер, интерфейстегі өзгерістер жаңа талаптармен жасалады.
2. Жүйелік талаптардың мәндері үлкен болады, мен *4-бөлімде* атап өткенімдей, ұйымдастыру талапты саясат пен процедураны көрсетеді.
3. Жүйеде қолданылатын кәсіпкерлік-үдеріс көп болған сайын, сол жүйедегі өзгеріске талап көбірек болады.

Көптеген уақыт аралығында, зерттеушілер бағдарламаның күрделілігін қарастырды, осы көрсеткіштер бойынша айтатын болсақ, бұл жөндеуге келетін (МакКейб 1976) цикломатикалық күрделілік (Банкер және т.б. 1993; Колеман және т.б. 1994; Кафура және Редди 1987; Козлов және т.б., 2008). Бұл таңғаларлық жағдай емес, зерттеулердің көрсетуі бойынша, жүйе немесе компонент күрделі болған сайын, қызмет көрсетуі де қымбаттай түседі. Өлшемдердің күрделілігі әсіресе, программалық компоненттер үшін пайдалы. Кафура және Редди (1987) жүйелердің компоненттер қатарын қарап шығып, бір нәрсе тапты, қызмет көрсетуге кеткен шығындар күрделі компоненттермен топтанады. Қызмет көрсетуге кеткен шығындарды азайту үшін, сіз күрделі компоненттерді қарапайым компоненттерге алмастыруыңыз керек. Жөндеуге келетін келесі үдерістің метрикалық мысалдары:

1. Техникалық қызмет көрсетудің жоспардан тыс қателіктерінің үлкеюі және есептемеуді қабылдамау қателер бағдарламаға еніп жатыр дегенді білдіреді. Бұл жөндеудің төмендегенін көрсетеді.
2. Орташа уақыт ықпалды талдау үшін және қажет программалық компоненттердегі өзгерісті көру үшін қажет. Бұл барлық компоненттердің зақымдануына әкеледі және жөндеу жарамдылығы төмендейді.
3. Орташа уақыт, қажетті өзгеріске деген сауалды жүзеге асыру үшін қажет. Қандай компоненттер зардап шеккенін бағалаған соң, жүйені және құжаттаманы өзгерту үшін белгілі бір уақыт беріледі.

4. Уақыттың үлкеюі, қажетті өзгерісті жүзеге асыру үшін қажет, бірақ бұл жөндеу жарамдылығын төмендеуін көрсетеді.
5. Көп уақыт аралығында, өзгеруге деген мүдденің артуы жөндеу жарамдылығының төмендеуін білдіреді.

Сіз қызмет көрсетуге кеткен шығындарды болжау үшін жүйелерді бақылап отыруға, өзгерістерге және болжауларға болжаған сауалдар туралы мәліметті қолдана аласыз. Шығынды бағалау үшін жетекшілердің көпшілігі сезгіштікпен және тәжірибемен бұл мәліметтерді топтастырады. Құндар бағалары (Бем және т.б. 2000), *24-бөлімде* аталып өткендей, программалық қамтамасыз етулерге кеткен техникалық қызмет көрсетудің шығындарын, қазіргі кодты және жаңа кодты жасаумен түсіндіріледі.

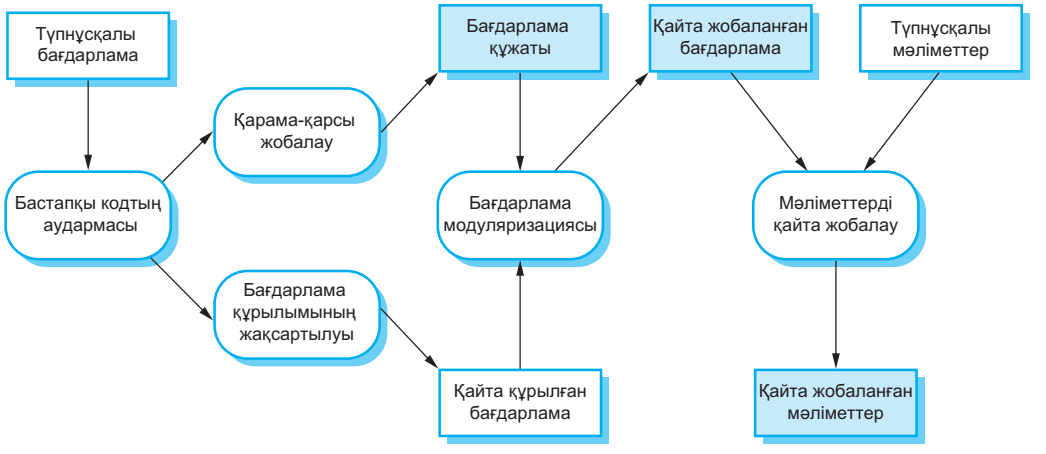
9.3.2. Программалық реинжинирингті қамтамасыз ету

Мен алдыңғы бөлімде айтқандай, өзгерту үшін бағдарламаны түсіну қажет, содан соң жүзеге асыру керек. Бірақ көптеген мұра етілген жүйелерді түсіну және өзгерту қиын. Бағдарлама орындалуы немесе түсінікті болғандықтан оңтайлы болған шығар немесе уақыт өте келе,бағдарламаның бастапқы құрылымы, өзгерістердің қатарынан зақымданған шығар.

Мұра етілген бағдарламалық қамтамасыз ету жүйелері қарапайым қызмет көрсету үшін, сіз олардың құрылымдарын және жүйелерін қайтадан жобалай аласыз. Жүйелерді қайтадан құжаттауды және бағдарламаны жаңа бағдарламалық тілге аударуды, сонымен қатар өзгерістерді және құрылымдарды жаңартуды реинжиринг қамтиды. Программалық қамтамасыз етудің функционалдық мүмкіндіктері өзгермейді және жүйелердің архитектурасындағы маңызды өзгерістерден құтылу үшін сіз оларды сынап көруіңіз керек.

Реинжинирингтан екі маңызды пайда бар, бұл алмастыру емес:

1. Тәуекелдің төмендеуі кризистік маңызды кәсіпкерлік-программалық қамтамасыз етуде, қайта жабдықтауда биік тәуекел болып тұр. Қателер жүйелердің спецификациясында жасалуы немесе дамытудың мәселелері болуы мүмкін. Жаңа программалық қамтамасыз етуді енгізудегі тоқтаулар кәсіпкерлікте жоғалып жатыр дегенді білдіреді және бұл қосымша шығыстар залал алып келді.
2. Жаңа программалық қамтамасыз етулерді әзірлеуге кеткен шығындарға қарағанда, реинжиниринг шығындарының төмендеуі едәуір аз болуы мүмкін. Ульрих (1990) шығыстарды қайтадан іске асыру үшін 50 миллион доллар керек, ал жүйені табысты етіп 12000000 долларға қайтадан құруға болады деп мысал келтірген болатын.



9.11-сурет. Қайта әзірлеу үдерісі

9.11-сурет реинжиниринг үдерісінің ортақ үлгісі болып табылады. Бұл үдеріске үлес қосқан мұрагерлік бағдарламалар жақсартылған және де сол бағдарламамен қайта өңделген. Бұл үдерістегі реинжиниринг қызметі төмендегі нұсқаларды қамтиды:

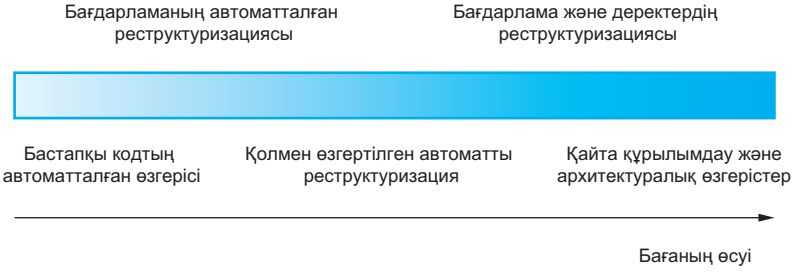
1. Аудармадағы құралдардың көмегімен қазіргі код ескі программалық тілден бір тілдің қазіргі нұсқасына немесе басқа тілге өзгертілген.
2. Бағдарламаларды кері жобалау талданып және одан мәліметтер алынған. Өз ұйымдарына және функционалдық құжаттауға көмектеседі. Тағы, бұл үдеріс әдетте, толық автоматтандырылған.
3. Жеңіл оқу және түсіну үшін, бағдарламаның құрылымы талданып өзгертілген. Ішінара бұл автоматты болуы мүмкін, бірақ кейбір қол кірісулер, әдетте, керек болып жатады.
4. Модульдік бағдарламалар, бағдарламаның сабақтас бөліктері бірге топталады. Кейбір жағдайларда, бұл рефакторингке қосады (мысалы, бір жүйе бірнеше қойманы қолдану үшін жасалуы мүмкін).
5. Бағдарламадағы өзгерістерді бейнелеу үшін бағдарламаның өзгертіліп шығуы осы реинжинирингке байланысты. Осы базалардағы сызбаларды қайтадан қарап шығуды білдіреді және қазіргі базаларды осы жаңа құрылымға өрнектеу қажет. Сіз әдетте, мәліметтерді тазартуға тиістісіз. Мысалы, өз қателіктерін іздестіру, дұрыстау және қайталанатын жазуларды алып тастау, т.б. Бұл реинжиниринг деп аталады.

Барлық қадамдар *9.11-суретте* көрсетілгендей, реинжиниринг бағдарламасы міндетті түрде болуын талап етпейді. Егер сіз қосымша программалаулар тілін әлі қолдансаңыз, сізге бастапқы кодты аудару қажет емес. Егер сіз барлық автоматтандырылған реинжинирингті істей алсаңыз, сізге кері жобалаудың

көмегімен құжаттаманы қалпына келтіру маңызды емес. Реинжиниринг жүйедегі құрылымдарды уақытында өзгерту үшін қажет.

Басқа компоненттерді қолдану мақсатында, жүйелік программалық қамтамасыз етулердің бастапқы интерфейстерін жабу керек және жаңасын ұсыну қажет. Бұл ораушы мұра жүйесі үлкен масштабпен дүркін қолданатын сервистерді әзірлеу үшін маңызды техника болып саналады.

Реинжинирингқа кеткен шығындар істелген жұмыспен тікелей байланысты.



9.12-сурет. Қайта әзірлеу шақтары

9.12-суретте көрсетілгендей, реинжинирингке ықтимал етудің толық спектрі бар. Бастапқы кодты аудару ең оңай жол болып табылады, шығындар осылай сол жақтан оңға жаққа қарай өсіп жатады. Бөлікті реинжиниринг сәулеттік көші-қон сияқты ең қымбат болып табылады.

Программалық қамтамасыз етумен реинжиниринг мәселесінде шектеулер болуы мүмкін. Бұл мүмкін емес, мысалы, жүйені өрнектеу үшін объекті-бағдарланған жүйе функционалдық тұрғыдан қолданылып жазылуы тиіс. Маңызды сәулетті өзгерістер немесе радикалдық жүйелерді басқару автоматты түрде орындалмайды, сондықтан олар өте қымбат. Реинжиринг жөндеу жүйелерін жақсарта алады.

9.3.3. Алдын алатын рефакторингқа қызмет көрсету

Бағдарламаға жақсартулар енгізу үдерісін, рефакторинг арқылы бағдарламаны жақсарту мақсатымен, оның күрделілігін кішірейту керек немесе жеңіл түсіну мақсатымен жасау керек (Опдайк және Джонсон 1990). Рефакторинг кейде Объектілі-бағдарланған әзірлеулермен шектеліп қалады, бірақ қағидаларды кез келген дамытуға қолдануға болады. Сіз бағдарламаны қайта ұйымдастыру кезінде, функционалдықты қосуға тиісті емессіз, бірақ бағдарламаны жақсартуға талпынуыңыз керек. Сондықтан рефакторингті келешектегі өзгертулерді қысқартатын «алдын алу қызмет көрсету» деп айтуға болады.

Реинжинирингте және рефакторинг та бағдарламалық қамтамасыздандыруды оңай түсінуге және де өзгертуге бағытталған. Сіз қызмет көрсетуде жеңілірек келген жаңа жүйені жасап өңдеу үшін реинжиниринг мұра еткен жүйені автоматтандырылған түрін қолдана аласыз. Рефакторинг – дамытулардағы және

қалыпты дамудағы барлық үдерістің жақсаруын ұсынып отыр. Бұл кодтың азып-тозуларынан және құрылымнан құтылу үшін жасалған. Бұл әдістер өзгерістің айналасында негізделген.

Бағдарламаның сапасы, сондықтан тез арада құлдырайды, өңдеушілер өз бағдарламаларын жиі қайта ұйымдастырады. Рефакторингтің көмегімен қателерді түзетуге болады. Кез келген қате, сынау кезінде шығу керек.

Фаулер және т.б. (1999) болжауы бойынша, кейбір код бағдарламасын жақсартатын жағдайлар бар (оларды «жаман иістер» деп атайды). Рефакторингтің көмегімен жақсартуға болатын жаман иістердің нұсқалары:

1. Қайталанатын кодты және өте ұқсас кодтарды бір пакетте сақтауға болады. Мысалы, алып тастауға және бір әдіс сияқты жүзеге асыруға немесе қажеттілік шара бойынша функцияны шақыруға болады.
2. Егер әдіс өте ұзын болса, ол қысқа әдістерге өзгертілуі тиіс.
3. Бұл жиі қайталанатын кодтар қосқыштың кейбір құндылығына бағынышты. Қосқыш бағдарламаның қасында болуы мүмкін. Жан-жақты бағдарланған бағдарламалауда сіз полиморфизмді жиі қолдансаңыз болады.
4. Осы жиналып қалулар, жабысулар бағдарламада жиі кездеседі (кластарда, параметрлерде). Барлық, инкапсуляциялық мәліметтерді алмастыруға болады.
5. Келешекте өңдеушілер ортақтықтарды бір бағдарламаға қосады. Кейде жай ғана алып тасталынуы мүмкін.

Фаулер өз кітабында және веб-сайттарында кейбір қарапайым жекешілік бойынша қолдануға болатын рефакторинга өрнектеулерін жеке дара немесе жұптаса отырып, жағымсыз иістен құтылу үшін ұсынып отыр. Бұл өрнектелген мысалдар, қайталанғандарды шеттетуді және жаңа әдістерді жасауды қосады. Сіз тестердің тізбегін және әдістерді жоғары тартасыз, сіз супер кластың бір әдісімен класс тармағындағы ұқсас әдістерді алмастырасыз. Дамытуларды, тұтылуды сондай сияқты өз редакторларыңызға рефакторингті қосасыз.

Рефакторингтің көмегімен бағдарлама әзірлеуде, бағдарламаларға қызмет көрсетуде біршама шығындар азайды. Рефакторинг үшін техникалық қызмет көрсетудің нашарлауы мүмкін емес жағдай. Сіз сонымен бірге қымбатырақ болса да, рефакторингтің дизайны туралы ұмытпауыңыз керек. Дизайн – рефакторингтің анықтау үлгілерін, жобаларын білдіреді *(7-бөлім)* және бұл жобалаулар үлгілерін іске асыру үшін кодпен қазіргі кодты алмастыру керек (Кариевский 2004).

9.4. Басқарулар жүйелерінің мұрасы

Жаңа жүйелерге технологиялық үдерістердің қазіргі қолдануымен игерілген программалық қамтамасыз етулерді дамытуды және қалыпты дамулардың жүйелерін интегралдау керек сияқты. Сонымен қатар, кәсіпкерлік жүйелер болып

табылатын, әлі де мұра етілген көп жүйелер бар. Олар ұлғаймалы болуы тиіс және өзгеретін электрондық іскер операцияларға бейімделуі керек.

Олар қолданған мұра етілген жүйелердің ережесі портфел сияқты, ұйымдардың көпшілігі, бұл жүйелерді сүйемелдеу және жаңғырту үшін бюджетпен шектелген. Өз инвестициялардан максимал серпуді қалай алу керектігін шешу қажет. Ескі болжамдар бойынша, олардың жүйелерінен реалистік бағалар жасау және даму үстіндегі бұл жүйелер үшін өте қолайлы стратегия болып табылады. Төрт стратегиялық нұсқалар бар:

1. Жүйе кәсіпкерлік үдерістерде тиімді үлеспен істемеген кезде, бұл опция таңдалуы тиіс. Кәсіпкерлік үдерістер өзгертіледі, уақыт өте келе жүйе орнатылады және олар бұрынғы жүйеге тәуелді болмайды.
2. Жүйені өзгерістерсіз және бұрынғыдай техникалық қызмет көрсетумен қалдырыңыз, бұл нұсқаларды жүйе бұрынғыдай керек болған кезде, бірақ жеткілікті тұрақты болғанда және жүйені қолданушылар өзгеріске аз сауалдар жібергенде ғана таңдау керек.
3. Бұл опцияны жүйенің сапасы төмендеген кезде және жүйеге бұрынғыдай жаңа өзгертулер ұсынылған кезде таңдаған дұрыс. Бұл үдеріс интерфейстің жаңа компоненттерін әзірлеуін қоса алады, бастапқы жүйе басқа жүйемен жұмыс істей алады.
4. Барлығын алмастыру немесе жаңа жүйемен жүйелер бөлігін ғана алмастыру. Бұл нұсқаны ескі жүйе өзінің қызметін жалғастыра алмайтын кезде немесе сатудағы жүйелер жаңа жүйелерді өңдеуде қолжетімді бағалар қойған кезде таңдау керек. Көптеген жағдайларда, жүйелердің негізгі компоненттері қайтадан алмастыруға мүмкіндік бар жерде жүйелермен алмастырыла алады.

Шын мәнінде, бұл нұсқалар бір-бірін жоққа шығармайтын ұғымдар. Жүйе бірнеше бағдарламалардан тұрған кезде, әртүрлі нұсқалар әрбір бағдарламаға қолданыла алады.

Мұра етілген жүйелерді бағалаған кезде, сіз кәсіпкерлік және техникалық көзқараспен қарауға тиістісіз (Уаррен 1998). Кәсіпкерлік көзқараспен қараған кезде, сіз кәсіпкерлік жүйені қажетсіне ме, жоқ па соны шешуіңіз керек. Техникалық көзқараспен қараған кезде, сіз бағдарламалық қамтамасыз етуді, бағдарламалық жүйені және аппараттық қамтамасыз етуді бағалауыңыз керек. Өз шешіміңізді хабарлау үшін, кәсіпкерліктің сапасын және жүйелердің құндарын қолдана аласыз.

Мысалы, ұйымда 10 мұра етілген жүйелер бар деп айтайық. Сіз әрбір жүйелердегі кәсіпкерліктің сапасын және құнын бағалауға тиістісіз. Сіз кәсіпкерліктің құнын және жүйенің сапасын диаграмма түрінде көрсете аласыз. Бұл *9.13-суретте* көрсетілген.

9.13-суреттен сіз, жүйенің 4 кластері бар екенін көре аласыз:

1. *Кәсіпкерліктің төмен сапасы, төмен құны.* Бұл жүйелер пайдаланымда қымбат болатынын ескере отырып және бұның кәсіпкерлікте қарқындауы жеткіліксіз. Бұл жүйелер жойылуы тиіс.
2. *Кәсіпкерліктің төмен сапасы, жоғары құны*. Бұл жүйелер кәсіпкерлікке маңызды үлестерін қосады, сондықтан олар жойыла алмайды. Бірақ олардың төмен сапасы қайтадан инвестициялау дегенді білдіреді. Олардың сапасын жоғарылату үшін, бұл жүйелерді қайтадан құру қажет. Егер қолайлы жүйе бар болса, оларды алмастыруға болады.
3. *Кәсіпкерліктің жоғары сапасы, төмен құны.* Бұл жүйе кәсіпкерлікке үлкен үлес қоспайды, оның қызмет көрсетуі қымбат болуы мүмкін. Егер қымбат өзгерістер керек болмаса, осылай жалғаса беруі мүмкін, егер қымбат өзгерістерді қажет етпесе және аппараттық жүйе бұрынғыдай қолданылса, оларды өзгерту керек емес. Егер қымбат өзгерістер қажет бола түссе, бағдарламалық қамтамасыз етуді болдырмау керек.
4. *Кәсіпкерліктің жоғары сапасы, жоғары құны.* Бұл жүйелер пайдаланымда сақталуы тиіс. Олардың жоғарғы сапасын сақтап қалу үшін, сіз өзгертулерді инвестицияға салуға тиісті емессіз. Дегенмен, жүйелердің биік деңгейде қызмет көрсетуі жалғасуы тиіс.

9.13-сурет. Жүйенің дұрыстығын бағалау мысалы

Жүйенің кәсіпкерлік құнына баға беру үшін сіз жүйелердің қатысушыларын анықтауға тиістісіз, жүйелердің түпкі қолданушыларын және олардың жетекшілерін, сондай-ақ жүйе туралы сұрақтар қатарын енгізу керек. Талқылау үшін, сізде негізгі төрт сұрақ бар:

1. *Жүйені қолдану*. Егер жүйелер белгілі бір мерзімге дейін қолданылса немесе адамдардың саны аз болса, бұл кәсіпкерліктің төмендігін білдіреді. Мұра етілген жүйелер, мүмкін кәсіпкерлікті қанағаттандыру үшін игерілген

шығар. Сіз жүйелерді қолданған кезде абайлауыңыз керек. Дегенмен, ол кәсіпкерлік үшін маңызды жүйелер болып табылады.

2. *Демеулік ететін кәсіпкерлік-үдерістер.* Жүйе енгізілген кезде, кәсіпкерлік-үдерістер жүйені қолдана алады. Егер жүйе икемсіз болса, кәсіпкерлік-үдерістің өзгеруі мүмкін емес жағдай. Төңіректегі орталардың өзгеруінен, кәсіпкерлік-үдерістер ескіруі мүмкін. Сондықтан жүйе кәсіпкерліктің төменгі сатысында болады және бұл кәсіпкерлікті тиімсіз қолдануға мәжбүрлейді.
3. *Жүйенің сенімділігі және кәсіпкерлік-есептер.* Егер жүйе сенімді болмаса және мәселелер кәсіпкерлік-клиенттерге тікелей ықпал етсе немесе кәсіпкерліктегі адамдарды есептер алаңдатса, шешім қабылдау үшін, жүйе кәсіпкерлік үшін маңызды болмайды.
4. *Жүйелердің шығуы.* Маңызды сұрақтардың бірі, кәсіпкерлікте табысқа жұмыс жасау үшін жүйелердің шығындары маңызды болып табылады. Егер кәсіпкерлік бұл шығындарға тәуелді болса, жүйе кәсіпкерлік үшін жоғары құндылыққа ие болады. Егер сирек қолданылатын жүйе шығарылса, бұл кезде саудалық құны төмен болуы мүмкін.

Фактор	Сұрақтар
Тасымалдаушы тұрақтылығы	Тасымалдаушы бар ма? Тасымалдаушы ақшалай тұрақты ма? Және де тасымалдаушыға әзірлеу үдерісіне қатысу ұнай ма? Тасымалдаушының толықтай үдеріске қатыса алмаған жағдайында, жүйе әзірлеуді жалғастыратын басқа тасымалдаушы табыла ма?
Сәтсіздіктің дәрежесі	Жүйеде сәтсіздіктердің жоғарғы деңгейдегі құжаттандырылуы бар ма?
Ескілік	Бағдарламаға және техникалық қамсыздандыруға қанша жыл болды? Жүйені жаңғыртудың көптеген экономикалық және кәсіптік тиімділіктері бар.
Өнімділік	Жүйенің өнімділігі қажет деңгейде ме? Жүйенің өнімділігінің мәселелері пайдаланушыларға әсер ете ме?
Жалғастырудың құны	Техникалық қамсыздандыруды және бағдарлама лицензияларын жалғастырудың шығындары қандай? Ескі техникалық қамсыздандырудың шығындары жаңа техникаға қарағанда әлдеқайда жоғары болады.
Үйлесімділік	Жүйенің басқа жүйелермен үйлесуінің мәселелері бар ма? Мысал ретінде компиляторлардың әртүрлі операциялық жүйелермен үйлесіп, мәселесіз жұмыс атқаруы.

9.14-сурет. Қоршаған ортаны бағалау факторлары

Мысалы, серіктестік туристік жүйелік тапсырыстарды пайдалынымға береді, бұл қызметкерлердің көмегімен, осы сапарды ұйымдастыруға міндетті қызметкерлер арқылы іске асады. Олар туристік агент арқылы бекітілген тапсырыстарды орналастыра алады. Билеттерді жеткізген соң, серіктестік оларға есепшотты шығарып қояды. Дегенмен кәсіпкерліктің құны тұтынушылардың қанша пайызы қолданылатынымен бағаланады. Бұл сапарлар арзан және ыңғайлы, адамдар шаруаны тікелей өз веб-сайттары арқылы алуға болады. Бұл жүйе қолданылуы мүмкін, бірақ оны ұстаудың ешқандай нақты мағынасы жоқ. Функционалдық мүмкіндіктер сыртқы жүйелерден де қолайлы.

Керісінше, серіктестік клиенттердің барлық тапсырыстарын және тауарлардың ретінің өзгеруі туралы хабарламаны жіберетін жүйе шығарды. Бұл клиенттер үшін тиімді, өйткені олар барлығынан хабардар болып отырады. Бизнестің мұндай жүйеден шығуы маңызды, демек кәсіпкерлік үшін бұл үлкен жетістік болып табылады.

9.14-суретте көрсетілгендей, техникалық көзқарастан программалық жүйеге баға беру үшін, жүйенің жұмыс істегенін және төңіректегі орталардың қолданылуын есепке алу керек. Төңіректегі орта маңызды болғандықтан, аппараттық және операциялық жүйені жаңарту төңіректегі ортаның өзгеруінің нәтижесінде болады.

Фактор	Сұрақтар
Түсініктілік	Берілген жүйенің кодын түсіну қаншалықты қиын? Жүйені бақылау мәзірі қаншалықты күрделі? Кодтағы айнымалылардың аттары айқын ба және олар өзінің функцияларын орындайды ма?
Құжаттандыру	Жүйе қалай құжаттандырылады? Құжаттама дайын ба, дұрыс па?
Дерек	Деректердің айқын үлгісі бар ма? Қайталанатын деректер бар ма? Жүйенің пайдаланып жатқан деректері жаңа ма?
Жалғасымдық	Жүйені жалғастыратын мүмкіндік қандай? Жалғастыру мәселелерінің пайдаланушыларға әсері қандай?
Бағдарламалау тілі	Жүйені әзірлеуге қолданылған бағдарламалау тілімен бүгінгі компиляторлар жұмыс істей ала ма?
Құрылымды басқару	Барлық бөлімдердің барлық нұсқалары құрылымды басқару жүйесімен әзірленді ме?
Деректерді тестілеу	Жүйені тестілейтін деректер бар ма?
Жеке қабілеттілік	Қосымшаны жалғастыруға қабілеті жететін мамандар бар ма? Жүйемен жұмыс жасаған, тәжірибесі бар адамдар бар ма? Осы адамдардың барлығы қолжетімді ме?

9.15-сурет. Қосымшаны бағалау факторлары

Жүйенің экологиялық үдерісте қызмет көрсетуін және оның өлшемдерін жасауға тиістісіз. Мысалы, аппаратты шығыстар, жаңартуларды қолдану және аппаратты программалық қамтамасыз етулерді жасау.

Төңіректегі ортаға баға беру үшін, міндетті түрде *9.15-суретті* қарап шығу керек. Назар аударыңыз, барлығы төңіректегі орталардың техникалық мінездемелері емес. Сіз, сонымен бірге аппаратты және программалық қамтамасыз етуші жабдықтаушылардың сенімділігін есепке алуға тиістісіз. Техникалық сапаға баға беру үшін, сіз, ең бірінші сенімділікті, сүйемелдеудегі қиындықтарды және жүйелік құжаттаманы бағалауыңыз керек. Сапаны бағалауда пайдалы болуы мүмкін:

1. Жүйені өзгерту деген сауалдар, әдетте, жүйені бұзады, құрылымды және қосымша өзгерістер енгізу қиын болады, жинаған мән жоғары болған сайын, жүйелердің сапасы төмендейді.
2. Қолданбалы интерфейстердің саны. Жүйелердегі форма негізінде бұл өте маңызды фактор, әрбір форма жеке қолданбалы интерфейс сияқты қарала алады. Бірнеше интерфейстерде, сол көбірек ықтималдық етеді, бұл интерфейстер сәйкессіздіктерден және жүйеден шығаруы мүмкін болады.
3. Жүйе қолданатын көлемді деректер жүйенің сапасын төмендетеді.

Идеалды, нақты бағаларды қорытындылар үшін қолдану керек және мұра етілген жүйемен не істеу керектігін ойластыру қажет. Көптеген жағдайлар, шешімдер саяси пікірлер негізінде құрылған, бірақ бұл басты мақсат емес. Егер екі кәсіпорынды бірлестіретін болсақ, бұл өте саяси қуатты әріптеспен әріптес болу үшін оның жүйелерін ұстау керек және басқа жүйелерден бас тарту керек. Егер жоғарғы ұйымның басшысы жаңа апараттық платформаға көшуге шешім қабылдаса, бұл қосымшаны ауыстыруды талап етеді. Егер нақты бір жылда жүйені өрнектеу үшін бюджет жоқ болса, жүйеге техникалық қызмет көрсетуі жалғастырылады.

НЕГІЗГІ ТҰЖЫРЫМДАР

- Программалық қамтамасыз етулерді және қалыпты дамуларды спираль үлгінің қолдануымен көрсетілгендей, интеграция ретінде қарауға болады.
- Программалық қамтамасыз етуге кеткен шығындар, әдетте, программалық қамтамасыз етулерді әзірлеулерге кеткен шығындардан асып жатады.
- Программалық қамтамасыз етудің қалыпты даму үдерістері өзгеріске ұшырап жатады, өзгерістерді жоспарлауларға және өзгерістерді енгізулерге және шығарылымға талдау жасалады.
- Лехман заңдарының көпшілігі үздіксіз өзгерістер туралы және жүйенің ұзақ мерзімді қалыпты дамуын сипаттау жайында.

- Бағдарламалық қамтамасыз етудің негізгі үш түрі бар, атап айтқанда, өзгерістегі қателіктер, жаңа шарттарда жұмыс жасау үшін программалық қамтамасыз етулерді өзгерту және жаңа енгізулер немесе талаптарды өзгертулер.
- Программалық қамтамасыз етудің реинжинирингі реструктуризациямен байланысты және оны жеңіл түсіну және өзгерту үшін, программалық қамтамасыз етулерді құжаттау қажет.
- Рефакторинг ақпаратқа аз ғана өзгерістер жасай отырып, оның функционалдығын сақтайды, профилактикалық қызмет ретінде қарауға болады.
- Жүйені алмастыру үшін, түрлендіру немесе қолдау үшін кәсіпкерліктің дәстүрлі жүйесінің құнын және ақпараттық қосымша сапасының құнын бағалап, көз жеткізу керек.

ҚОСЫМША ӘДЕБИЕТТЕР

'Software Maintenance and Evolution: A Roadmap'. As well as discussing research challenges, this paper is a good, short overview of software maintenance and evolution by leading researchers in this area. The research problems that they identify have not yet been solved. (V. Rajlich and K.H. Bennett, *Proc. 20th Int. Conf. on Software Engineering*, IEEE Press, 2000.) http://doi.acm.org/10.1145/336512.336534.

Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices. This excellent book covers general issues of software maintenance and evolution as well as legacy system migration. The book is based on a large case study of the transformation of a COBOL system to a Java-based client-server system. (R. C. Seacord, D. Plakosh and G. A. Lewis, Addison-Wesley, 2003.)

Working Effectively with Legacy Code. Solid practical advice on the problems and difficulties of dealing with legacy systems. (M. Feathers, John Wiley & Sons, 2004.)

ЖАТТЫҒУЛАР

9.1. Неліктен бағдарламалық қамтамасыз ету өзгертілуі тиіс немесе біртіндеп пайдасыз бола бастайтынын түсіндіріңіз.

9.2. Лехман заңдарының логикалық дәлелдеуін түсіндіріңіз. Қандай жағдайларда заң қате болып саналады?

9.3. *9.4-суретте* көрсетілгендей, программалық қамтамасыз етудің үдерісін пайдалана отырып қандай өзгерістер енгізуге болатынын айтыңыз.

9.4. Теңіздегі мұнай өнеркәсіп үшін программалық қамтамасыз етулерді әзірлейтін менеджері ретінде сіз игерілген серіктестіктерді жөндеуге ықпал ететін факторларды айқындайтын есепті алдыңыз. Қызмет көрсету үдерісін талдау үшін бағдарламаны қалай жасауға болатынын және сіздің серіктестігіңіз үшін жөндеуге келетін лайықты көрсеткіштерді ұсынуыңыз керек.

9.5. Программалық қамтамасыз етулердің негізгі түрлерін қысқаша сипаттаңыз. Неліктен кейде айырмашылықты анықтау қиын?

9.6. Реинжиниринг жүйелерінің құнының ықпал ететін қандай негізгі факторлар бар?

9.7. Қандай жағдайда ұйымдар жүйеден бас тартады, жүйенің бағасы шамамен көрсетілгенде ме немесе бұл жоғарғы сапа және бизнестің жоғарғы құны болған кезде ме?

9.8. Ескі жүйенің қалыпты дамуларының қандай стратегиялық нұсқалары бар.

9.9. Неліктен ақпаратты қолдау мәселелері мұра етілген жүйелермен алмастыру дегенді блдіретінін түсіндіріңіз.

9.10. Өзгертілетін және сақталатын кодты жасау бағдарламалық қамтамасыз ететін инженерлердің кәсіби жауапкершілігі ме, егер бұл жұмыс берушіге тікелей айтылмаған болса?

ПАЙДАЛАНЫЛҒАН ӘДЕБИЕТТЕР

Arthur, L. J. (1988). *Software Evolution*. New York: John Wiley & Sons.

Banker, R. D., Datar, S. M., Kemerer, C. F. and Zweig, D. (1993). 'Software Complexity and Maintenance Costs'. *Comm. ACM*, **36** (11), 81–94.

Boehm, B. W., Abts, C., Brown, A. W., Chulani, S., Clark, B. K., Horowitz, E., Madachy, R., Reifer, D. and Steece, B. (2000). *Software Cost Estimation with COCOMO II*. Upper Saddle River, NJ: Prentice Hall.

Coleman, D., Ash, D., Lowther, B. and Oman, P. (1994). 'Using Metrics to Evaluate Software System Maintainability'. *IEEE Computer*, **27** (8), 44–49.

Erlikh, L. (2000). 'Leveraging legacy system dollars for E-business'. *IT Professional*, **2** (3), May/June 2000, 17–23.

Fowler, M., Beck, K., Brant, J., Opdyke, W. and Roberts, D. (1999). *Refactoring: Improving the Design of Existing Code*. Boston: Addison-Wesley.

Guimaraes, T. (1983). 'Managing Application Program Maintenance Expenditures'. *Comm. ACM*, **26** (10), 739–46.

Hopkins, R. and Jenkins, K. (2008). *Eating the IT Elephant: Moving from Greenfield Development to Brownfield*. Boston: IBM Press.

Kafura, D. and Reddy, G. R. (1987). 'The use of software complexity metrics in software maintenance'. *IEEE Trans. on Software Engineering*, **SE-13** (3), 335–43.

Kerievsky, J. (2004). *Refactoring to Patterns*. Boston: Addison Wesley.

Kozlov, D., Koskinen, J., Sakkinen, M. and Markkula, J. (2008). 'Assessing maintainability change over multiple software releases'. *J. of Software Maintenance and Evolution*, **20** (1), 31–58.

Krogstie, J., Jahr, A. and Sjoberg, D. I. K. (2005). 'A longitudinal study of development and maintenance in Norway: Report from the 2003 investigation'. *Information and Software Technology*, **48** (11), 993–1005.

Lehman, M. M. (1996). 'Laws of Software Evolution Revisited'. Proc. European Workshop on Software Process Technology (EWSPT'96), Springer-Verlag. 108–24.

Lehman, M. M. and Belady, L. (1985). *Program Evolution: Processes of Software Change*. London: Academic Press.

Lehman, M. M., Perry, D. E. and Ramil, J. F. (1998). 'On Evidence Supporting the FEAST Hypothesis and the Laws of Software Evolution'. Proc. Metrics '98, Bethesda. Maryland: IEEE Computer Society Press. 84–8.

Lehman, M. M., Ramil, J. F. and Sandler, U. (2001). 'An Approach to Modelling Long-term Growth Trends in Software Systems'. Proc. Int. Conf. on Software Maintenance, Florence, Italy: 219–28.

Lientz, B. P. and Swanson, E. B. (1980). *Software Maintenance Management*. Reading, Mass.: Addison-Wesley.

McCabe, T. J. (1976). 'A complexity measure'. *IEEE Trans. on Software Engineering.*, **SE-2** (4), 308–20.

Nosek, J. T. and Palvia, P. (1990). 'Software maintenance management: changes in the last decade'. *Software Maintenance: Research and Practice*, **2** (3), 157–74.

Opdyke, W. F. and Johnson, R. E. (1990). 'Refactoring: An Aid in Designing Application Frameworks and Evolving Object-Oriented Systems'. 1990 Symposium on Object-Oriented Programming Emphasizing Practical Applications (SOOPPA '90), Poughkeepsie, New York.

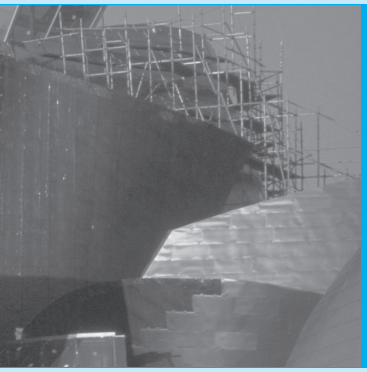
Poole, C. and Huisman, J. W. (2001). 'Using Extreme Programming in a Maintenance Environment'. *IEEE Software*, **18** (6), 42–50.

Rajlich, V. T. and Bennett, K. H. (2000). 'A Staged Model for the Software Life Cycle'. *IEEE Computer*, **33** (7), 66–71.

Sousa, M. J. (1998). 'A Survey on the Software Maintenance Process'. 14th IEEE International Conference on Software Maintenance (ICSM '98), Washington, D.C.: 265–74.

Ulrich, W. M. (1990). 'The Evolutionary Growth of Software Reengineering and the Decade Ahead'. *American Programmer*, **3** (10), 14–20.

Warren, I. E. (1998). *The Renaissance of Legacy Systems*. London: Springer.



2 БӨЛІМ

Функционалдық сенімділік және қауіпсіздік

Бағдарламалық жүйелердің өлшемі мен кешенділігі үнемі ұлғайып отыратындықтан, бағдарламалық қамтамасыз етуді әзірлеу кезінде кездесетін негізгі проблема – аталған жүйелерге біз сеніп тапсыра алатындай дәрежеге қол жеткізу болып табылатынына мен нақты сенемін. Жүйеге сену үшін біз ол қажет болған жағдайда қолжетімді болатынына, ал оның жұмысқа қабілеттілігі біздің үмітімізге сәйкес болатынына сенімді болуымыз керек. Біздің компьютерлеріміз немесе деректеріміз оның салдарынан қатерге ұшырамауы тиіс. Бұл жүйенің функционалдық сенімділігі және қауіпсіздігі жүйе функционалдығы нюанстарына қарағанда үлкен маңызға ие екенін білдіреді. Осылайша, оқулықтың аталған бөлігі студенттерді және тәжірибеден өтуші бағдарламалық қамтамасыз ету әзірлеушілерді функционалдық сенімділік және қауіпсіздік маңызды тақырыптарымен таныстыруға арналған.

Бұл бөлімнің бірінші тарауында 10-тарауда бір қарағанда бағдарламалық қамтамасыз етудің функционалдық сенімділігімен ортақ ештеңесі жоқ әлеуметтік-технологиялық жүйелер сипатталады. Алайда, қауіпсіздіктің және функционалдық сенімділіктің көптеген ақауларының себебі операторлар және ұйымдастырушылық кемшіліктер болып табылады. Сондықтан, біз жүйенің функционалдық сенімділігі және қауіпсіздігін талдау кезінде оларды айналып өте алмаймыз. Бағдарламалық қамтамасыз етуді әзірлеушілер тіпті ең жақсы әдістер мен технологиялар жүйенің абсолюттік функционалдық сенімділігі және қауіпсіздігін қамтамасыз ете алмайтынын білуге және есте сақтауға міндетті.

11-тарауда біз функционалдық сенімділік пен қауіпсіздіктің негізгі тұжырымдамаларымен таныссамыз, сонымен қатар, сенімді жүйелер құруға пайдаланылатын анықтау және қалпына келтіру, алдын алудың негізгі принциптерін ұғынамыз. Жүйенің функционалдық сенімділігі және қауіпсіздігіне қойылатын талаптарды анықтау үшін пайдаланылатын арнайы тәсілдер сипатталатын 12-тарау талаптарды әзірлеу тақырыбы ашылған 4-тарауды толықтырады. 12-тарауда біз оқырмандарды ресми айрықшаманы пайдалану әдістерімен қысқаша таныстырамыз. Ғаламторда Сіз осы мәселеге арналған қосымша тарауды таба аласыз.

13 және 14-тарауларда функционалдық сенімді және қауіпсіз жүйе құру үшін пайдаланылатын бағдарламалық қамтамасыз етуді әзірлеу әдістері сипатталған. Функционалдық сенімділікті қамтамасыз ету және қауіпсіздікті қамтамасыз етуді біз жеке-жеке қарастырамыз, алайда оларға ортақ нәрселер көп. Біз бағдарламалық қамтамасыз ету архитектурасының маңыздылығын талқылаймыз және функционалдық сенімділік пен қауіпсіздікке қол жеткізуге көмектесетін бірқатар жобалау бойынша ұсынымдарды және бағдарламалау әдістерін ұсынамыз. Сонымен қатар, біз ақаулар мен сыртқы шабуылдарға төтеп беруге қабілетті жүйелер құру үшін құралдардың молшылығы мен алуан түрлілігі неге маңызды екенін түсіндіреміз. Сол сияқты, біз жүйелердің қауіпсіздігіне қатер төнгенде олардың қызмет көрсетуін жалғастыруға мүмкіндік беретін бағдарламалық қамтамасыз етудің бас тартуға табандылығы немесе тіршілік қабілеті жайында өзекті тақырыпты көтереміз.

Осы бөлімнің соңғы тарауы – 15-тарау функционалдық сенімділікті және қауіпсіздікті қамтамасыз етуге арналған. Онда жүйені верификациялау және қателерді табу мақсатында модельді тексеруді және статикалық талдауды пайдалану ерекшеліктері түсіндіріледі. Аталған әдістер критикалық жүйелерді әзірлеу кезінде табысты қолданылады. Біз сонымен қатар, жүйенің функционалдық сенімділігі және қауіпсіздігін сынауға айрықша тәсілдерді қарастырамыз, сол сияқты, функционалдық сенімділік туралы қасиеттің бақылаудың сыртқы органдары өкілдерін жүйенің қауіпсіздігі мен сенімділігіне сендірудің қажетті құралы бола алатынын түсіндіреміз.

10

Әлеуметтік технологиялық жүйелер

Мақсаттары

Бұл тарау сіздерді «әлеуметтік-технологиялық жүйе» (адамдарды, бағдарламалық қамтамасыз етуді және ақпараттық қамтамасыз етуді байланыстыратын жүйе) түсінігімен таныстыруға, сонымен қатар, қауіпсіздік пен функционалдық сенімділікті жүйенің көзқарасынан қарастыру қажеттігін көрсетуге арналған. Бұл тарауды оқып болғаннан кейін Сіз:

- әлеуметтік-технологиялық жүйе түсінігінде не ұғынылатынын білесіз, компьютерлік, техникалық және әлеуметтік-техникалық жүйелер арасындағы айырмашылықты түсінесіз;
- жүйенің интеграциялық қасиеттерімен: жүйенің сенімділігі, функционалдығы, қауіпсіздігі және қорғанысымен танысасыз;
- Жүйені жобалау үдерісінің бөлігі болып табылатын сатып алу, әзірлеу және пайдалану операциялары туралы білесіз;
- жүйенің қорғанысы және функционалдық сенімділігі неге жекелей қарастырылмауы тиіс екенін және жүйе кемшіліктерінің (мысалы, оператордың қателіктері) оларға қалай әсер ететінін түсінесіз.

Мазмұны

Компьютерлік жүйеде бағдарламалық қамтамасыз ету және ақпараттық қамтамасыз ету өзара тығыз байланысты. Ақпараттық қамтамасыз етусіз бағдарламалық қамтамасыз ету тек абстракция, адамның кейбір білімі мен идеяларының қарапайым көрінісі болып табылады. Бағдарламалық қамтамасыз ету болмаса ақпараттық қамтамасыз ету тек пайдасыз қондырғылар жинағы болып қалады. Алайда, егер Сіз жүйе құру мақсатымен біріктіретін болсаңыз, күрделі есептеулерді жүзеге асыруға және нәтижелерін ұсынуға қабілетті машина жасап шығарасыз.

Бұл жүйенің іргетастық сипаттарының бірін көрсетеді – бұл жай ғана бөлшектер жиынынан әлдеқайда үлкен нәрсе. Жүйелер қасиеттерге ие болады, бұл қасиеттер олардың компоненттері интеграцияланып бірге жұмыс істеген кезде байқалады. Сол себепті, бағдарламалық қамтамасыз етуді жобалау – қандай да бір ерекшеленген әрекет емес, неғұрлым күрделі жүйелерді әзірлеу үдерістерінің құрамдас бөлшегі. Бағдарламалық жүйелер – ерекшеленген жүйелер емес, белгілі әлеуметтік немесе ұйымдастырушылық мақсаттарға қызмет ететін неғұрлым кең жүйенің маңызды компоненттері.

Мысалы, жабайы табиғат учаскесінде орнатылған метеожүйенің бағдарламалық қамтамасыз етілуі метеостанцияның бақылау-өлшеу аспаптарын реттейді. Ол басқа да бағдарламалық жүйелермен байланысты жүзеге асырылады және ауа-райын болжаудың әлдеқайда қомақты ұлттық және халықаралық жүйелерінің бөлшегі болып табылады. Ақпараттық не бағдарламалық қамтамасыз ету болсын, аталған жүйелер оларды басқаратын және жұмыс нәтижелерін талдайтын адамдар мен ауа-райын болжау үдерістерінің бар болуын білдіреді. Жүйе сонымен қатар, жеке тұлғаларға, мемлекетке, өнеркәсіптік кәсіпорындарға және т.б. ауа-райы болжамын ұсыну мақсатында жүйеге сенім артатын ұйымдардың болуын жобалайды. Бұл көлемді жүйелерді кейде әлеуметтік-технологиялық жүйелер деп атайды. Олардың құрамына адамдар, үдерістер, нормалар және т.б. технологиялық емес элементтер, сонымен қатар, компьютерлер, бағдарламалық қамтамасыз ету және басқа да құрал-жабдықтар сияқты технологиялық компоненттер кіреді.

Әлеуметтік-технологиялық жүйелердің күрделілігі сондай, оларды тұтас алғанда түсіну практикалық тұрғыдан мүмкін емес. Әдетте олар *10.1-суретте* көрсетілгендей қабаттар түрінде қарастырылады. Аталған қабаттар әлеуметтік-технологиялық жүйе блоктарын құрайды:

1. *Құрал-жабдықтар қабаты.* Аталған қабат кейбірі компьютерлер болып табылуы мүмкін ақпараттық құрылғылардан тұрады.
2. *Операциялық жүйе қабаты.* Аталған қабат ақпараттық қамтамасыз етумен өзара әрекет етеді және жүйенің жоғары бағдарламалық қабаттарының жалпы құралдарының жинағы болып табылады.
3. *Алмастыру және деректерді басқару қабаты.* Берілген қабат операциялық жүйе құралдары мүмкіндіктерін ұлғайтады және функционалдығы жоғары өзара әрекеттестікті жүзеге асыруға көмектесетін интерфейстің болуын қамтамасыз етеді (мысалы, қашықтағы жүйеге қолжетімділік, жүйенің деректер базасына және т.б. қол жетімділік). Кейде аталған қабатты ол қосымша

мен операциялық жүйе арасында орналасқандықтан, платформааралық бағдарламалық қамтамасыз ету деп те атайды.

4. *Қолданбалы деңгей.* Аталған қабат нақты қолдану ерекшелігіне бағдарланған қажетті функционалдықты қамтамасыз етеді.
5. *Өндірістік үдеріс қабаты.* Аталған деңгейде ұйымдастырушылық өндірістік үдерістер анықталады және жүзеге асырылады, олар үшін бағдарламалық жүйе қолданылады.
6. *Ұйымдастыру қабаты.* Аталған қабатқа жоғарғы деңгейдегі стратегиялық үдерістер, сонымен қатар, жүйені пайдалану кезінде сақталуы қажетті өндірістік қағидалар, саясат пен нормалар кіреді.
7. *Әлеуметтік қабат.* Бұл қабатта жүйені пайдалануға негіз болып табылатын әлеуметтік нормалар мен заңдар анықталады.

10.1-сурет. Әлеуметтік-технологиялық жүйе блогы

Теориялық түрде, өзара әрекеттестіктердің көпшілігі көршілес қабаттар арасында жүзеге асырылады, әрбір қабат жоғарғы қабаттан төменгі қабат туралы толық ақпаратты жасырып тұрады. Бірақ, тәжірибе жүзінде әрқашан бұлай бола бермейді. Қабаттар арасында күтпеген әрекеттестіктер туындауы мүмкін, бұл жалпы жүйе үшін проблема туындауына алып келеді. Мысалы, жеке ақпаратқа қол жеткізуді реттейтін жаңа заң шықты делік. Аталған заң әлеуметтік қабатта туындаған. Оның туындауы жаңа ұйымдастырушылық үдерістер әзірлеуге және өндірістік үдерістердің өзгерулеріне әкеп соғады. Алайда, кез келген қолданбалы жүйе қажетті құпиялық деңгейін ұсына алмайды, сондықтан алмасу және деректерді басқару қабатына өзгерістер енгізу қажет.

Бағдарламалық қамтамасыз етудің қорғалуын және функционалдық сенімділігін зерттеу кезінде бағдарламалық қамтамасыз ету жұмысын жекелей талдамай, тұтас жүйе туралы ойлаған маңызды. Бағдарламалық қамтамасыз етудегі ақаулар салдары әдетте қауіпті емес, себебі бағдарламалық қамтамасыз ету бұзылған жағдайда да, оны жеңіл және арзан қалпына келтіруге болады. Алай-

да, бағдарламалық қамтамасыз етудің аталған ақаулары жүйенің басқа бөліктеріне тараса, олар бағдарламалық қамтамасыз етудің физикалық және адами ортасына ықпалын тигізеді. Бұл жағдайда ақау салдары әлдеқайда маңызды. Адамдарға орын алған ақауды өтеу немесе алдын алу үшін қосымша жұмыстарды атқаруға тура келеді; мысалы, құрал-жабдық физикалық зақымдануы мүмкін, деректер жоғалтылуы немесе зақымдануы ықтимал, құпиялық бұзылуына байланысты белгісіз салдарға алып келуі мүмкін.

Сондықтан да, егер Сіз әзірленетін бағдарламалық қамтамасыз етуге жақсы қорғанысты және функционалдық сенімділікті қамтамасыз еткіңіз келсе, онда оны жүйе деңгейі тұрғысынан қарастыру қажет.

Жүйенің басқа да элементтері үшін бағдарламалық қамтамасыз ету ақауларының салдарын түсіну қажет.

Сол сияқты, жүйенің басқа элементтерінің бағдарламалық қамтамасыз етудің ақауларына қалай алып келетінін, олардың БҚ аталған ақауларының алдын алуға немесе өтеуге қалай көмектесе алатынын түсіну керек.

Осылайша, бағдарламалық қамтамасыз ету ақауына қарағанда жүйе ақауы әлдеқайда елеулі проблема болып табылады. Бұл Сізге бағдарламалық қамтамасыз етудің оның тікелей ортасымен қалай өзара әрекеттестікте екенін тексеріп, сол арқылы төмендегіні кепілдендіру үшін қажет:

1. Бағдарламалық қамтамасыз етудегі ақаулар жүйенің тек көршілес қабаттарына ғана әсер етеді, барлық басқа қабаттарының жұмысына аса елеулі әсерін тигізбейді. Атап айтқанда, бағдарламалық қамтамасыз етудегі ақаулар жүйелік ақауларға алып келмеуі тиіс.
2. Сіз жүйенің блогы бағдарламалық емес қабаттарындағы ақаулар мен олқылықтар бағдарламалық қамтамасыз етуге әсер ету мүмкіндігін түсінесіз. Сіз сонымен қатар, аталған ақаулықтарды анықтауды жеңілдету үшін қандай тексерулер жүргізу керектігі, қалпына келтіру үшін қандай қолдау көрсетілуі қажеттігі туралы ойлануыңыз тиіс.

Бағдарламалық қамтамасыз ету иілгіш қасиетке ие болатындықтан, бағдарламалық қамтамасыз етуді әзірлеушілерге күтпеген жүйелік проблемаларды жиі шешуге тура келеді. Радиолокациялық қондырғының орналасуына қарай радиолокациялық түсірімдерде бөтен бейнелер пайда болатынын елестетіп көріңізші.

Мұндай кедергілерді жою үшін қондырғыны басқа учаскеге ауыстыру тиімсіз шешім болып табылады, сондықтан жүйені әзірлеушілерге бөтен бейнелерді жоюдың басқа тәсілін ойлап табу қажет. Мүмкін, олардың шешімі бағдарламалық қамтамасыз етудің бейнелерді өңдеу қуатын жақсарту болуы мүмкін. Алайда, мұндай шешім тұтас алғанда бағдарламалық қамтамасыз етудің жұмысын тежеп, жұмысқа қабілеттігін жол берілмейтін деңгейге дейін төмендетуі мүмкін. Аталған жағдайда, проблеманы «бағдарламалық қамтамасыз етудің қатесі» деп сипаттауға болады, ал негізінен бұл жалпы жүйені жобалау барысында жіберілген қате болып саналады. Мұндай әзірлеушілер кездесетін ақпараттық қамтамасыз

етудің құнын ұлғайтпай қуатты арттыру проблемалары кең тараған. Осылай аталатын бағдарламалық қамтамасыз ету қателері бағдарламалық қамтамасыз етудің ішкі проблемаларының салдары болып табылмайды, жүйе талаптарына сәйкестікті қамтамасыз ету мақсатында бағдарламалық қамтамасыз етуді өзгерту әрекеттерінің нәтижесі болып саналады. Мұндай жағдайға тамаша мысал ретінде Денвер аласы әуежайында жүкті тасымалдау жүйесіндегі ақауды келтіруге болады (Суорц, 1996), онда бағдарламалық қамтамасыз етуді басқарушыға пайдаланылатын құрал-жабдықтың шектеулерімен жұмыс істеуге тура келген.

Жүйені жобалау (Стивенс және басқ., 1998; Тайер, 2002; Том, 1993; Уайт және басқ., 1993) – бұл жүйенің бағдарламалық қамтамасыз етілуін әзірлеу ғана емес барлық жүйені әзірлеу үдерісі. Бағдарламалық қамтамасыз ету аталған жүйелердің бақылаушы және интеграциялаушы элементі болып табылады, ал бағдарламалық қамтамасыз етуді әзірлеу шығындары жалпы жүйені жобалау шығындарының үлкен бөлігін құрайды. Бағдарламалық қамтамасыз етуді әзірлеушіге бағдарламалық қамтамасыз етудің басқа бағдарламалық және ақпараттық жүйелермен өзара қалай әрекеттесетінін, сонымен бірге, БҚ қалай пайдалану көзделетінін түсіну қажет. Аталған мәліметтер Сізге бағдарламалық қамтамасыз етудің шектеулерін түсінуге, неғұрлым тиімді БҚ әзірлеуге және жалпы жүйені жобалау үдерісіне қатысуға көмектеседі.

10.1 Күрделі жүйелер

«Жүйе» термині барлық жерде де жиі қолданылады. Біз компьютерлік жүйелер, операциялық жүйелер, төлем жүйелері, білім жүйесі, өкімет жүйесі және басқа да жүйелер туралы айтып отырмыз. Әрбір жағдайда «жүйе» сөзі әртүрлі мағынаға ие, бірақ олардың бәріне ортақ бір сипаты бар: жүйе әр уақытта тек оның бөлшектері жиынтығына қарағанда, маңызды нәрсе болып табылады.

Абстрактілік жүйелер (мысалы, үкіметтік жүйелер) аталған кітаптың шегінен тыс қалады. Біз құрамына компьютерлер кіретін, белгілі мақсатқа ие (мысалы, байланысты қамтамасыз ету, навигациялық қолдау, еңбекақыны есептеп шығару) жүйелерге басты назар аударамыз. Жүйелердің мұндай түрлеріне төмендегідей анықтама беруге болады:

> *Жүйе – белгілі мақсатқа қол жеткізу мақсатында жиынтық түрде жұмыс істейтін түрлі типтегі өзара байланысты компоненттердің әдейі интеграциялануы.*

Аталған анықтама жүйенің қомақты диапазонына таратылады. Мысалы, лазерлік көрсеткіш сияқты қарапайым жүйе бірнеше ақпараттық компоненттерден және қарапайым бағдарламалық басқару құралынан тұруы мүмкін. Керісінше, әуе қозғалысын басқару жүйесі бірнеше мыңдаған бағдарламалық және ақпараттық компоненттерден тұрады, сонымен қатар, аталған компьютерлік жүйеден алынатын ақпараттар негізінде шешім қабылдайтын адамның қатысуын талап етеді.

Барлық күрделі жүйелер үшін компоненттерінің қасиеттері мен әрекетінің өзара үздіксіз байланыстылығы тән. Жүйенің жеке алғанда әрбір компонентінің табысты қызмет атқаруы басқа компоненттерінің қызметіне байланысты. Осылайша, бағдарламалық қамтамасыз ету процессор жұмыс істеген жағдайда ғана жұмыс істейді. Ал процессор өз кезегінде есептеулерді тек қажетті есептеулерді анықтайтын бағдарламалық қамтамасыз ету сәтті орнатылса ғана жүзеге асырады.

Күрделі жүйелер әдетте иерархияға ие және басқа да көптеген жүйелерден тұрады. Мысалы, бұйрық берудің әскери жүйесі құрамына әскери қимылдардың орналасқан орны туралы ақпарат ұсынатын географиялық ақпарат жүйесі және т.б. күруі мүмкін. Мұндай жүйелерді «қосалқы жүйелер» деп атайды. Қосалқы жүйелер тәуелсіз жүйелер ретінде дербес жұмыс істей алады. Мысалы, сол географиялық ақпараттық жүйе көлік логистикасы немесе шұғыл әрекет ету жүйелерінде де сәтті қолданылуы мүмкін. Құрамына бағдарламалық қамтамасыз ету кіретін жүйелер екі санатқа бөлінеді:

1. *Техникалық компьютерлік жүйелер.* Аталған санат құрамына ақпараттық және бағдарламалық компоненттер кіретін, бірақ процессор мен процедуралар жоқ жүйелерден құралады. Техникалық жүйелерге теледидарлар, ұялы телефондар және басқа да бағдарламалық қамтамасыз етілген құрал-жабдықтар мысал бола алады. Компьютерлік ойындардың және т.б. бағдарламалық жүйелерінің басым бөлігі аталған санатқа жатады. Жеке тұлғалар мен ұйымдар техникалық жүйелерді белгілі мақсатпен қолданады, алайда, берілген мақсатты түсіну жүйе бөлігі болып табылмайды. Мысалы, біз пайдаланатын мәтіндік редактор өзінің кітаптар жазу үшін қолданылатынын білмейді.
2. *Әлеуметтік-технологиялық жүйелер.* Аталған жүйелер құрамына бір немесе одан да көп техникалық жүйелер кіреді, алайда, әрқашан жүйе ішінде жүйенің пайдалану мақсатын түсінетін адамдардың қатысуын қажет етеді. Әлеуметтік-технологиялық жүйелер белгілі операциялық үдерістерді жүзеге асырады, ал адамдар (операторлар) жүйенің ажырамас бөлігі болып табылады. Олар ұйымның қағидалары мен саясатын басқарады. Сонымен қатар, оларға ұлттық заңнама және нормативтік саясат сияқты сыртқы шектеулер ықпал ете алмайды. Мысалы, берілген кітап құрамына түрлі үдерістер мен техникалық жүйелер кіретін әлеуметтік-технологиялық баспа жүйесінің көмегімен шығарылды.

Әлеуметтік-технологиялық жүйелер – іскерлік мақсаттарды (мысалы, сату көлемінің ұлғаюы, өндірісте қолданылатын шикізат шығынының қысқаруы, салық жинау, әуе кеңістігінің қауіпсіздігін бақылау және т.б.) жүзеге асыруға көмектесуге арналған кәсіпорын масштабы жүйелері болып табылады. Аталған жүйелер ұйымдастырушылық ортаға біріктіріліп орнатылғандықтан, бұл жүйелерді сатып алуға, әзірлеуге және пайдалануға ұйымның саясаты және процедуралары, сонымен қатар, оның жұмыс мәдениеті ықпал етеді. Жүйенің пайдаланушылары

ұйымды басқару қалай жүзеге асырылатыны, ұйымның ішінде және одан тыс жерде адамдармен өзара әрекеттестігі әсер ететін адамдар болып табылады.

Әлеуметтік-технологиялық жүйелерді әзірлеу кезінде Сізге олар пайдаланылатын ұйымдастырушылық ортаны түсіну қажет. Егер мұндай түсінік болмаса, жүйе өзіне қойылатын талаптарды қанағаттандырмауы, ал оның пайдаланушылары жүйені пайдаланудан бас тартулары мүмкін.

Жүйе арасындағы талаптарға және құрылымға, әлеуметтік-технологиялық жүйе қызметіне әсер етуі мүмкін ұйымдастырушылық факторлар:

1. *Үдерістің өзгеруі.* Жүйе ортаның жұмыс үдерісіне өзгерістер енгізуді талап етуі мүмкін. Аталған жағдайда қызметкерлерді тиісті түрде оқытуды жүргізу қажет. Егер мұндай өзгерістер айтарлықтай болса немесе нәтижесінде ұйымдағы жұмыс орындарын қысқартуды көздесе, онда пайдаланушылар мұндай жүйені енгізуге қарсы болуы мүмкін.
2. *Жұмыстың өзгеруі.* Жаңа жүйелер кәсіпорындағы білікті жұмысшылардың санын азайтуы немесе олардың жұмыс түрін өзгертуі мүмкін. Мұндай жағдайда пайдаланушылар ұйымға мұндай жүйені енгізуге белсенді қарсылық көрсетуі ықтимал. Жаңа компьютерлік жүйе талаптарын сақтау мақсатында басқарушылардың жұмыс әдісін өзгертуді көздейтін құрылымдар көбіне қабылданбайды. Басқарушылардың бойында жүйе олардың ұйымдағы алатын орнына қауіп төндіреді деген сезім туындауы мүмкін.
3. *Ұйымдастырушылық өзгерістер.* Жүйелер ұйымдағы саяси биліктің құрылымын өзгертуі мүмкін. Мысалы, егер ұйым күрделі жүйеге тәуелді болса, онда жүйеге қолжетімділікті реттейтін тұлғалар айтарлықтай саяси билікке ие болады.

Әлеуметтік-технологиялық жүйелердің әрқайсысы қорғаныс пен функционалдық сенімділік үшін өте маңызды үш түрлі сипатқа ие:

1. Олар жүйенің тек жеке бөлшектеріне ғана тән емес, тұтас жүйенің қасиеттері болып табылатын қасиеттерге ие. Тәуелсіз қасиеттер жүйе компоненттеріне, сол сияқты олардың өзара байланысына негізделеді. Мұндай кешенділікті ескере отырып, тәуелсіз қасиеттерді жүйе құралған кезде ғана бағалауға болады. Қорғаныс және қызметтік сенімділік жүйенің тәуелсіз қасиеттері болып табылады.
2. Олар көбінесе айқындалмаған болып келеді. Бұл белгілі деректерді енгізген кезде олар әрқашан ұқсас деректерді бере бермейді. Жүйенің әрекеті операторға байланысты, ал адамдар әрқашан бірдей әрекет етпейді. Сонымен қатар, жүйені пайдалану жүйе компоненттері арасында жаңа өзара байланыстардың туындауына алып келуі мүмкін, сондықтан оның әрекеті де өзгереді. Сол себепті жүйенің қателіктері мен ақаулары уақытша сипатта болуы, ал адамдардың қатенің туындағаны немесе туындамағаны туралы пікірлері бір жерден шықпауы мүмкін.

3. Жүйе ұйымның міндеттері мен мақсаттарын ұстап тұратын деңгей тек қана жүйенің өзіне байланысты емес. Ол сонымен қатар, аталған мақсаттардың тұрақтылығына, ұйым мақсаттары арасындағы қайшылықтарға және өзара қатынастарға, ұйымдағы адамдардың аталған мақсаттарды қалай түсіндіретініне байланысты. Ұйым басқармасының жүйе іске асыру тиіс мақсаттарды дұрыс түсіндірмеуі нәтижесінде «табысты» жүйенің өзі «қате» жүйе ретінде қарастырылуы мүмкін.

Әлеуметтік-технологиялық аспектілер жүйенің өз мақсаттарын орындау сәттілігін анықтауда көбінесе сынаушы рөл атқарады. Өкінішке орай, әлеуметтік және мәдени зерттеулер саласында тәжірибесі жоқ әзірлеушілерге оларды ескеру өте қиын.

Қасиет	Сипаты
Көлемі	Жүйенің көлемі (жалпы алатын орны) компонент блоктарының қалай орналасқаны және жалғанғанына байланысты түрленіп отырады.
Сенімділігі	Жүйенің сенімділігі компоненттерінің сенімділігіне байланысты, алайда, күтпеген өзара әрекеттестіктер ақаулардың жаңа түрлерін туғызуы мүмкін, яғни жүйе сенімділігіне кері әсерін тигізеді.
Қорғалуы	Жүйенің қорғалуы (оның шабуылдарға төтеп беру қабілеті) өлшеуге келмейтін кешенді қасиет болып табылады. Жүйе әзірлеушілері ойластырмаған бұзу әрекеттері жүргізілуі мүмкін, мұндай жағдайда орнатылған қорғаныс құралдары дәрменсіз болады.
Жөндеуге жарамдылығы	Бұл қасиет жүйе проблемаларының олар анықталғаннан кейін қалай жеңіл жойылатынын көрсетеді. Ол проблема диагностикасының қабілеттеріне, ақаулы компоненттерге қолжетімділікке және аталған компоненттерді модификациялау немесе ауыстыру мүмкіндіктеріне байланысты.
Практикалығы	Бұл қасиет жүйені пайдалану қалай жеңіл жүргізілетінін көрсетеді. Ол техникалық жүйе компоненттеріне, оның операторларына және пайдалану ортасына байланысты.

10.2-сурет. Тәуелсіз қасиеттер мысалдары

Жүйенің ұйымға әсерін жақсырақ түсіну үшін әр түрлі методологиялар әзірленді. Мысалы алып қарағанда: Мамфорд (1989) әлеуметтік техниктері және Чекландтың бағдарламалық жүйелер методологиясы (1981; Чекланд және Шолс, 1990). Сол сияқты, компьютерлік жүйенің жұмысқа әсерлерін әлеуметтік зерттеулер жүргізілді (Экройд және басқ., 1992; Андерсон және басқ., 1989; Зухман, 1987).

10.1.1 Жүйенің тәуелсіз қасиеттері

Жүйе компоненттері арасындағы кешенді байланыс жүйенің тек бөлшектердің жиынтығы емес екенін білдіреді. Ол жалпы жүйенің қасиеттері болып табылатын қасиеттерге ие. Аталған «тәуелсіз қасиеттер» (Чекланд, 1981) жүйенің қандай да бір белгілі бір бөлігіне жатқызылмайды. Олар тек жүйе компоненттері интеграцияланғанда пайда болады. Бұл қасиеттердің кейбірі (мысалы, салмақ) қосалқы жүйелердің ұқсас қасиеттерінен тікелей туындауы мүмкін. Алайда, олар көбінесе қосалқы жүйелердің кешенді өзара қатынастарына негізделген. Жүйе қасиетін жүйенің жеке компоненттері қасиеттерінен анықтау мүмкін емес. Кейбір тәуелсіз қасиеттер мысалдары *10.2-суретте* көрсетілген.

Тәуелсіз қасиеттердің екі түрі бар:

1. Егер жүйенің мақсаты оның компоненттерінің интеграциясынан кейін туындаса, тәуелсіз қасиеттер функционалдық болып табылады. Мысалы, велосипед өз компоненттері құралғаннан кейін ғана көлік құралы ретінде функционалдық қасиетке ие болады.
2. Функционалдық емес тәуелсіз қасиеттер өзінің операциялық ортасында жүйенің әрекетіне байланысты. Сенімділік, жұмысқа қабілеттілік, қорғаныс және қауіпсіздік функционалдық емес тәуелсіз қасиеттерге мысал бола алады. Аталған қасиеттер компьютерлік жүйелер үшін сыни маңызға ие, себебі бұл қасиеттердің төменгі белгіленген деңгейіне қол жеткізуге қабілетсіздік жүйені қолайсыз етеді. Кейбір пайдаланушылар жүйенің кейбір функцияларына мұқтаж болмауы мүмкін, сондықтан жүйе оларсыз да тиімді бола алады.

10.3-сурет. Қателіктің таралу схемасы

Алайда, сенімсіз және жылдамдығы тым баяу болатын жүйені барлық пайдаланушылар да қабылдамайды.

Функционалдық сенімділіктің қасиеттері (мысалы, сенімділік) жеке компонент қасиеттеріне де, олардың өзара әрекеттестігіне де тәуелсіз. Жүйе компоненттері

өзара тәуелді болып табылады. Бір компоненттің қателігі барлық жүйе бойынша таралуы және басқа компоненттерге де кері әсерін тигізуі мүмкін.

Алайда, көбінесе компоненттің бұл қателіктері басқа компоненттерге қалай әсер ететінін болжау мүмкін емес. Сондықтан жүйе компоненттері сенімділігі туралы мәліметтерге сүйене отырып, жалпы жүйенің сенімділігін бағалау іс жүзінде мүмкін болмайды.

Әлеуметтік-технологиялық себептерді талдау кезінде сенімділікті үш түрлі көзқарас тұрғысынан қарастырған дұрыс:

1. *Ақпараттық қамтамасыз етудің сенімділігі.* Ақпараттық компоненттердің істен шығу ықтималдығы қандай? Бұл компоненттерді жөндеу қанша уақытты алады?
2. *Бағдарламалық қамтамасыз етудің сенімділігі.* Бағдарламалық компоненттің қате қорытынды беру ықтималдығы қаншалықты үлкен? Бағдарламалық қамтамасыз етудің қателігі ақпараттық қателіктерден БҚ ескірмейтіндігімен ерекшеленеді. Қателіктер көбіне уақытша сипатта болады. Жүйе қате нәтиже алынғаннан кейін де жұмысын жалғастыра береді.
3. *Оператордың сенімділігі.* Жүйе операторының қателік жіберіп, дұрыс емес деректерді енгізу мүмкіндігі қандай? Аталған қателікті бағдарламалық қамтамасыз етудің анықтай алмай, барлық жүйе бойынша тарату ықтималдығы қандай?

Ақпараттық қамтамасыз ету, бағдарламалық қамтамасыз ету сенімділігі тәуелсіз болып табылмайды. *10.3-суретте* бір деңгейде туындаған қателіктер жүйенің басқа барлық деңгейлеріне таралу мүмкіндігі көрсетілген. Ақпараттық қамтамасыз етудегі ақау жалған сигнал беруі мүмкін, олар бағдарламалық қамтамасыз ету күтетін енгізулер санына енгізілмейді. Нәтижесінде, бағдарламалық қамтамасыз етудің әрекетін болжау қиын, ол болжанбайтын деректер беруі мүмкін. Аяғында бұл операторды шатастыруы ықтимал.

Оператордың қателігі ықтималдығы ол үшін стресті болып саналатын оқиғалар да арта түседі. Осылайша, ақпараттық қамтамасыз етудегі ақау жүйе операторының қателік жіберуіне, ол өз кезегінде бағдарламалық қамтамасыз етудің жаңа проблемалары туындауына немесе деректерді қосымша өңдеу қажеттілігіне алып келуі мүмкін. Бұл ақпараттық қамтамасыз етудің жүктемесі ұлғаюына және оның істен шығуына әсер етеді. Осылайша, қалпына келтіру оңай болған бастапқы қателік жүйенің толықтай ажырауына алып келуі мүмкін елеулі проблемаға айналады.

Жүйенің сенімділігі жүйені пайдалану контекстіне байланысты. Алайда, жүйе ортасын толық анықтау мүмкін емес, ал әзірлеушілер операциялық жүйе ортасына ешқандай шектеулер қоя алмайды. Орта шеңберінде жұмыс істейтін әртүрлі жүйелер проблемаларға болжанбайтын түрде әрекет етуі мүмкін, ал бұл барлық жүйелердің сенімділігіне әсер етеді.

Мысалы, жүйе бөлме температурасында жұмыс істеуге арналып шығарылды делік. Жүйенің электрондық компоненттері температураның белгілі диапазоны шеңберінде жұмыс істеуге жобаланған (мысалы, 0-ден 45 градусқа дейін).

Берілген температура диапазонынан тыс жерде компоненттердің әрекетін болжау мүмкін емес.

Мысалы, аталған жүйе қолдану ортасында ауа кондиционерімен қатар орнатылды делік. Кондиционер істен шыққан жағдайда ол электрондық компоненттерді қыздыра бастауы мүмкін, нәтижесінде жүйенің қызып кетуі орын алады. Мұндай жағдайда компоненттер және барлық жүйе істен шығады. Егер о баста жүйе басқа жерде орналасқан болса, мұндай проблема туындамас еді. Сол сияқты, ауа кондиционері дұрыс жұмыс істесе де ешқандай проблема болмас еді. Алайда, аталған машиналардың физикалық жақындығынан олардың арасында болжанбаған өзара байланыс туындап, жүйенің істен шығуына әсер етті.

Сенімділік сияқты, жұмысқа қабілеттілік және қолайлылық қасиеттерін бағалау қиын, бірақ жүйені пайдалану басталғаннан кейін өлшеуге болады. Алайда, қорғаныс және қауіпсіздік сияқты қасиеттерді өлшеу мүмкін емес. Берілген жағдайда біз тек жүйенің әрекеті белгілері ғана емес, тиімсіз және қаламайтын әрекеттермен кездесіп отырмыз. Жүйе егер ол өз деректеріне рұқсат етілмеген енулерге жол бермесе қорғалған болып саналады. Алайда, рұқсат алудың барлық мүмкін нұсқаларын анықтау, тиісінше олардың алдын алу мүмкін емес. Сондықтан, әзірлеуші мұндай «тиіс емес» үлгідегі қасиеттерді стандартты түрде ғана бағалайды. Басқаша айтқанда, Сіз жүйенің оған біреудің енуі орын алғанға дейін ғана қауіпсіз болып саналатынын түсінесіз.

10.1.2 Жүйенің детерминистік еместігі

Егер жүйе толық болжанатын болса ол детерминистік болып табылады. Егер уақыт аспектілерін ескермесек, сенімді ақпараттық қамтамасыз етуде жұмыс жасайтын бағдарламалық жүйелер енгізудің белгілі тізбектілігін ала отырып, шығарулардың әрқашан дәл сондай тізбектілігін құратын болады. Әрине, абсолюттік сенімді ақпараттық қамтамасыз ету болмайды, алайда, ақпараттық жүйені детерминистік деп тану үшін әдетте ақпараттық қамтамасыз ету жеткілікті түрде сенімді деп табылады.

Бір жағынан алғанда, адамдар детерминистік болып табылмайды. Абсолютті түрде бірдей ақпаратты қабылдағанда (мысалы, тапсырманы орындауға сұраныс) реакция сұранысты жасаған адамның, ортаның басқа адамдарының эмоциялық және физикалық жағдайына, сонымен қатар, берілген уақытта олардың немен айналысып жатқанына байланысты болады. Кейде адамдар жұмысқа қуанышпен кіріседі, кейде біріге жұмыс істеуден бас тартады.

Әлеуметтік-технологиялық жүйелер оларға адамдар кіретіндіктен, сонымен қатар, ақпараттық қамтамасыз етуге, бағдарламалық қамтамасыз етуге және осы жүйелердің деректеріне өзгерістер енгізу жиілігінің жоғарылығына байланысты ішінара детерминистік емес болып табылады. Аталған өзгерістер арасындағы өзара байланыстар күрделі болып келеді, сондықтан, жүйенің әрекетін болжау мүмкін болмайды. Бұл фактінің өзі проблема болып табылмайды, алайда, функциялық сенімділік тұрғысынан қарағанда жүйе ақауларының туындауын анықтауды,

жүйелік ақаулардың туындау жиілігін бағалауды қиындатады. Мысалы, жүйеге 20 сынау енгізулерінен тұратын жинақ ұсынды делік. Ол берілген енгізулерді өңдейді және нәтижелерді жазып отырады. Біраз уақыттан кейін сол 20 сынау енгізулері өңделеді, ал нәтижелері алдыңғылармен салыстырылады. Нәтижесінде бес сәйкессіздік табылды. Бұл жүйенің бес ақауы орын алды дегенді білдіреді ме? Немесе аталған сәйкессіздіктер жүйе әрекетінің стандарттық өзгерісінің нәтижесі болып табыла ма? Сіз мұны тек аталған нәтижелерді терең талдау жолымен және әрбір жеке енгізуді өңдеу әдісімен анықтай аласыз.

10.1.3 Табыстылық өлшемдері

Жалпы және тұтас алғанда, күрделі әлеуметтік-технологиялық жүйелер «қаскөй» проблемаларды жою үшін әзірленген (Риттел және Уэббер, 1973). «Қаскөй» проблема – бұл кешенді болып табылатын, құрамына бірнеше аймақтар кіретін проблема, оның толық сипатын беру мүмкін емес. Түрлі мүдделі қатысушылар проблеманы әр түрлі көреді, алайда олардың ешқайсысы проблеманы толық түсінбейді. Проблеманың шынайы келбеті тек шешімін тапқаннан кейін анықталуы мүмкін. «Қаскөй» проблемаға көрнекті мысал – жер сілкінісін болжау. Ешкім жер сілкінісінің эпицентрі қай жерде болатынын және қашан басталатынын, оның қоршаған ортаға қандай әсер тигізетінін дәл айта алмайды. Сол сияқты, ірі жер сілкінісімен қалай күресуге болатынын да сипаттау мүмкін емес.

Бұл жүйе табыстығының өлшемін анықтауды қиындатады. Сіз қалай анықтайсыз, жүйе оны алу үшін ақша төлеген компанияның іскерлік мақсаттарын және міндеттерін жоспарға сәйкес жүзеге асыра ала ма? Табыстылықты анықтау әдетте жүйені әзірлеу мен сатып алудың бастапқы себептеріне негізделмейді. Ол қолдану сәтінде жүйенің қаншалықты әсерлі екеніне негізделеді. Себебі іскерлік орта тез өзгеруге бейім, іскерлік мақсаттар мен міндеттер жүйені әзірлеу сатысының өзінде елеулі өзгерістерге ұшырауы мүмкін.

Бір-біріне қайшы келетін мүдделі қатысушылардың әрқайсысы әр түрлі түсіндіретін мақсаттар көп болғанда жағдай да күрделене түседі. Мысалы, клиниканың деректер базасы жүйесі (мысал *1-тарауда* толық сипатталған) екі белгілі мақсатты жүзеге асыру үшін әзірленген:

1. Психикалық аурумен ауыратын сырқаттарды емдеу және күту сапасын жақсарту.
2. Сырқаттарды емдеу және күту, емдеудің құны туралы толық есептер ұсыну арқылы кірісті ұлғайту.

Өкінішке орай, аталған мақсаттар бір-біріне қайшы келді, себебі толық есептер құру үшін дәрігерлер мен медбикелерге әдетте стандарттық медициналық карталарға енгізілмейтін қосымша ақпараттарды ұсынуға тура келді. Бұл емделушілерді күту сапасының төмендеуіне алып келді, себебі клиника қызметкерлерінің олармен сөйлесуге уақыттары әлдеқайда аз қалды.

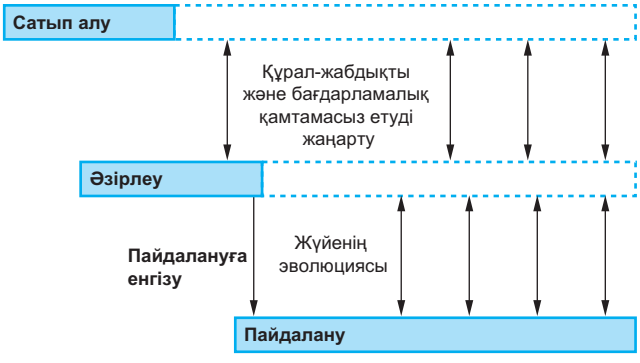
Дәрігерлердің көзқарасы бойынша аталған жүйе деректерді қолмен тіркеген алдыңғы жүйені ешбір жақсартпаған; басқарушы қызметкерлердің пікірінше жүйе өз табыстылығын дәлелдеген. Қорғалу және функционалдық сенімділік сипаттарының мәні жүйе табыстылығы өлшемдерін анықтауды одан әрі қиындата түседі. Жаңа жүйені енгізу мақсаты қолданыстағы жүйені деректер ортасы неғұрлым жақсы қорғалған басқа жүйеге ауыстыру жолымен қауіпсіздікті жақсарту болуы мүмкін. Мысалы, жүйені орнатқаннан кейін оған шабуыл жасалды делік, нәтижесінде қорғаныста тесік анықталып, деректер зақымдалды. Бұл жүйеде қателіктер бар дегенді білдіреді ме? Біз нақты жауап бере алмаймыз, себебі біз бұрынғы ескі жүйеге дәл осындай шабуыл жасалса, оған келетін шығындардың көлемін білмейміз.

10.2 Жүйелерді жобалау

Жобаны жүйелеу әлеуметтік-техникалық жүйелерді сатып алу, анықтау, құрылымдау, іске асыру, тексеру, пайдалану және қызмет көрсету бойынша барлық операцияларды қамтиды. Жүйені әзірлеушілер тек қана бағдарламалық қамтамасыз етуге емес, ақпараттық қамтамасыз етуге және жүйенің өз ортасымен және пайдаланушылармен өзара әрекеттестігіне назар аударады. Олар жүйе ұсынатын қызметтерді, жүйені әзірлеуге және қызметіне салынатын шектеулерді, сонымен қатар, өз мақсатын іске асыру үшін жүйені пайдалану әдістерін ойластыруы қажет.

Ірі және күрделі әлеуметтік-технологиялық жүйелердің қызмет ету мерзімін ішінара үндесетін үш сатыға бөлуге болады (*10.4-сурет*):

1. *Сатып алу немесе алу.* Бұл сатыда жүйенің мақсаты анықталады; жоғары деңгейдегі жүйеге талаптар белгіленеді; ақпараттық құрылғылар, бағдарламалық қамтамасыз ету және адамдар арасында функционалдық қалай бөлінетіні туралы шешімдер қабылданады; жүйе құрылатын компоненттер алынады.
2. *Әзірлеу.* Бұл сатыда жүйені әзірлеу жүзеге асырылады. Әзірлеу үдерістерінің құрамына жүйені әзірлеу бойынша барлық операциялар кіреді: талаптардың сиректеуі, жүйе құрылымын құру, бағдарламалық және ақпараттық қамтамасыз етуді жобалау, жүйені интеграциялау және тестілеу. Операциялық үдерістер анықталады, пайдаланушылар үшін жүйемен жұмысқа үйрету курстары құрылады.
3. *Пайдалану.* Аталған сатыда жүйе пайдалануға беріледі, пайдаланушылар қажетті оқытудан өткізіледі. Әдетте, жоспарланған операциялық үдерістер жүйе қолданылатын шынайы жұмыс ортасы ерекшеліктерін көрсету мақсатында түзетіледі. Жүйе жаңа талаптардың жылжуына қарай дамып отырады. Уақыт өткеннен кейін жүйенің құндылығы төмендейді, ол пайдаланудан шығарылады және ауыстырылады.



10.4-сурет. Жүйені жобалау сатылары

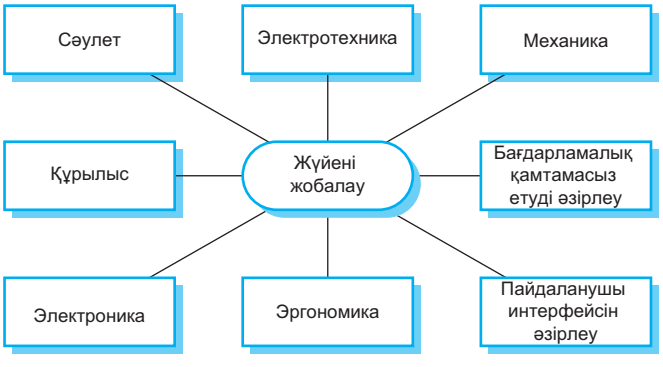
Аталған сатылар тәуелсіз болып табылмайды. Жүйені пайдалануға енгізгеннен кейін қызметін арттыру немесе жоғарғы сұранысқа сәйкес болу үшін жүйенің ескіретін компоненттерін ауыстырып, үнемі жаңа құрал-жабдықтар мен бағдарламалық қамтамасыз ету сатып алу қажет. Осы сияқты, өзгерістер енгізуге сұраныстар жүйені одан әрі дамытуды талап етеді.

Жүйенің жалпы қорғанысы және функционалдық сенімділігі барлық үш сатының операцияларына негізделеді. Құрылым нұсқалары жүйе масштабы және оның ақпараттық, бағдарламалық қамтамасыз етілуіне қатысты сатып алу сатысы шешімдерімен шектелуі мүмкін. Кейбір қорғаныс құралдарын іске асыру мүмкін болмауы ықтимал. Бұл әдетте жүйенің болашақтағы ақауларына алып келуі мүмкін белгілі кемшіліктерге байланысты орын алады. Адамның спецификациялау, құрылымдау және әзірлеу кезіндегі жол берген қателіктері жүйеге қателіктер енгізілгенін білдіреді. Сәйкес емес тестілеу аталған кемшіліктердің жүйе пайдалануға енгізілгенге дейін анықталмауының себебі болуы мүмкін. Пайдалану үдерісінде жүйе конфигурациясы қателіктері оның одан әрі жұмысындағы жаңа кемшіліктерге себеп болуы ықтимал. Жүйе операторлары жүйені пайдалану уақытында қателіктер жіберуі мүмкін. Сатып алу сатысында жол берілген қателіктер жүйеге өзгерістер енгізу кезінде ұмытылып қалуы мүмкін, бұл да жоғарыда айтылғандай, жүйенің бірқатар жаңа кемшіліктеріне себеп болады.

Жүйені жобалау мен бағдарламалық қамтамасыз ету арасындағы негізгі айырмашылықтардың бірі жүйе қызметінің барлық мерзімі ішінде әр түрлі саладан бірқатар сарапшыларды тарту болып табылады. 10.5-суретте әуе қозғалысы ұйымының жаңа жүйесін әзірлеу және сатып алу сатыларында қолданылуы мүмкін технологиялық тәртіп көрсетілген. Сонымен қатар, сәулетші мамандар және инженер-құрылысшылар тартылады, себебі әуе қозғалысы ұйымының жаңа жүйесі әдетте жаңа ғимаратта орнатылады. Электр мамандары мен механиктер тамақтану мен ауаны кондициялау талаптары мен қызмет көрсетуді белгілеуге тартылады. Электроника бойынша инженерлер компьютерлермен, радарлармен, басқа да құрал-жабдықтармен жұмыстарды жолға қою үшін тартылады. Эргономика бойынша мамандар бақылаушылар үшін жұмыс орындарын жобалайды, ал

бағдарламалық қамтамасыз етуді және пайдаланушы интерфейсін әзірлеушілерге жүйені бағдарламалық қамтамасыз ету жүктеледі. Әр түрлі саладан мамандарды тарту күрделі әлеуметтік-технологиялық жүйелер көптеген түрлі аспектілерге ие болғандықтан маңызды. Алайда, тәртіптер арасындағы айырмашылықтар жүйенің кемшіліктеріне себеп болуы, әзірленетін жүйенің функционалдық сенімділігін және қорғалуын елеулі түрде нашарлатуы мүмкін:

1. Әр түрлі тәртіптер түрлі заттарды атау үшін бірдей сөздерді пайдаланады. Түрлі салаларда қызмет атқарушы инженерлердің келіссөздері барысында түсініспеушіліктер жиі орын алады. Егер мұндай сәттер жүйені әзірлеу уақытында анықталып жойылмайтын болса, онда олар жүйеге қателік енгізуге алып келуі мүмкін. Мысалы, электроника бойынша инженерлер бағдарламалау негіздерін C# тілінде білуі мүмкін, алайда Java бағдарламалау әдісін функциясы бойынша C тілінде бағдарламалаумен салыстыруға болатынын білмеуі мүмкін.



10.5-сурет. Жүйені жобалау үдерісінде іске қосылған тәртіптер

2. Әрбір тәртіп басқа тәртіптің нені жасай алатыны немесе жасай алмайтыны туралы өзінің жорамалын жасайды. Көбінесе аталған жорамалдар мүмкін болып табылады деген сәйкес емес түсінікке негізделген. Мысалы, пайдаланушы интерфейсін әзірлеуші графикалық пайдаланушы интерфейсін ұсынуы мүмкін, бірақ ол үлкен көлемдегі деректерді өңдеуді талап етеді, бұл соның салдарынан жүйе процессорына артық жүктеме түсіреді дегенді білдіреді.
3. Тәртіптер өздерінің кәсіби шекараларын қорғауға тырысады, бұл белгілі шешімдерге қатысты дауларға алып келеді, себебі, аталған шешімдер олардың кәсіби тәжірибесі болуын талап етеді. Осылайша, бағдарламалық қамтамасыз етуді әзірлеуші ғимаратта бағдарламалық қамтамасыз етуге негізделген есіктерді бұғаттау жүйесін орнату қажеттігі туралы дауласуы мүмкін, бірақ механикалық жүйедегі немесе кілтпен жабылатын есіктер іс жүзінде сенімдірек болуы да ықтимал.

10.3 Жүйені сатып алу

Жүйені жобалаудың бастапқы сатысы жүйені сатып алу болып табылады (кейде жүйені сатып алу деп аталады). Аталған сатыда сатып алынатын жүйенің масштабы, бюджет және жүйені жобалаудың уақытша шеңбері, сонымен қатар жоғары деңгейдегі жүйеге талаптар туралы шешімдер қабылданады. Осындай ақпараттардың негізінде жүйені сатып алу қажеттігі, талап етілетін жүйе түрі, жабдықтаушы немесе жүйенің жабдықтаушысы туралы одан әрі шешімдер қабылданады.

Аталған шешімдерді қабылдауға түрткі болатындар:

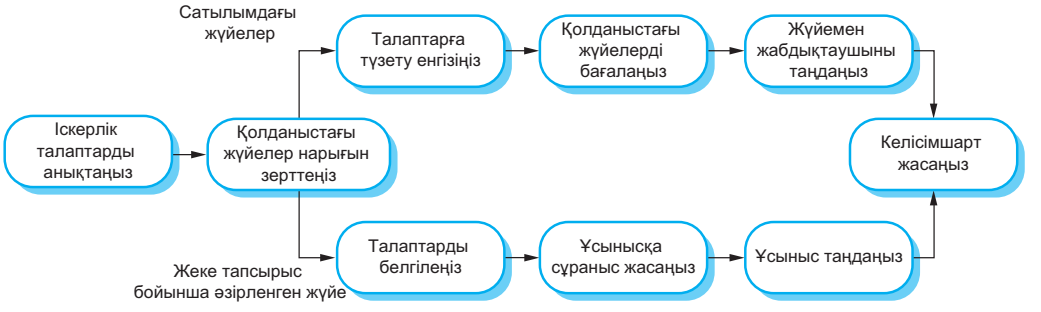
1. *Басқа ұйымдастырушылық жүйелердің жай-күйі.* Егер ұйым бірнеше аралас жүйелерге ие болса және олардың бірлескен жұмысы қиын немесе тым қымбатқа түссе, оларды ауыстыра алатын жүйені сатып алу бірқатар елеулі іскерлік артықшылықтар беруі мүмкін.
2. *Сыртқы қағидаларды сақтау қажеттілігі.* Соңғы уақытта іскерлік кәсіпорындар жиі-жиі мемлекеттің бақылауына түсіп, сыртқы нормалар мен қағидаларды сақтау фактілерін дәлелдеуге міндетті (мысалы, АҚШ-та Сарбейнс-Оксли бухгалтериясын жүргізу заңы). Бұл сәйкес келмейтін жүйелерді ауыстыру немесе нормаларды сақтау мониторингі үшін арнайы жаңа жүйе сатып алу қажеттігіне себеп болады.
3. *Сыртқы бәсекелестік.* Егер кәсіпорынға бәсекелестікті көтеру немесе белгілі деңгейде ұстап тұру қажет болса, іскерлік үдерістердің нәтижелілігін арттыруға қабілетті жаңа жүйелер сатып алуға қаржы құю оңтайлы шешім болуы мүмкін.

Әскери жүйелер туралы сөз қозғасақ, төнген жаңа қауіп алдында әскери қуатты арттыру қажеттілігі жаңа жүйелерді сатып алуға маңызды себеп болып табылады.

4. *Бизнесті қайта құру.* Коммерциялық кәсіпорындар және басқа да ұйымдар нәтижелілігін көтеру және/немесе клиенттерге қызмет көрсету деңгейін арттыру мақсатында өз құрылымын жиі жаңартады. Қайта құру жаңа жүйені қолдауды талап ететін іскерлік үдерістегі өзгерістерге алып келеді.
5. *Қолжетімді бюджет қаражаты.* Қолжетімді бюджет қаражаты – бұл жаңадан сатып алынатын жүйелердің масштабын анықтайтын айқын фактор.

Сонымен қатар, жаңа үкіметтік жүйелер көбінесе мемлекеттік саясатқа және саяси өзгерістерге сәйкес болу үшін сатылып алынады. Мысалы, саясаткерлер өз пікірлерінше терроризмге қарсы күрес жүргізуде тиімді құрал болып табылатын жаңа бақылау жүйесін сатып алу қажет деп шешулері мүмкін. Аталған жүйені сатып алу электоратқа мемлекеттің қажетті шараларды қабылдап жатқанын

көрсетеді. Алайда, мұндай жүйелер көп жағдайда шығындардың тиімділігі талданбастан, яғни сатып алудың түрлі басымдықтары салыстырылмастан сатып алынады.



10.6-сурет. Жүйені сатып алу үдерістері

Ірі, күрделі жүйелер әдетте сатылымдағы жүйелердің және арнайы әзірленген компоненттердің қоспасы түрінде болады. Бағдарламалық қамтамасыз етудің жүйелер үшін жиі қолданыла бастауының бір себебі олардың қолданыстағы ақпараттық компоненттерді пайдалануға мүмкіндік беретіні болып табылады. Бұл жерде бағдарламалық қамтамасыз ету ақпараттық компоненттердің бірге жұмыс істеуін қамтамасыз ететін «желім» рөлін атқарады. Мұндай «желімдеуші» бағдарламалық қамтамасыз етуді әзірлеу қажеттігі сатылымдағы компоненттер қаншалықты үнемді болса да керекті деңгейде әсерлі бола бермейтіндігіне байланысты. *10.6-суретте* сатылымда қолжетімді және жеке әзірлеуді талап ететін компоненттерді сатып алудың қарапайым моделі көрсетілген. Үдерістің диаграммада көрсетілген келесі сәттеріне назар аудару маңызды:

1. Егер талаптар аталған компоненттердің сипаттамасы негізінде анықталмаса, сатылымдағы компоненттер талаптарға әрқашан жауап бере бермейді. Осылайша, жүйені таңдау кезінде талаптарға барынша жауап беретін, сатылымда бар нұсқаларды таңдаған дұрыс. Сізге кейін талаптарды модификациялауға тура келуі мүмкін. Бұл басқа барлық қосақы жүйелерге ықпалын тигізуі мүмкін.
2. Егер жүйе жеке тапсырыс бойынша әзірленсе, талаптар мен ерекшеліктер сатып алынатын жүйеге қатысты келісімшарттың бөлшегі болуы тиіс. Осылайша, олар заңды және техникалық құжат түрінде болады.
3. Жүйені құруға мердігер таңдап алынғаннан кейін, Сіз жүйеге өзгеріс енгізу құны сияқты сәттерді талқылауға және талаптардың кезекті өзгерісін белгілеуге мүмкіндік беретін келісімшартты талқылау кезеңі белгіленуі тиіс. Осындай түрде, сатылымда қолжетімді жүйені таңдағаннан кейін Сіз өнім берушімен құн, лицензия шарттары, жүйенің мүмкін болатын өзгерістері және т.б. туралы талқылай аласыз.

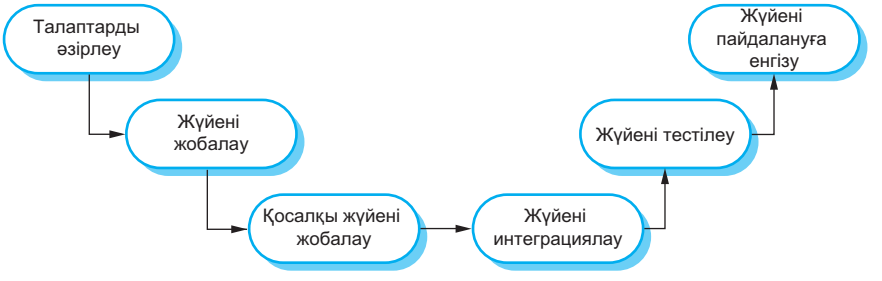
Әлеуметтік-технологиялық жүйелерді бағдарламалық және ақпараттық қамтамасыз етуді әдетте жалпы әлеуметтік-технологиялық жүйені сатып алатын ұйымнан бөлек өзге ұйым (өнім беруші) әзірлейді. Бұған клиенттің қызмет аясы әрдайым бағдарламалық қамтамасыз етуді әзірлеу болып табылмайтыны себеп болады, сондықтан оның қызметкерлері жүйені өз бетімен әзірлеуге қажетті білім мен машықтарға ие емес. Компаниялардың тым аз бөлігі ірі әлеуметтік-технологиялық жүйенің барлық компоненттерін жобалау, өндіру және тестілеу мүмкіндіктеріне ие. Сол себепті, әдетте негізгі мердігер деп аталатын өнім беруші көп жағдайда әр түрлі қосалқы жүйелерді әзірлеу бойынша бірнеше субмердігерлермен келісімшарттар жасайды. Ірі жүйелер жағдайында (мысалы, әуе қозғалысын басқару жүйесі) бірнеше өнім берушілер келісімшарт жасау мақсатында консорциум құруы мүмкін. Консорциум аталған жүйе үлгісіне қажетті барлық құралдарға ие болуы тиіс. Бұл компьютерлік ақпараттық қондырғылар, бағдарламалық қамтамасыз етуді әзірлеушілер, перифериялық өнім берушілер, сонымен қатар, мамандандырылған құрал-жабдықтар өнім берушілері (мысалы, радиолокациялық жүйемен жабдықтаушылар) болуын көздейді. Клиент әдетте субмердігерлермен емес негізгі мердігермен жұмыс жасайды. Субмердігерлер негізгі мердігер ұсынған ерекшелікке сәйкес жүйенің бөлшектерін әзірлейді және өндіреді. Олардың жұмыстары аяқталғаннан кейін негізгі мердігер барлық жеке компоненттерді біріктіріп клиентке ұсынады. Келісімшарт талаптарына қарай клиент мердігерге сумердігерлерді таңдау бостандығын береді немесе мақұлданған тізімнен таңдап алуды талап етеді. Сатып алу сатысында қабылданған шешімдер мен таңдаулар жүйенің қорғанысына және функционалдық сенімділігіне әсер етеді. Мысалы, егер сатылымда қолжетімді жүйені сатып алу туралы шешім қабылданса, ұйым оның жүйенің қорғанысы және функционалдық сенімділігі бойынша талаптарға шектеулі түрде ғана әсер ететінін қабылдауы тиіс. Бұл қасиеттер көбінесе жүйенің өнім берушілері қабылдаған шешімдерге байланысты. Сонымен қатар, сатылымда қолжетімді жүйелер көпшілікке белгілі кемшіліктері болуы немесе кешенді конфигурацияны талап етуі мүмкін. Конфигурация қателіктері егер ену нүктесі тиісті қорғалмаса, қауіпсіздік проблемасының негізгі көзі болып саналады.

Екінші жағынан, жеке тапсырыс бойынша әзірленген жүйені сатып алу шешімі қорғаныс пен функционалдық сенімділікке талаптарды анықтау мен түсінуге үлкен күш жұмсау қажеттілігін білдіреді. Егер компанияның тәжірибесі аталған салада шектеулі болса, мұны орындау жеңіл болмайды. Егер жүйенің жұмысқа қабілеттілігі және функционалдық сенімділігінің белгілі деңгейіне қол жеткізу қажет болса, әзірлеуге қосымша уақыт қажет болып, жоспарланған бюджет ұлғаюы мүмкін.

10.4 Жүйені әзірлеу

Жүйені әзірлеу үдерісінің міндеті жүйенің барлық компоненттерін сатып алу немесе әзірлеу, одан кейін түпкілікті жүйе құру үшін аталған компоненттерді

интеграциялау болып табылады. Талаптар сатып алу және әзірлеу үдерістері арасындағы көпір рөлін атқарады. Функционалдық және функционалдық емес талаптар сатып алу сатысында анықталады. Сіз мұны аталған үдерістер ортақ сәттерге ие болғандықтан, әзірлеуді басы ретінде қабылдауыңыз мүмкін (*10.4-суретті* қара). Компоненттерге қатысты келісімшарт келісімделгеннен кейін талаптарды толығырақ әзірлеу басталады. *10.7-суретте* жүйені әзірлеу үдерісінің моделі көрсетілген. Жүйені жобалаудың аталған үдерісі *2-тарауда* сипатталған бағдарламалық қамтамасыз етуді әзірлеудің каскадтық моделіне елеулі әсер етеді.



10.7-сурет. Жүйені әзірлеу

Қазіргі уақытта каскадтық модель бағдарламадық қамтамасыз етуді әзірлеуге жарамайды деп айтылса да, жүйені әзірлеу үдерістерінің басым бөлігі аталған модельді ұстанатын жоспарға негізделген. Жоспар негізіндегі үдерістер жүйенің түрлі бөліктері бір мезгілде әзірленетіндіктен жүйені жобалау кезінде қолданылады. Құрамына ақпараттық қамтамасыз ету және басқа да құрал-жабдықтар кіретін жүйелер үшін әзірлеу сатысында өзгерістерді енгізу өте қымбат, кейде іс жүзінде мүмкін емес болады. Сондықтан ақпараттық қамтамасыз етуді әзірлеуге немесе құрылыс жұмыстарына кіріспес бұрын жүйенің барлық талаптарын түсіну өте маңызды. Ақпараттық қамтамасыз ету проблемаларын жою мақсатында жүйе құрылымын қосымша өңдеу сирек жағдайларда мүмкін болады. Осыған байланысты функциялардың басым көпшілігі жүйенің бағдарламалық қамтамасыз етілуіне жүктеледі. Бұл жүйені әзірлеудің үдерісіне жаңа талаптарға жауап ретінде бірқатар өзгерістер енгізуге мүмкіндік береді.

Жүйені жобалаудың неғұрлым көп шатасқан аспектілерінің бірі – компанияның үдерістің әрбір сатысында түрлі терминологияны пайдалануы. Үдеріс құрылымы да түрленіп отыруы мүмкін. Кейде талаптарды анықтау әзірлеу үдерісінің бөлшегі болып табылады, ал кейде жеке операция түрінде болады. Алайда, жүйені әзірлеу алты іргетастық операциядан тұрады:

1. *Талаптарды әзірлеу.* Іскерлік талаптар және сатып алу үдерісінде анықталған жоғарғы деңгейдегі талаптар неғұрлым мұқият әзірлеуді қажет етеді. Талаптар ақпараттық қамтамасыз ету, бағдарламалық қамтамасыз ету немесе үдерістерге арналуы және іске асырудың басымдығына ие болуы мүмкін.

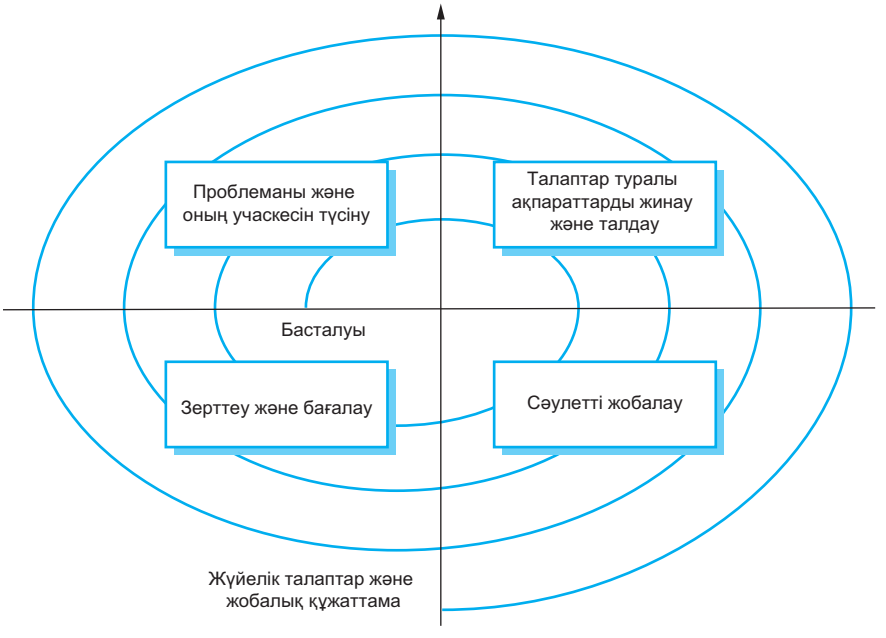
2. *Жүйені жобалау.* Аталған үдеріс көп жағдайларда талаптарды әзірлеу үдерісімен үндеседі. Ол жүйенің жалпы сәулетін, жекелеген компоненттердің идентификациясын және олардың өзара байланысын анықтауды көздейді.
3. *Қосалқы жүйелерді жобалау.* Аталған саты жүйенің бағдарламалық компоненттерін әзірлеуді; сатылымда қолжетімді бағдарламалық және ақпараттық қамтамасыз етудің, қажет болған жағдайда мамандандырылған ақпараттық қамтамасыз етудің конфигурациясын; жүйенің операциялық үдерістерін анықтауды; негізгі іскерлік операцияларды қайта құруды көздейді.
4. *Жүйенің интеграциясы.* Аталған сатыда компоненттер жаңа жүйе құру мақсатында біріктіріледі. Тек интеграциядан кейін ғана жүйенің тәуелсіз қасиеттері айқын көрінеді.
5. *Жүйені тестілеу.* Тестілеу әдетте үдерісі барысында проблемалар анықталатын көлемді және ұзақ операция болып табылады. Аталған проблемаларды жою, жүйе жұмысын реттеу, жаңа талаптарды іске асыру мақсатында қосалқы жүйелерді жобалау және жүйені интеграциялау фазалары екінші рет жүргізіледі. Жүйені тестілеу жүйені әзірлеуші тарапынан немесе осы жүйені сатып алған ұйым атынан тестілеу ретінде көрсетілуі мүмкін.
6. *Жүйені пайдалануға енгізу.* Аталған үдеріс барысында жүйе өз пайдаланушылары үшін қолжетімді бола түседі, ескі жүйеден деректерді ауыстыру, басқа орта жүйелерімен байланыс орнату жүзеге асырылады. Үдерістің шарықтау шегі белсендіру болып табылады, осыдан кейін пайдаланушылар жүйені өздерінің күнделікті жұмыстарында пайдалана бастайды.

Үдеріс тұтас алғанда жоспарға сүйенсе де, талаптарды және жүйені жобалау жұмыстары өзара тығыз байланысты. Талаптар мен жоғарғы деңгейдегі құрылым бір мезгілде әзірленеді. Ескі жүйелер қоятын шектеулер құрылымның кейбір нұсқаларын алып тастауы мүмкін, ал аталған нұсқалар талаптарда көрсетілуі мүмкін. Сізге құрылымдау мен талаптарды әзірлеу үдерісін ұйымдастыру үшін бастапқы жобалауды жүргізуге тура келуі мүмкін. Жобалау барысында Сіз қолдағы талаптар мен проблемаларды анықтауыңыз мүмкін, нәтижесінде жаңа талаптар туындайды. Сіз аталған үдерістер арасындағы байланысты спираль түрінде көруіңізге болады (*10.8-сурет*).

Спираль жобалау шешіміне талаптардың ықпал ету және керісінше фактісін көрсетеді, сондықтан аталған үдерістерді кезекті етудің мәні бар. Ортадан бастап спиральдың әрбір орамы жобалауға және талаптарға қатысты жаңа ақпаратты қоса алады. Бір орамдар талаптарға, ал басқалары жобалауға бағытталады. Кейде талаптар мен жобалауды әзірлеу барысында алынған жаңа мәліметтер проблеманың өзінің тұжырымдалуын өзгерту қажеттігін білдіреді.

Іс жүзінде барлық жүйелер үшін құрылымдардың талаптарды қанағаттандыратын көптеген нұсқалары бар. Бұл ақпараттық қамтамасыз етуді, бағдарламалық қамтамасыз етуді және адамдар орындайтын операцияларды

қиыстыратын бірқатар шешімдерді жобалайды. Сіз одан әрі әзірлеуге таңдайтын шешім барлық талаптарға сәйкес келетін тиімді техникалық шешім болуы мүмкін. Алайда, көлемді ұйымдастырушылық және саяси талдау шешім таңдауға әсер ете алады. Мысалы, үкіметтік клиент шетелдік өнім берушілердің орнына шығарған өнімінің сапасы төмен болса да, ұлттық өнім берушілерді таңдауы мүмкін. Аталған факторлар спиральдық модельді зерттеу және бағалау сатысында айқын көрінеді.



10.8-сурет. Талаптар мен жобалау спиралі

Үдеріс қосалқы жүйелерді әзірлеу және анықтауға арналған талаптар және жоғарғы деңгей жобасы туралы жеткілікті мөлшерде ақпарат алынды деп шешілгенде аяқталады.

Қосалқы жүйелерді жобалау фазасында жүйенің ақпараттық және бағдарламалық компоненттері іске асырылады. Жүйелердің кейбір түрлері үшін (мысалы, ғарыштық аппараттар жүйесі) барлық бағдарламалық және ақпараттық компоненттер әзірлеу үдерісі кезінде құрылуы және жобалануы мүмкін. Алайда, көптеген жүйелердің кейбір компоненттері нарықта бар жүйелер болып табылады. Әдетте бар өнімдерді сатып алу құны мамандандырылған компоненттерді әзірлеу құнынан әлдеқайда төмен.

Қосалқы жүйелерді әзірлеу әдетте параллель түрде жүргізіледі. Қосалқы жүйенің шегінен тыс проблемалармен кездескен кезде жүйенің модификациясы сұранысын жүргізу қажет. Егер жүйелер ақпараттық құрылғылардың көлемді базасы түрінде болса, өндіріс басталғаннан кейін модификацияны енгізу өте қымбат болып табылады. Әдетте мұндай жағдайда проблеманың «айналма жолдары» іздестіріледі. «Айналма жолдар» әдетте бағдарламалық қамтамасыз етуді өзгертуді

көздейді, бұған бағдарламалық қамтамасыз етуге тиесілі иілгіштік қасиеті себеп болады.

Жүйелерді интеграциялау сатысында Сіз бір-біріне тәуелсіз әзірленген қосалқы жүйелерді түпкілікті жүйе құрау үшін біріктіресіз. Интеграцияны барлық қосалқы жүйелер бір мезгілде интеграцияланатын «үлкен жарылыс» тәсілі көмегімен жүзеге асыруға болады. Алайда, техникалық және басқарушылық себептер бойынша инкременттік интеграция үдерісіне басымдық беріледі, оның барысында қосалқы жүйелер сатылы түрде қосылады:

1. Әдетте қосалқы жүйелерді әзірлеудің барлық элементтерді әзірлеу бір уақытта аяқталуына негізделген кестесін жасау мүмкін емес.
2. Инкременттік интеграция қатені анықтау құнын төмендетеді. Егер бірнеше қосалқы жүйелер бір мезгілде интеграцияланатын болса, тестілеу кезінде анықталған қателік бұл қосалқы жүйелердің кез келгенінде болуы мүмкін.

Жеке қосалқы жүйе жұмыс істеп тұрған жүйемен интеграцияланатын болса, қателіктің соңғы қосылған элемент әсерінен немесе қолданыстағы қосалқы жүйенің жаңа қосалқы жүйенің арасындағы өзара әрекеттестіктен туындағанын оңай анықтауға болады. Қазіргі уақытта жүйелердің көпшілігі сатылымда бар бағдарламалық және ақпараттық қамтамасыз етулер интеграциялау жолымен құрылады, іске асыру мен интеграция арасындағы айырмашылық шайылған түрде. Кейбір жағдайларда жаңа ақпараттық және бағдарламалық қамтамасыз етуге қажеттілік жоқ, ал интеграция жүйені іске асыру фазасы болып табылады. Интеграция уақытында және үдеріс аяқталғаннан кейін жүйе тестіленеді. Негізгі назар жүйенің жалпы әрекеті мен компоненттері арасындағы өзара әрекеттістікті тестілеуге аударылуы тиіс. Бұл міндеттеме жеке қосалқы жүйелердің жойылуға тиіс проблемаларын анықтауға көмектеседі. Қосалқы жүйелердің басқа қосалқы жүйелерге қатысты қате жорамалдар негізінде орын алған қателіктері көбінесе жүйені интеграциялау сатысында анықталады. Бұл түрлі қосалқы жүйелердің іске асырылуына жауапты мердігерлердің арасында даулардың туындауына алып келеді.

Проблемалар қосалқы жүйелердің өзара әрекетіне байланысты екені анықталғанда, мердігерлер қандай қосалқы жүйенің дұрыс емес екені туралы дауласады. Бұл туындаған проблеманы қалай шешуге болатыны туралы келіссөздер бірнеше апталарға, тіпті айларға созылуы мүмкін. Жүйені әзірлеу үдерісінің соңғы сатысы жеткізу және пайдалануға енгізу болып табылады. Бағдарламалық қамтамасыз ету ақпараттық қамтамасыз етуге орнатылады да пайдалануға дайындалады. Бұл жүйенің елеулі конфигурациясын талап етуі мүмкін, нәтижесінде ол қолданылатын ортаның барлық ерекшеліктері, ескі жүйенің деректерін көшіру, пайдаланушылық құжаттаманы және оқытуды дайындау көрсетіледі. Аталған сатыда Сізге жаңа жүйенің ортаның басқа жүйелерімен өзара әрекеттестікте екеніне көз жеткізу үшін жүйені екінші рет конфигурациялау жүргізу қажет. Жүйені пайдалануға енгізгенде көптеген қиындықтар туындауы мүмкін. Шынайы пайдаланушылық орта жүйені әзірлеушілер ойластырғаннан өзгеше болуы

мүмкін, мұндай айырмашылықтарға жүйенің бейімделуі қиындықсыз өтпеуі мүмкін. Қолдағы деректер көлемді тазалауды талап етуі, ал олардың бір бөлігі жай ғана жоқ болуы мүмкін. Басқа жүйелермен өзара әрекеттестік сәйкес емес түрде құжатталуы ықтимал.

Жүйені әзірлеу үдерістерінің функционалдық сенімділікке және қорғанысқа әсері айқын көрінеді. Осы аталған үдерістерді жүзеге асыру кезінде функционалдық сенімділікке және қорғанысқа, сонымен қатар, шығындар, кестелер, тиімділік және функционалдық сенімділік арасындағы теңгерімге қатысты шешімдер қабылданады. Әзірлеу үдерісінің кез келген сатысында адамның қателері жүйеге қателіктердің енуіне алып келуі мүмкін, ал бұл болашақта жүйенің ақауына әкеп соғады. Тестілеу және валидация үдерістері қолжетімді құралдар мен уақытша шеңберлер санымен шектеледі. Нәтижесінде жүйені тестілеу сәйкес емес болуы да ықтимал. Пайдаланушыларға тек жүйені пайдалану барысында тестілеу ғана қалады. Жүйені пайдалануға енгізу кезіндегі проблемалар жүйенің операциялық ортаға сәйкес еместігінің белгісі болуы мүмкін. Бұл өз кезегінде пайдаланушылық жүйенің қателіктеріне әкеп соғуы ықтимал.

10.5 Жүйені пайдалану

Опрациялық үдерістер – жүйені белгілі мақсаты бойынша пайдалану үдерісі. Мысалы, әуе қозғалысын басқару жүйесі операторлары әуе кемелерінің әуе кеңістігіне енуі және шығуы, оларға жылдамдықты немесе биіктікті өзгерту қажеттілігі, төтенше жағдай туындауы кезінде белгілі үдерістерді сақтауы тиіс.

Жаңа жүйелер үшін аталған операциялық үдерістержүйені әзірлеу сатысында анықталуы және құжатталуы тиіс. Жаңа жүйенің тиімді қолданылуын кепілдендіру үшін операторларды оқыту, басқа да жұмыс үдерістерін бейімдеу қажет. Бұл сатыда белгісіз проблемалар туындауы мүмкін, себебі жүйе ерекшелігінде қателіктер мен олқылықтар болуы ықтимал. Жүйенің өзі барлық талаптар мен ерекшеліктерге сәйкес келсе де, оның функциялары шынайы операциялық қажеттіліктерді қанағаттандырмауы мүмкін. Тиісінше, операторлар жүйені оны әзірлеушілер жорамалдағандай қолданбауы мүмкін.

Оператордың болуының негізгі артықшылығы – адамдардың болжанбаған оқиғалар кезінде олардың мұндай жағдайларда тәжірибесі болмаса да тиімді әрекет ету қабілеті. Осылайша, бір әрекет дұрыс орындалмай жатса, кейде белгіленген үдерісті бұзуға тура келсе де операторлар жағдайды түзей алады. Сол сияқты операторлар үдерістерді бейімдеу және жетілдіру мақсатында өз ғимараттарын да пайдаланады. Әдетте нақты операторлық үдерістер жүйені әзірлеушілер болжағандарға қарағанда айырмашылықта болады.

Бұл дегеніміз, Сізге иілгіш, бейімделгіш операциялық үдерістерді әзірлеу қажет. Операциялық үдерістер елеулі шектеулер қоймауы, операциялардың белгілі режимде орындалуын талап етпеуі тиіс, ал бағдарламалық қамтамасыз ету белгілі үдерісті сақтау фактісіне негізделмеуі қажет. Операторлар үдерістерді әдетте

жақсартып отырады, себебі олар шынайы оқиғада ненің тиімді, ненің тиімді емес екенін біледі.

Пайдалануға енгізгеннен кейін туындайтын проблема – бұл жаңа жүйенің ескі жүйелермен қатар өзара әрекеттістігі болып табылады. Физикалық сыйыспаушылық проблемалары немесе бір жүйеден екіншісіне деректерді беруде қиыншылықтар туындауы мүмкін. Азырақ байқалатын проблемалар пайдаланушылық интерфейстердің айырмашылығынан болады. Жаңа жүйені енгізу оператор қателігі жиілігін ұлғайтуы мүмкін, себебі, оператор басқа жүйеге арналған пайдаланушы интерфейсі командаларын пайдаланады.

10.5.1 Адамның қателері

Берілген тарауда бұрынырақ біз әлеуметтік-технологиялық жүйелердің маңызды сипаты олардың детерминистік еместігі болып табылатынын айтқанбыз. Бұған себептердің бірі жүйенің бөлігі болып табылатын адамдар өздерін бірқалыпты ұстай бермейтіндігімен түсіндіріледі. Кейде, жүйені пайдалану кезінде олар қателіктер жібереді де, нәтижесінде бұл жүйенің істен шығуына алып келеді. Мысалы, оператор нақты бір әрекеттердің болғаны туралы арнайы жазба жасауды ұмытып кетуі мүмкін, ал екінші оператор оның әрекетінен хабарсыз болғандықтан, бұл әрекетті қайтадан жүзеге асырады (қателесіп). Егер бұл жерде әңгіме банкілік шоттан соманы алу немесе аудару туралы болса, жүйенің қателігі туындайды, себебі, шоттағы түпкілікті сома дұрыс емес болып шығады.

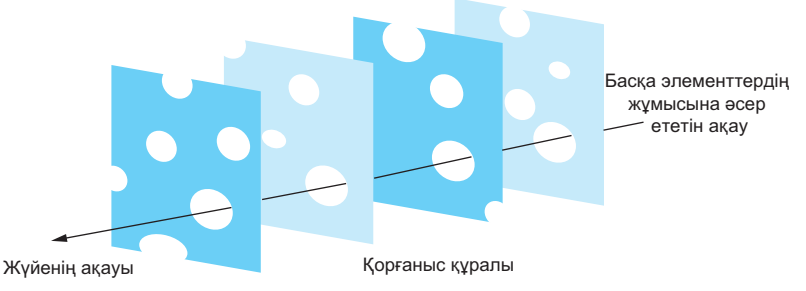
Ризон (2000) айтқандай, адамның қатесі әрқашан туындайтын болады, оларды қарастырудың екі түрлі тәсілі бар:

1. *Тұлғалық тәсіл.* Қателіктер үшін жауапкершілік адамға жүктеледі, ал «қауіпсіз емес әрекеттер» (мысалы, оператор қорғаныс қоршауын белсендіруді ұмытып кеткенде) адамның зейінсіз немесе абайсыз әрекетінің салдары болып саналады. Мұндай тәсілді ұстанатын адамдар адамның қателіктері санын тәртіптік шаралар қолдану, қатаң процедуралар енгізу, қайта оқыту және т.б. арқылы қысқартуға болады деп есептейді. Олар орын алған қателік оған жауапты адамның кінәсі деп санайды.

2. *Жүйелік тәсіл.* Негізгі жорамал, адамдардың қателесуге бейімділігі және әрқашан қателік жіберетіні болып табылады. Адамдар жіберетін қателер көбінесе жүйені жобалау шешімдерінің салдары болып саналады және жүйе операторларына кері әсерін тигізетін жұмыстың немесе ұйымдастыру факторларының қате әдістеріне алып келеді. Жақсы жүйелер адамның қателік жіберу мүмкіндігін өзі танып білуі тиіс және адам қатесін анықтау және алдын алу құралдарына ие болуы шарт. Қате орын алған жағдайда негізгі міндет бұған кінәлі адамды іздеу емес, жүйенің қатені неге өз бетінше түзетпегенін түсіну болып табылады.

Біз жүйелік тәсіл дұрыс деп, жүйені әзірлеушілер оны пайдалану кезінде адам қателері орын алу мүмкіндігін ескерулері қажет деп санаймыз. Осылайша, жүйенің қорғанысы мен функционалдық сенімділігін жақсарту үшін әзірлеушілер жүйенің құрамына кіретін, адам қателерінен қорғау құралын да әзірлеулері тиіс. Олар аталған құралды жүйенің техникалық компоненттеріне кіргізуі қажет пе? Егер қажет болмаса, онда олар үдерістердің бөлігі, нәтижелілігі тексерулер мен адам талқысына байланысты операторлар үшін нұсқаулық түрінде болуы мүмкін. Жүйенің мұндай қорғаныс құралдарына мысалдар:

1. Әуе қозғалысын бақылау жүйесі жанжалдық ахуалдардың алдын алатын автоматтық жүйемен жабдықталуы мүмкін. Бақылаушы әуе кемесінің пилотына жылдамдықты немесе биіктікті өзгерту туралы нұсқау бергенде жүйе басқа бір ұшақтың траекториясымен қиылыспайтынына сенімді болу үшін оның траекториясын экстраполяциялайды. Егер траекториялар қиылысатын болса, онда дабыл жүйесі іске қосылады.
2. Дәл осындай жүйе бақылау нұсқаулығының нақты белгіленген тіркеу процедурасына ие болуы мүмкін. Бұл процедуралар бақылаушыға олардың дұрыс нұсқаулық бергенін және ақпараттың жалпы тексеруге қолжетімділігін қамтамасыз еткенін тексеруге көмектеседі.
3. Әуе қозғалысын бақылауда әдетте бір-бірінің жұмысын үнемі бақылап отыратын бақылаушылар командасы болады. Осылайша, қателік жіберуге жол берілген жағдайда да оны апатқа ұшырату оқиғасына дейін жеткізбей анықтау және түзету ықтималдығы жоғарылай түседі.



10.9-сурет. Ризонның «швейцария ірімшігі» үлгісіндегі жүйе ақауының моделі

Кез келген қорғаныс құралдарының белгілі кемшіліктері бар. Ризон оларды «ықтимал шарттар» деп атайды, себебі олар әдетте қандай да бір өзге проблема туындаған жағдайда ғана жүйенің ақауына ықпал етеді. Мысалы, жанжалдық оқиғалардың алдын алатын жүйенің кемшілігі, ол көп мөлшерде жалған дабыл белгілерін береді. Осылайша, бақылаушылар жүйенің шын мәнінде маңызды ескертуін елемеуі де мүмкін. Процедуралық жүйенің кемшілігі әдеттегі емес, бірақ маңызды ақпаратты тіркеу біршама қиындық тудырады. Барлық іске жұмылдырылған адамдар стрестік жағдайда және бір қателікті қайта-қайта жіберіп жүргенде, басқа операторлардың тексеруі оң нәтижелер бермеуі мүмкін. Ықтимал

шарттар жүйеге орнатылған қорғаныс құралдары жүйе операторы жіберген қателікті анықтай алмағанда жүйенің ақауына алып келеді. Адамның қатесі ақау үшін іске қосу механизмі болып табылады, бірақ ақаудың жалғыз ғана себебі болып саналмайды. Ризон мұны «швейцария ірімшігі» тәрізді жүйе ақауының жақсы танымал моделі көмегімен түсіндіреді (*10.9-суретті* қара).

Аталған модельге сәйкес жүйеге орнатылған қорғаныс құралы швейцария ірімшігінің кесектерімен салыстырылады. Швейцария ірімшігінің кейбір сұрыптарында (мысалы, эмменталь) тесіктері бар – міне осы негізде ықтимал шарттарды ірімшіктегі тесіктермен салыстыру жүргізіледі. Аталған тесіктердің орналасуы тұрақты емес, алайда өзгерістері әлеуметтік-технологиялық жүйенің жалпы жай-күйіне байланысты. Егер әрбір қорғаныс құралын ірімшік кесегі түрінде қарастырса, ақау оператор қателік жіберіп, қуыстары бір-біріне сәйкес келіп тізбектеле орналасқанда орын алады. Жүйенің басқа элементтерінің жұмысына әсер ететін қателік осы тесіктер арқылы өтіп жүйенің жалпы істен шығуына алып келеді. Әдетте қорғаныс құралдарының қуыстары мұндай түрде тізбек құрмайды, операциялық қателіктерді жүйе анықтайды. Адам қатесінен болатын жүйелік ақаудың ықтималдығын қысқарту үшін әзірлеушілердің міндеттері:

1. Түрлі қорғаныс құралдарымен жабдықталған жүйелер әзірлеу. Бұл олардың «тесіктері» бір сызыққа тізбектелуін болдырмас үшін түрлі орындарда орналасулары мүмкіндігін арттырады, демек қателіктің өту мүмкіндігі елеулі қысқарады.
2. Жүйенің ықтимал шарттарының санын барынша төмендету. Басқаша айтқанда, әзірлеушілер жүйедегі «тесіктердің» саны мен өлшемін қысқартулары тиіс.

Әрине, жүйе құрылымы тұтас алғанда жүйе ақауына себеп болуы мүмкін басқа элементтердің жұмысына әсер ететін ақаулардың туындауына жол бермеуге тырысуға тиіс. Бұл операторлардың бір жұмысты қайта жасамауына, алаңдамауына, оларға тым көп мөлшердегі ақпаратты өңдеуге тура келмейтініне кепілдік беретін операциялық үдерістерді және жүйені әзірлеуге негізделеді.

10.5.2 Жүйенің эволюциясы

Үлкен, күрделі жүйелердің қызмет мерзімі де біршама ұзақ болып келеді. Өздерінің қызмет мерзімінде олар бастапқы жүйелік қателерді түзету, пайдалану кезінде туындаған жаңа талаптарды жүзеге асыру мақсатында өзгерістерге ұшырайды. Жүйе компьютерлері неғұрлым жаңасына, жылдамдығы неғұрлым жоғары машиналарға ауыстырылады. Жүйені пайдаланатын ұйым қайта құрылуы, яғни жүйені басқаша түрде қолдануға көшуі мүмкін. Жүйенің сыртқы ортасы өзгеріп, соның себебімен жүйе де өзгеруі ықтимал. Осылайша, эволюция (ортаның өзгеруіне бейімделуге қажетті жүйенің өзгеруі) – бұл жүйенің әдеттегі операциялық үдерістерімен қатар жүретін үдеріс. Жүйенің эволюциясы жүйенің операциялық

үдерістерін, ақпараттық қамтамасыз етуді, бағдарламалық қамтамасыз етуді жетілдіру және өзгерту мақсатында әзірлеу үдерісін екінші рет жүзеге асыруды білдіреді.

Мұраланған жүйелер

Мұраланған жүйелер – бұл бұрын әзірленген, көбіне ескі немесе ескірген технологияға негізделген әлеуметтік-технологиялық компьютерлік жүйелер. Аталған жүйелер тек ақпараттық және бағдарламалық қамтамасыз етуден ғана емес, мұраланған үдерістер мен процедуралардан – ескі бағдарламалық қамтамасыз етуде негізделгендіктен өзгертілуі киын міндеттерді орындаудың ескі тәсілдері. Жүйенің бір бөлігін өзгерту барлық басқа компоненттерді қоса өзгертуді талап етеді. Мұраланған жүйелер көбінесе жүйелер бизнесі үшін қиын болып табылады. Оларды ауыстыру үлкен тәуекелмен байланысты болғандықтан сақталады.

http://www.SoftwareEngineering-9.com/LegacySys/

Жүйе эволюциясы бағдарламалық қамтамасыз етудің эволюциясы сияқты (9-тарауды қара) бірнеше себептер бойынша қымбат болып табылады:

1. Ұсынылатын өзгерістерді бизнес пен технология көзқарасы тұрғысынан мұқият талдау қажет. Өзгерістер техникалық прогресс себептерімен байланысты емес жүйенің мақсаттарына сәйкес келуі тиіс.
2. Қосалқы жүйелер ешқашан толық тәуелсіз болып табылмайтындықтан, бір қосалқы жүйені өзгерту жанама түрде басқа қосалқы жүйенің әрекетіне немесе жұмысқа қабілеттілігіне әсер етуі мүмкін. Осылайша, басқа да қосалқы жүйелерге өзгерістер енгізу қажет болады.
3. Жобалаудың соны шешімдерінің шығу себептері көп жағдайда тіркелмейді. Сондықтан жүйенің эволюциясы үдерісінде жобалаудың белгілі шешімдері неге қабылданғанын анықтауға тура келеді.
4. Жүйенің ескіру шамасына қарай олардың құрылымдары өзгерістермен бұрмаланады, сол себепті одан әрі өзгерістер енгізу құны ұлғаяды.

Уақытпен бірге эволюцияланған жүйелер көбінесе ақпараттық қамтамасыз ету мен бағдарламалық қамтамасыз етудің ескірген технологиясына негізделген. Егер олар ұйымда қиын рөл атқаратын болса, онда оларды «мұраланған жүйелер» деп атайды. Ұйымдар әдетте аталған жүйелерді көп жағдайда қуана-қуана ауыстырар еді, алайда, жүйені ауыстыруға байланысты ақталмауы мүмкін шығындардан қауіптенеді.

Функционалдық сенімділік және қорғаныс тұрғысынан жүйені өзгерту бірқатар проблемалар мен кемшіліктердің себебі болып табылады. Егер өзгерістерді жүйені әзірлеген тұлғалардан басқа тұлғалар енгізетін болса,

онда олар жобалаудың нақты шешімдері функционалдық сенімділік және қорғанысты қамтамасыз ету мақсатында қабылданғанын білмеуі мүмкін. Осылайша, жүйелерді өзгерту кезінде оны әзірлеу барысында қорғау мақсатында іске асырылған қорғаныс құралдары жойылып кетуі мүмкін. Сонымен қатар, толық қайтадан тестілеу өткізу құны жоғары болғандықтан, әрбір жеке өзгеріс енгізген сайын тестілеу жүргізу мүмкін емес. Осылайша, жүйенің басқа компоненті қатесінен болатын жанама әсерлер әр уақытта анықтала бермейді.

НЕГІЗГІ ТҰЖЫРЫМДАР

- Әлеуметтік-технологиялық жүйелер құрамына ақпараттық қамтамасыз ету, бағдарламалық қамтамасыз ету және адамдар кіреді, ал жүйе ұйымның ішінде орналасады. Олар ұйымдастырушылық және іскерлік мақсаттар мен міндеттерді қолдауға арналған.
- Адамдық және ұйымдастырушылық факторлар (мысалы, ұйымның құрылымы мен саясаты) әлеуметтік-технологиялық жүйелердің жұмысына елеулі әсер етеді.
- Жүйенің тәуелсіз қасиеттері жекелеген компоненттер емес жүйенің тұтас сипаттамасын көрсетеді. Аталған қасиеттерге жұмысқа қабілеттілік, сенімділік, қолайлылық, қауіпсіздік және қорғаныс жатады. Жүйенің табыстылығы және дұрыс жұмыс істеуі осы тәуелсіз қасиеттерге байланысты болады.
- Жүйені жобалаудың іргетастық үдерістері жүйені сатып алу, жүйені әзірлеу және жүйені пайдалану болып табылады.
- Жүйені сатып алуға қандай жүйені сатып алу керек және кімнің бұл жүйені жеткізуші рөлін атқаратыны туралы шешім қабылдау кезінде жүзеге асырылатын барлық операциялар жатады. Осы сатыда сонымен қатар, жоғарғы деңгейдегі талаптар әзірленеді.
- Жүйені әзірлеу дегеніміз талаптарды анықтау, құрылымдау, құру, интеграциялау және тестілеуді білдіреді. Әзірлеудің критикалық үдерісі түрлі өнім берушілер әзірлеген қосалқы жүйелерді бірлескен жұмыс үшін қосу жүргізілетін интеграциялау болып табылады.
- Жүйе пайдалануға енгізілген кезде операциялық үдерістер және жүйенің өзі бизнес талаптарының өзгеруіне қарай бірқатар өзгерістерге ұшырайды.
- Адамның қатесі қандай жағдайда да орын алатын құбылыс болып табылады, сондықтан жүйелер мұндай қателіктерді анықтауға қабілетті қорғаныс құралдарымен жабдықталуы тиіс, себебі мұндай қателіктер жүйенің істен шығуына алып келеді. Ризонның «швейцария ірімшігі» үлгісіндегі моделі адамның қателігі ықтимал ақаулармен қосылып, жүйенің ақауына қалай алып келетінін түсіндіреді.

ҚОСЫМША ӘДЕБИЕТТЕР

'Airport 95: Automated baggage system'. An excellent, readable case study of what can go wrong with a systems engineering project and how software tends to get the blame for wider systems failures. (*ACM Software Engineering Notes*, 21, March 1996.)http://doi.acm.org/10.1145/227531.227544.

'Software system engineering: A tutorial'. A good general overview of systems engineering, although Thayer focuses exclusively on computer-based systems and does not discuss sociotechnical issues. (R. H. Thayer. *IEEE Computer*, April 2002.) http://dx.doi.org/10.1109/MC.2002.993773.

Trust in Technology: A Socio-technical Perspective. This book is a set of papers that are all concerned, in some way, with the dependability of sociotechnical systems. (K. Clarke, G. Hardstone, M. Rouncefield and I. Sommerville (eds.), Springer, 2006.)

'Fundamentals of Systems Engineering'. This is the introductory chapter in NASA's systems engineering handbook. It presents an overview of the systems engineering process for space systems. Although these are mostly technical systems, there are sociotechnical issues to be considered. Dependability is obviously critically important. (In *NASA Systems Engineering Handbook*, NASA-SP2007-6105, 2007.) http://education.ksc.nasa.gov/esmdspacegrant/Documents/NASA%20SP-2007-6105%20Rev%201%20Final%2031Dec2007.pdf.

ЖАТТЫҒУЛАР

10.1. Күрделі әлеуметтік-технологиялық жүйелерден тұратын үкіметтік функциялардың екі мысалын келтіріңіз және неге аталған функциялардың жақын болашақта автоматтандырылмайтынын түсіндіріңіз.

10.2. Компьютерлік жүйе орналасқан ортаның жүйеге қалай әсер ететінін және неліктен жүйенің ақауына себеп болатынын түсіндіріңіз. Өз жауабыңызды осы тарауда пайдаланылған мысалдармен келтіріңіз.

10.3. Неге жүйе компоненттері қасиеттері негізінде күрделі жүйенің тәуелсіз қасиеттері туралы қорытынды жасауға болмайды?

10.4. Неге кейде әлеуметтік-технологиялық жүйенің ақауы бар-жоғын анықтау қиын? Өз жауабыңызды осы тарауда пайдаланылған клиника базасының жүйесі туралы мысалдармен келтіріңіз.

10.5. «Қаскөй» проблема дегеніміз не? Неге медициналық мұрағат ұлттық жүйесін әзірлеуді «қаскөй» проблема деп атауға болатынын түсіндіріңіз.

10.6. Еуропалық мұражай консорциумы үшін Ежелгі Грецияға арналған тақырыптық көрме жәдігерлерін көрсететін виртуальдық мұражай мультимедиялық жүйесін әзірлеу қажет. Жүйе әдеттегі Интернет-браузердің көмегімен пайдаланушыларға ежелгі Греция қалаларының үш өлшемді модельдерін көруге, сонымен қатар, виртуальдық шынайылыққа ену мүмкіндігін беруі тиіс. Жоғарыда аталған жүйе консорциумға кіретін мұражайларда орнатылғанда қандай саяси және ұйымдастырушылық қиындықтар туындауы мүмкін.

10.7. Неге жүйенің интеграциясы жүйені әзірлеудің маңызды бөлігі болып табылады? Жүйені интеграциялау барысында белгілі қиындықтар туғызуы мүмкін әлеуметтік-технологиялық сипаттағы үш проблеманы көрсетіңіз.

10.8. Неге мұраланған жүйелердің бизнес үшін критикалық рөл атқаруы мүмкін екенін түсіндіріңіз.

10.9. Инженер-электрик немесе бағдарламалық қамтамасыз етуді әзірлеуші сияқты жүйе әзірлеушісі мамандығын дербес мамандық деп танудың «иә» және «қарсы» аргументтерін келтіріңіз.

10.10. Өзіңізді қаржы жүйесін әзірлеу үдерісіне тартылған инженермін деп есептеңіз. Орнату кезінде Сіз аталған жүйенің жұмыс орындарын едәуір қысқартатынын анықтадыңыз. Ортаның адамдары жүйені орнатуды аяқтау үшін Сізге негізгі ақпаратқа рұқсат беруден бас тартуда. Сіздің аталған проблемаға жүйе әзірлеушісі ретінде араласу деңгейіңіз қандай? Сіз келісімшартқа сәйкес орнатуды аяқтауға кәсіби жауапкершілікті жүктейсіз бе? Сізге клиент ұйым аталған проблеманы шешкенше жұмысыңызды тоқтатып қою қажет емес пе?

ПАЙДАЛАНЫЛҒАН ӘДЕБИЕТТЕР

Ackroyd, S., Harper, R., Hughes, J. A. and Shapiro, D. (1992). *Information Technology and Practical Police Work*. Milton Keynes: Open University Press.

Anderson, R. J., Hughes, J. A. and Sharrock, W. W. (1989). *Working for Profit: The Social Organization of Calculability in an Entrepreneurial Firm*. Aldershot: Avebury.

Checkland, P. (1981). *Systems Thinking, Systems Practice*. Chichester: John Wiley & Sons.

Checkland, P. and Scholes, J. (1990). *Soft Systems Methodology in Action*. Chichester: John Wiley & Sons.

Mumford, E. (1989). 'User Participation in a Changing Environment—Why we need it'. In *Participation in Systems Development*. Knight, K. (ed.). London: Kogan Page.

Reason, J. (2000). 'Human error: Models and management'. *British Medical J.*, **320** 768–70.

Rittel, H. and Webber, M. (1973). 'Dilemmas in a General Theory of Planning'. *Policy Sciences*, **4**, 155–69.

Stevens, R., Brook, P., Jackson, K. and Arnold, S. (1998). *Systems Engineering: Coping with Complexity*. London: Prentice Hall.

Suchman, L. (1987). *Plans and situated actions: the problem of human-machine communication*. New York: Cambridge University Press.

Swartz, A. J. (1996). 'Airport 95: Automated Baggage System?' *ACM Software Engineering Notes*, **21** (2), 79–83.

Thayer, R. H. (2002). 'Software System Engineering: A Tutorial.' *IEEE Computer*, **35** (4), 68–73.

Thomé, B. (1993). 'Systems Engineering: Principles and Practice of Computer-based Systems Engineering'. Chichester: John Wiley & Sons.

White, S., Alford, M., Holtzman, J., Kuehl, S., McCay, B., Oliver, D., Owens, D., Tully, C. and Willey, A. (1993). 'Systems Engineering of Computer-Based Systems'. *IEEE Computer*, **26** (11), 54–65.

11.

Сенімділік пен қауіпсіздік

Мақсаттары

Бұл тараудың мақсаты бағдарламалық қамтамасыз етуді таныстыру мен оның қауіпсіздігі болып табылады.
Осы тараумен танысқанда, сіз:

- жүйелі бағдарламалық қамтамасыздандыру функционалды мінездемелеріне қарағанда сенімділік пен қауіпсіздік анағұрлым қажет екенін түсінесіз;
- сол сияқты сенімділіктің ең басты төрт қырымен, дәлірек айтқанда қолжетімділік, сенімділік, сақталу және қауіпсіздікпен танысасыз;
- Қауіпсіздік пен сенімділікке байланысты пайдаланылатын арнайы терминологияларды білетін боласыз;
- сенімді, қауіпсіз бағдарламаларды қамтамасыз ету мақсатында жүйелерді қалыптастыру үшін қателерді болдырмау және оларды пайдалану барысында байқап, жойып отыру мен жұмысқа келтіретін кедергілерді азайтудың қажет екенін түсінесіз.

Мазмұны

Компьютерлік жүйе біздің өмірімізге үңги енген сайын оны пайдалану барысында бағдарламалық қамтамасыз ету мен жүйе жұмысының істен шығу проблемалары да көбейе түседі. Серверлік бағдарламалармен қамтамасыз етудегі жүйенің істен шығуы электронды коммерциямен айналысатын компанияларға табыстарын төмендетіп, өз клиенттерін жоғалтуға соқтырады. Автокөліктерде орнатылған бағдарламалық қамтамасыз етудегі қателіктер оның қымбат тұратын жөндеу жұмыстарына себеп болып, тіпті жол апаттарына жеткізуі мүмкін. Зиянды бағдарламалар компания компьютерлерін зақымдап, оны тазалауға қомақты қаржы жұмсау және аса қажетті ақпараттарды мүлдем жоғалтып алу қаупін тудырады. Кең көлемдегі бағдарламалық қамту үкімет үшін, компаниялар мен жеке тұлғаларға өте маңызды болғандықтан, оның сенімді болуы да аса қажет. Бағдарламалық қамту қалаған уақытында қолжетімді және қажетсіз ауытқуларсыз жұмыс істеп, құпия ақпараттарды шет шығармауы керек. «Сенімділік» деген компьютердегі терминді қолжетімді, сенімді, сақталуы мен қауіпсіздігі үшін 1995-жылы Laprie ұсынған болатын. Біздің *11.1-бөлімімізде* көрсетілгендей бұл аталған ерекшеліктер бір-бірімен тығыз байланысты болғандықтан, төртеуіне ортақ бір терминді қолдануға болады.

Келесі аталатын функцияларға қарағанда, жүйенің сенімді болуы маңыздырақ:

1. Жүйедегі ауытқу ондағы қолданушыларға әсер етеді. Көптеген жүйелер сирек пайдаланатын функциялармен толығады. Егер осы функциялық толығуды алып тастаса аздаған пайдаланушыларға әсер ететін болады. Ал қолжетімділікке әсер ететін жүйедегі ауытқулар барлық пайдаланушыларға кері әсерін тигізеді. Ол өз кезегінде жұмысты дұрыс жүргізуге мүмкіндік бермейді.
2. Әдетте пайдаланушылар сенімсіз, қорғалмаған немесе қауіпті жүйеден бас тартады. Егер жүйе сенімсіз, қауіпті деп саналса, пайдаланушы оны қолданбайды. Оған қоса тұтынушы бұл компания шығарған өзге тауарларды да сатып алмайды, өйткені оның басқа өнімдері де сенімсіз, қауіпті болуы мүмкін.
3. Жүйенің істен шығуына байланысты шығын көлемді болады. Мысалы, реакторларды немесе ұшақтардың ұшу бағытын басқару саласында жүйелердің істен шығуы тым қымбатқа түсері белгілі.
4. Сенімсіз жүйе ақпараттардың жоғалуына себепкер болады. Жоғалтылған ақпараттарды қайта жинау қиын. Ақпаратты сақтау мен жинақтау оны компьютерлік сараптамадан өткізуге қарағанда, анағұрлым қымбатқа түседі. Бүлінген немесе жоғалған ақпараттарды қалпына келтірудің бағасы қашан да жоғары.

Алдында *10-тарауда* айтқанымыздай, бағдарламалық қамту үлкен жүйенің бір саласы. Ол аппараттық қамтулы операциялық орталықта қызмет жасайды. Бұл жүйеде пайдаланушы адамдар мен ұйымдастыру және бизнес-процестер бағдарламалық қамтуды пайдалану арқылы жұмыс атқарады. Сондықтан сенімді жүйені жобалағанда мына жағдайларды есте ұстау қажет:

1. Аппараттық немесе аппарат құралдарының ауытқуы оның конструкцияларының, бөлшектерінің дұрыс дайындалмауынан, оның көрсететін қызмет уақытының өткенінен болуы мүмкін.

Қатерлі жүйелер

Жүйенің кейбір түрлері «қатерлі маңызы бар жүйелер» болып табылады. Мұндай жүйенің ауытқуы адамдардың жарақат алуы, қоршаған ортаға залал әкелуі және экономикалық көлемді шығынға ұшыратуы мүмкін. Оған медициналық құралдарға орнатылған (қауіпсіздікке арналған) инсулин помпасы, ғарыштық ұшу жүйесіндегі (тапсырмаларды орындаудағы қатерлер) және онлайн жағдайында ақша аудару жұмысын (коммерциялық қызмет көрсету саласындағы қатер) мысалға келтіруге болады.

Қатерлі жағдайды қалпына келтіру өте қымбатқа түседі. Қауіпсіз жүйені жасағанда ауытқулардың барынша аз болуын, ауытқу орын алған жағдайда қайта орнына келтіру механизмінің қосылуын қамтамасыз ету қажет.

http://www.SoftwareEngineering-9.com/Web/Dependability/CritSys.html

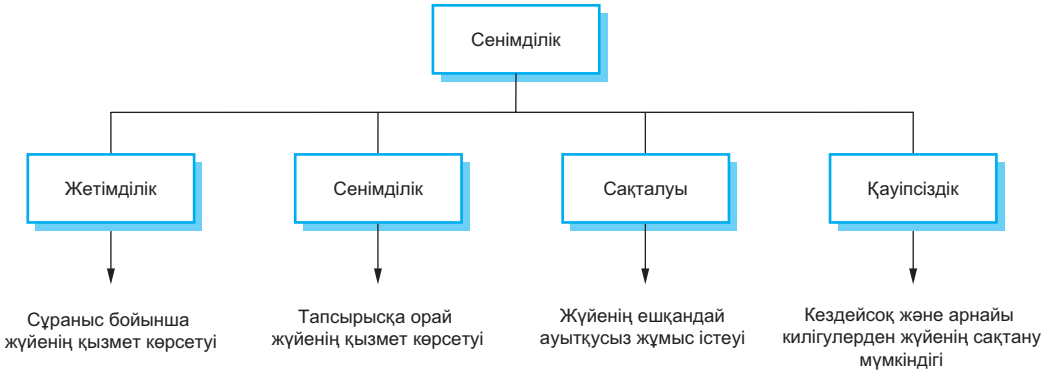
2. *Бағдарламалық қамтудағы ақаулар мен ауытқулар* спецификалық, жобалық немесе қолдану барысындағы қателердің салдарынан болуы мүмкін.
3. *Пайдалану кезіндегі істен шығу* адамның жүйені дұрыс басқармауы мен қате қолдануынан орын алады. Аппараттық және бағдарламалық қамту қазір сенімді болғандықтан, дұрыс қолданбау жүйенің ауытқуының жиі де басты себебі болып келеді. Мұндай ауытқулар өзара тығыз байланысты. Аппараттық қамту құрамының істен шығуы жүйелі операторлардың күтпеген жағдайды қалпына келтіріп, қосымша күш жұмсау керектігін көрсетеді. Бұл оларды күйзеліс күйге түсіріп, пайдаланушыны қате жіберуге мәжбүрлейді. Сөйтіп бағдарламалық қамтуда ауытқулар пайда болып, операторлардың жұмысы көбейіп, жүйкесіне әсер етеді. Жобаны жасаушылардың бағдарламалардың бір қырынан ғана емес, тұтас жүйесін қарастырғаны жөн. Егер құрал мен бағдарламалық қамту немесе операциялық процестер жалпы жүйенің әлсіз жерлері ескерілмей жекелеген бағытта жасалса, онда өзара байланыс кезінде жалпы жүйенің ауытқуы әбден мүмкін.

11.1 Сенімділік ерекшеліктері

Компьютердің істен шығуы бәрімізге таныс жағдай. Біздің компьютерлеріміз өзімізге белгісіз себептермен сынып немесе жұмысы ауытқып жатады. Бұл

компьютердегі жүйемен басқарылатын бағдарламалар дұрыс жұмыс істемей, кейде сақталған ақпараттарды бұзады. Мұндай ауытқуларға біз үйреніп алсақ та өзіміз пайдаланып жүрген компьютерге толық сене қоймаймыз.

Компьютерге деген сенімділік оның жүйесінің, кепілділігінің белгісі. Шын мәнінде, сенімділік оған деген сенімімізді көрсетеді. Жұмыс барысында пайдаланушы жүйенің еш ауытқусыз, өз ойлағанындай нәтиже күтеді. Бұл сан жағынан алғандағы сенімділікті ғана білдірмейді. Сұраныс бойынша жүйенің қызмет көрсетуі



11.1-сурет. Сенімділіктің басты ерекшеліктері

Бағдарламалық жүйеге байланысты «сенімсіз», «сенімді», «өте сенімді» деген терминдерді біз шартты түрде қолданып жүрміз. Сенімді мен пайдалы деген сөздер бір ұғымды білдірмейді. Бұл кітапты жазу үшін қолданылған мәтіндік процессорлар ең сенімді жүйе деп айта алмаймын. Кей кездерде ол да іске қосылмай, қайта қосуға тура келеді. Соған қарамастан, үлкен пайда келтіріп отырғандықтан, ондай ауытқуларға шыдауға болады. Сондықтан, жүйеге сенбегендіктен өз еңбектерімді бірнеше даналарын жасап сақтап қоюға тырысамын. Осылайша кездейсоқ болатын ауытқулардан сақтанып, өз сенімділігімді арттырамын.

Сақтанудың *11.1-суретінде* көрсетілгендей негізгі төрт түрі бар.

1. *Жетімділік.* Шартты түрде бұл пайдаланушының жүйені кез келген уақытта іске қосып, пайдалы қызметін көру болып табылады.
2. *Сенімділік.* Бұл да шартты түрде берілген уақыт аралығында жүйенің қолданушының сенім артқандай қызмет көрсетуі.
3. *Сақталуы.* Жүйенің қауіпсіздігі адамдар мен қоршаған ортаға тигізер залал мүмкіндігінің бағасы болып есептеледі.
4. *Қауіпсіздік.* Бұл жүйенің кездейсоқ немесе арнайы килигулерден сақталу мүмкіндігінің көрсеткіші болып есептеледі.

11.1-суретінде көрсетілген сенімділік ерекшеліктерін бұдан да басқа бірқатар қарапайым бөлімдерге тарқатуға болады. Мысалы, қауіпсіздікке тұтастық (бағдарлама мен жүйедегі ақпараттардың бүлінбей сақталуын қамтамасыз ету)

пен «құпиялықты» (ақпараттың тек қана белгілі бір адамдарға ғана ашық болуын жасау) сақтау сияқты тәсілдер кіреді. Сонымен бірге сенімділік «дұрыстықты» (жүйенің алдын ала көрсетілгендей жұмысты қамтамасыз етуі), «дәлдікті» (ақпараттың өз деңгейінде жетуі) және «өз уақытында» (ақпараттың қажет кезінде берілуі) болуын талап етеді.

Әрине, бұл осы жүйеге байланысты қолданылған сенімділіктің барлық ерекшеліктері емес. Инсулиндік помпыға қатысты бірінші тарауда көрсетілген жетімділік (ол қалаған кезде жұмыс істеу керек), сенімділік (инсулиннің дұрыс мөлшерін беру) және қауіпсіздік (инсулиннің қатерлі мөлшерін жібермеу) басты ерекшеліктері болып табылады. Егер біз айтқан помпа құпия ақпаратты сақтай алмаса, онда қауіпсіздік туралы сөз болуы да мүмкін емес. Ал, жергілікті жерлердегі ауа райын зерттеу жұмыстарында жетімдік пен сенімділік ең басты ерекшелік екенін ескеруіміз керек. Науқастар туралы ақпарат жүйесінде құпияны сақтау үшін қауіпсіздік ауадай қажет.

Сенімділіктің осы төрт ерекшелігімен қатар өзгелерін де атауға болады.

1. *Жүйенің ауытқуы* кез келген уақытта болуы мүмкін, оны тез орнына келтіруге болса ғана оның санын азайта аламыз. Ол үшін орын алған проблемаға сараптама жүргізуге мүмкіншілік, ауытқу болған бөлшектерге өзгертулер енгізуге жетімділік жасау керек. Бағдарламалық қамту жүйесін жөнге келтіру үщін пайдаланушы мекеме негізгі кодты білуі мен оған өзгерістер енгізу тәсілін де меңгергені абзал. Ашық бағдарламалық қамтуды оны іске асыру оңай, бірақ қайталанған жағдайда ол қиындай түседі.
2. *Технологиялық.* Жүйені пайдалануда жаңа талаптар пайда болады. Сондықтан жүйені өзгерту, оның пайдасын арттыруды көздеу арқылы жаңа талаптарды ескеру қажет. Сонда ғана бағдарламалық қамту жаңа талаптарға бейімделіп, қателердің кетпеуіне мүмкіндіктер туғызады.
3. *Тұрақтылық.* Ғаламторда тұрақтылық басты сапаның бірі. (Эллисон және басқалар, 1999 ж.). Жүйенің бүлініп, істен шыққан кезінде тұрақтылық қызмет көрсетуге жарамды сапасымен ерекшеленеді. Тұрақтылық жұмысы жүйедегі басты компоненттерді табуға негізделген және барынша аз, нақты қызмет түрін көрсетуге бағытталған. Тұрақтылықты дамыту үшін түрлі шабуылдарға тойтарыс беретін қуатты күшейту, шабуылды анықтау мен залалдарды қалпына келтіру жұмыстары атқарылады (Эллисон, 1999, 2002 жж.) Бұл туралы *14-тарауда* толық айтылады.
4. *Қателерге қарсы тұрақтылық*, бұл ерекшелікті жүйені пайдаланудың құрамдас бөлігі деп қарастырған дұрыс. Ол жүйенің пайдаланушылардың жіберген қателерін болдырмайтын мүмкіндік дәрежесін байқатады. Қателер пайда болғанда оны тауып, автоматты түрде түзеп немесе пайдаланушыға ақпаратты қайта енгізуді талап етеді.

Сенімділік деген ұғым жүйедегі жетімділік, қауіпсіздік, сақталу сияқты ерекшеліктермен өзара тығыз байланысты. Жүйедегі ақпаратты қасақана бұзса, ол бірден сенімсіз болмақ. Қауіпсіз пайдалану жүйенің сенімді және жетімді бо-

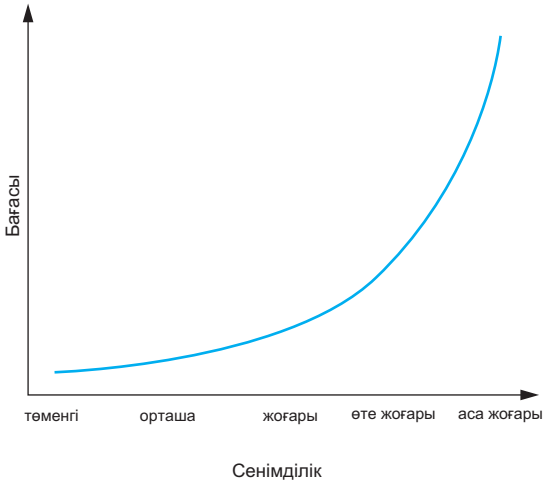
луына тікелей байланысты. Пайдалану кезінде тұрақты шабуылды жоққа шығару жүйенің қауіпсіздігіне кері әсерін тигізеді. Жүйеге вирус кірсе, сіз оған деген сенім мен қауіпсіздікті жоғалтасыз. Өйткені вирустан соң жүйеде ауытқулар пайда болады. Сондықтан сенімді бағдарламалық қамту жасағанда, сіз ешқандай кездейсоқ қате жібермегеніңізге кепілдік беруіңіз керек.

1. Жүйенің сенімділігіне әсер ететін, қалған қателерді анықтайтын тиімді валидациялық және верификациялық процестерді жасадыңыз.
2. Сіз жүйедегі жетімділік пен қауіпсіздікті сыртқы шабуылдардан қорғайтын механизмді құрастырдыңыз.
3. Сіз операциялық орта үшін дұрыс құрылған кең жүйе мен қосымша бағдарлама жасадыңыз.

Бұдан басқа сіздің құрған бағдарламаңыз ең жетілдірілген деген ойдан аулақ болып, оның ауытқуы мүмкін екенін есте ұстаған жөн. Сол себепті сіздің жүйеңізде жұмысты тез арада қайта қалпына келтіру механизмі болуы шарт.

Істен шығуға қарсы тұрақтылық дегеніміз – өзіндік бақылау жүргізу, қателерді анықтау, ауытқу орын алғанға дейін қателерден кейін қалпына келтіруге көмектесетін қосымша коды бар сенімді жүйе деген сөз. Жүйенің жұмысында тексеру тұрақты қажет болғандықтан, ол жүйенің өнімділігіне әсер етеді. Сол себепті жобаны жасаушылар өнімділік пен сенімділіктің тең ортасын таба білгені дұрыс. Жүйенің жұмысын баяулататындықтан, оған бақылау жасауға тура келуі мүмкін. Соған қарамастан, кейбір істен шығулар анықталмаған ақаулардың салдарынан болып, қосалқы қатерлер туындайды.

Концепциялау мен іске асыру және тексеруге жұмсалған қосымша қаржыландырумен бірге жүйедегі сенімділікті арттыратын жұмыста шығындарды барынша көбейтеді. Жекелеп алғанда, қауіпсіздікті қамтамасыз ететін басқару жүйесінің өте сенімді болу үшін жұмсалатын шығын да шаш етектен.



11.2-сурет. Бағаның сызған доғасы/сенімділік

Жүйенің талаптарға сай екенін тексеруден бөлек сыртқы тексеру органдарына оның қауіпсіз екенін дәлелдеуге тура келуі мүмкін. Мысалы, авиациялық жүйе ұшу қауіпсіздігіне әсер ететін қатердің төмендігіне тексеру органдарының көзін жеткізу қажет.

11.2-сурет шығын мен қосымша жетілдіру жұмыстары бір-біріне тәуелді екенін көрсетеді. Егер сіздің бағдарламаңыз сенімсіздеу болса, оны жоғары деңгейдегі бағдарламалық қамту арқылы шығын мөлшерін төмендетуге болады. Алайда сіз жақсы жүйені қолдансаңыз, жетілдіруге шығын көп шығып, одан үнемдік пайда аз болады. Сол сияқты сенімділікті көрсететін бағдарламалық қамтамасыз етуді тестілейтін проблема да бар. Ол орын алған істен шығулар мен ауытқуларға жасалатын көптеген сынақтар мен сараптамалардан тұрады. Егер сіздің бағдарламалық жүйеңіз сенімді болса ауытқулар да азаяды. Сондықтан бағдарламалар барысындағы проблемаларды жиірек сынақтан өткізу керек болады. Тексерулер қымбат тұрғандықтан, жоғары сенімділіктің де құны көтеріледі.

11.2 Жетімділік пен сенімділік

Жүйедегі жетімділік пен сенімділік өз ерекшеліктері арқылы тығыз байланыста болып, сандық негізде көрініс береді. Жетімділік пайдаланушының сұранысы бойынша жүйенің іске қосылып қызмет көрсету мақсатында жұмыс істейтінін анықтайды. Жүйенің сенімді жұмысын жүйе жобалағандай қызмет етуі анықтайды. Мыңнан екі қосылу сәтсіз болса, ауытқуға шаққандағы сенімділік деңгейі 0,002 мөлшерге сай болғаны. Егер жетімділік 0,999 болса жүйе 99,9 пайызға жұмыс істейтінін білдіреді. Бұл ерекшелік кейде басқаларына қарағанда, маңызды рөл атқарады. Егер қолданушыға үзіліссіз қызмет қажет болса, жүйеге жетімділікке де жоғары талап қойылады. Ол кез келген уақытта қолжетімді болуы шарт. Соған қарамастан, егер ауытқушылыққа байланысты шығындар аз болса, жүйе тез қалпына келіп, істен шығудың пайдаланушыға кері әсері азаяды. Мұндай жүйеде сенімділікке деген талап та аз болады.

Жоғары жетімділікті талап ететін байланысты қосу телефон стансасы осыған мысал бола алады. Пайдаланушылар телефон қоңырауын күткендіктен жетімділік жоғары маңызға ие. Байланыс орнатқанда жүйеде ақау пайда болса, ол тез арада қалпына келеді. Әдетте байланыс стансасы жүйені тоқтатып қайта қосады. Бұл тез жасалғандықтан, пайдаланушы жүйеде ауытқу болғанын байқамай да қалады. Сонымен сенімділік емес, керісінше жетімділік бұл жүйеде басты маңызға ие.

Жүйенің сенімділігі мен жетімділігін төмендегідей анықтауға болады:

1. *Сенімділік*. Белгілі бір ортада, белгілі бір уақыт аралығында, белгілі бір мақсатта жүйенің ақаусыз жұмыс істеу мүмкіндігі.
2. *Жетімділік.* Белгілі уақыт аралығында талап еткен қызметтің жүйеде көрсетілуі.

Сенімді жүйе жасауда біздің сезімдік мүмкіндігіміз өзіміз ойлағаннан әлдеқайда биік болып шығуы ғажап емес. Бұл жерде жүйенің қолдану орталығы есепке алыну қажет. Егер бір ортада жақсы көрсеткішке жетсе, келесі бір орта-

да ол бола қоймайды. Мысалы, мәтіндік процессордың сенімділігін бағалау үшін бағдарламалық қамту жүйесі мүлдем қызықтырмайтын бір кеңсені алайық. Олар пайдалану нұсқауларын орындап, жүйеде түрлі сынақ жасауға бармайды. Ал, университетті алсақ мұндағы жағдай басқаша. Студенттер жүйеде түрлі күтпеген сынақтар жүргізері анық. Бірақ бұл сынақтар жүйеге кездейсоқ қиындықтар мен ауытқулар әкелуі мүмкін.

Бұл қалыпты анықтаулар ауытқулар мен жетімділіксіздіктің соңы неге апарып соғарын ескермейді. Адамдар сәл ауытқуларды әдеттегідей қабылдап, шығынды көп талап ететін көлемді ауытқуларды ғана ауыр сезінеді. Мысалы, сақталған ақпараттарды бүлдірген ақаулар қайта қосу арқылы орнына келетін ауытқулардан анағұрлым қиын көрінеді.

Термин	Баяндау
Адами қате	Жүйені бүлдіруге әкелетін адами факторлар. Мысалы, үйде ауа райын зерттейтін бағдарлама бар делік, бағдарламашы келесі хабардың уақытын анықтау үшін өтіп жатқан уақытқа тағы бір сағат қосады. Бұл тек қана сағат 23.00 пен түн ортасы ауғанша мүмкін болмайды.
Жүйелік қате	Жүйеде қателіктердің орын алуына әкелетін бағдарламалық қамтудағы ерекшеліктер. Бір сағат қосу үшін алдын ала сағат 23.00 немесе одан асқанын ескермей кодты қосу ауытқуларға апарады.
Жүйелік қате	Пайдаланушы күтпеген жағдайда жүйенің ауытқу себептері уақыттың дұрыс қойылмауынан (24.xx емес 00xx) болады.
Жүйенің істен шығуы	Жүйе пайдаланушы күткендей жұмысты атқармай тұрғанда кейбір оқиғалардың белгілі бір уақытта орын алуы. Мысалы, ауа райы туралы ақпарат уақыт дұрыс таңдалмаған соң жарияланбайды.

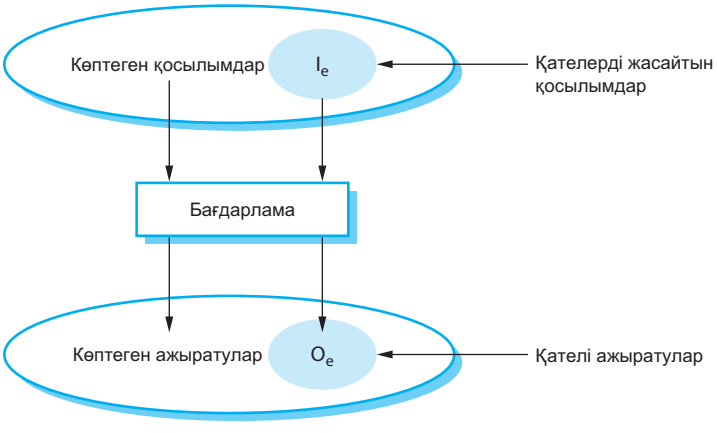
11.3-сурет. Сенімділік терминологиясы

Сенімділікті анықтауды, оның техникалық мүмкіндігін іске асырумен байланыстырамыз. Бірақ берілген техникалық мүмкіндіктері барлық жерде біздің күткен талапқа сай келмеуі мүмкін. Өкінішке орай, құжатта көрсетілген техникалық жазбалар шындығында дұрыс берілмейді. Сондықтан бағдарламаны жасаушыларға жүйе жайында бас қатыруына тура келеді. Олар бұл саладағы сарапшы болмағандықтан, жүйені жұмысқа қоса алмайды. Кейде пайдаланушылар техникалық нұсқауларға назар да аудармайды. Жетімділік жүйедегі ауытқуларға ғана тәуелді емес, оны жөндеуге кететін уақытқа да байланысты. Мысалы, А жүйесінде ауытқу жылына бір рет болса, В жүйесінде айына бір рет орын алады. Яғни, А жүйесі В жүйесіне қарағанда сенімдірек дегенді білдіреді. Айталық, А жүйесін ауытқудан кейін қайта іске қосуға үш күн қажет болса, ал В жүйесіне 10 минут керек делік. Бұл жерде В жүйесі жетімдірек болмақшы. Жүйелердің қанша уақыт іске қосылмауы да кейбір кезде маңызға ие болмайды. Мысалы, жүйе түнгі сағат 3-4 аралығында жетімсіз

болса, көптеген пайдаланушыларға әсері байқалмайды. Бірақ жұмыс уақытындағы жүйенің 10 минут әрекетсіз қалуы көптеген пайдаланушыны әлекке салары анық.

Сенімділік пен жетімділік проблемалары жүйедегі ауытқуларға байланысты. Ауытқулардың кейбірі техникалық қателер мен жүйедегі байланысқа тәуелді. Дегенмен ауытқудың басым көпшілігі жүйедегі қателерге байланысты болып тұрады. Сенімділік туралы сөз болғанда алдымен терминологияның дәлдігін тексерген жөн. «Ауытқу» мен «Істен шығу» ұғымдарының ара жігін ажырату керек. Мысалы, оны *11.3-суретте* жергілікті жерде ауа райын болжау тәжірибесінен көруге болады.

Егер іске қосу кодтың қателесуіне соқтырса, жүйеде ауытқу жағдайы пайда болады. *11.4-суретте* Литтлвуд (1990 ж.) жұмыс тәжірибесінден көптеген қосулар мен ажыратулардың әсері көрсетілген. Осы іс-қимылдарға бағдарлама бейімделіп соған байланысты сұранысты орындайды. Қосылымдардың басым бөлігі жүйенің істен шығуына әкелмесе де, кейбіреулері қате енгізіліп жүйенің жалпы жұмысына әсер етеді.

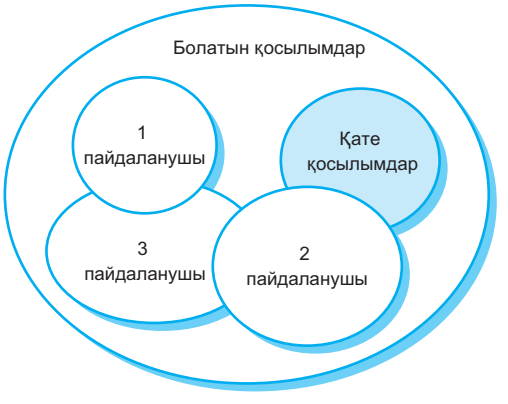


11.4-сурет. Қосылу мен ажыратудың жүйедегі көрінісі

Көптеген қосылулар бірнеше рет қайталанса да жүйенің ауытқуына соқтырады. Егер сирек пайдаланатын кодты қолдансақ, жүйедегі ауытқулар азаяды, өйткені бір адамға арналған сенімді жүйе екіншісінде қайталанбайды. Ал, көптеген қосылымдар болса, онда қатесі бар қосылымдармен түйісіп ауытқулар пайда болады. Дегенмен, *11.4-суреттің* I_e көлеңкеленген эллипсінде көрсетілген кейбір енгізулер немесе енгізулердің тәсілдері жүйенің жұмыс істемей қалуының немесе қате шығулардың себебі болды. Бағдарлама сенімділігі қате шығуларға алып келуі мүмкін енгізу деректерінің жиынтық мүшелері болып табылмайтын жүйе деректерінің санына байланысты болады. I_e жиынтықтағы енгізулер жүйенің жиі қолданылатын бөлшектерімен пайдаланылатын болса, онда жұмыс істемей қалулар жиі болып тұрады.

Дегенмен, I_e енгізулер сирек пайдаланылатын код арқылы орындалатын болса, онда пайдаланушылар жұмыс істемей қалуларға ұшырамайды. Себебі,

пайдаланушылардың әрқайсысы жүйені әр түрлі пайдаланады. Олардың сенімділік туралы түйсігі де әр түрлі. Бір пайдаланушының жүйесінде сенімділікке әсер ететін ақаулық болса, басқа адамның жұмыс режимінде ол анықталмауы мүмкін. (*11.5-сурет*) *11.5-суреттегі* жиынтық қате енгізулер *11.4-суреттегі* I_e жазуы бар эллипске сәйкес келеді. 2-пайдаланушымен жасалған жиынтық енгізулер көптеген қате енгізулермен қиылысуы мүмкін. Сондықтан 2-пайдаланушы жүйенің кейбір жұмыс істемей қалуларына ұшырауы мүмкін. Алайда 1-пайдаланушы мен 3 пайдаланушы ешқашан қате жиынтықтың енгізулерін пайдаланбайды. Ол үшін бағдарламалық қамтамасыз ету қашан да сенімді болады.



11.5-сурет. Бағдарламаны қолдану үлгісі

Іс жүзінде бағдарламаның сенімділігі қате ақпарат беретін көптеген қосылымдар мен пайдаланушыларға байланысты. Бір ғана бағдарламалық қамтудағы қателерді түзету жалпы жүйе жұмысын барынша жақсарта алмайды. Мысалы, Миллс (1987 ж.) белгілі қателердің 60 пайызын түзету барысында сенімділіктің 3 пайызға артқанын анықтаған. Желідегі ақаулар үнемі жалпы жүйедегі қателерді болдыра бермейді. Жүйедегі қателер де үнемі ауытқуларға әкеле бермейді. Оның себептері мынада:

1. Бағдарламадағы барлық кодтардың іске аспауы. Ауытқуы бар код бағдарламалық қамтуда болғандықтан, ешқашан қосылмайды. Қателердің ауысып-көшу қабілеті бар. Ол кодтың қате терілуінен қателердің ауысып келіп орын алуы мүмкін. Бірақ ол қатенің ауытқу тудырғанынша өзге қосылымдар оны ығыстырып, қатерлі жағдайды түзеуге мүмкіндігі бар.
2. Жүйе ақауларды анықтап, қорғаныс механизмін бақылай алады. Сондықтан қателерді бүкіл жүйе жұмысы бүлінбей тұрып анықтайды. Жүйедегі ауытқу тудырмайтын тағы бір себеп, пайдаланушылар өз тәжірибесінде қателі қосылымдардан алыс болуға бейімделеді.

Ақаулар, қателер мен ауытқулар арасындағы айырмашылық *11.3-суретте* көрсетілген. Соған қарап сенімділікті арттырудың үш бағытын айтуға болады.

1. *Ақаулардың алдын алу.* Жүйелерді істен шығаруға тигізетін адамның әсерін азайту тәсілін қалыптастыру. Мысалы, қатеге бейім тілдік құрылымдарды бағдарламаға енгізбеу.
2. *Ақауды анықтап, жөнге келтіру.* Верификация және валидация тәсілдерін қолдану арқылы жүйені пайдаланбай тұрып қателерді анықтау. Ақауларды анықтаудың бір көрінісі жүйелі түрде тестілеу жүргізу болып табылады.
3. *Істен шығуға қарсы тұрақтылық.* Бұл жүйедегі ауытқулар қателерге, ал қателер ауытқуларға айналмайтын жағдайды қалыптастырып, оған кепілдік беретін тәсіл. Өзіндік бақылау мен қосымша үлгілерді іске қосу жүйедегі істен шығуға қарсы тұрақтылықты сақтаудың мысалы бола алады.

Сенімді бағдарламалық қамту жасайтын бұл әдістер мен түрлі тәсілдерді қалай қолдану керектігі *13-тарауда* толығымен тарқатылады.

11.3. Қауіпсіздік

Қауіпсіздікке жоғары талаптары бар жүйелер – жүйенің жұмысы қауіпсіз болуы маңызды болатын жүйелерді білдіреді. Жүйе жұмыс істемей қалғанның өзінде адамдарға немесе қоршаған ортаға ешқандай зияндық алып келмеуі тиіс. Қауіпсіздікке жоғары талаптары бар жүйеге, мысалы, әуе кемелеріндегі басқару жүйесі мен мониторинг, химиялық және фармацевтикалық салалардағы технологиялық процестерді басқару жүйелері, сондай-ақ автомобильдердің қозғалысын басқару жүйелері кіреді.

Қауіпсіздікке жоғары талаптары бар жүйелерді аппараттық басқаруда бағдарламалық қамтамасыз етуді басқаруға қарағанда, енгізу мен талдау оңай. Дегенмен, қазір біз тек аппараттық басқарумен басқару мүмкін болмайтын күрделі жүйелерді құрып жатырмыз. Бағдарламалық қамсыздандыруды басқару – көптеген анықтағыштарды және басқаруы күрделі жетекті механизмдерді басқару қажеттігіне байланысты маңызды. Мысалы, жетілдірілген, аэродинамикалық тұрақты әскери ұшақтарға үнемі бағдарламалық қамсыздандыру құлауды болдырмай, ұшуды басқарудың беткі жағын реттеу үшін қажет.

Қауіпсіздікке жоғары талаптары бар бағдарламалық қамсыздандыру екі класқа бөлінеді.

1. Қауіпсіздікке жоғары талаптары бар алғашқы БҚ. Бұл жүйеде контроллер (бақылаушы) ретінде ішінен құрылған бағдарламалық қамсыздандыру. Бұндай бағдарламалық қамсыздандырудың жұмыс істемей қалуы адамдардың жарақаттануына немесе қоршаған ортаға зиянды келтіруге алып келеді. 1-тарауда аталған инсульді помпалар үшін БҚ қауіпсіздікке жоғары талаптар бар БҚ алғашқы жүйесінің мысалы болып табылады. Ондай жүйенің жұмыс істемей қалуы жарақаттануға алып келуі мүмкін.
2. Қауіпсіздікке жоғары талаптары бар екінші БҚ. Бұл жанама жарақаттануға алып келуі мүмкін бағдарламалық қамсыздандыру. Бұндай бағдарламалық

қамсыздандырудың мысалы жұмыс істемей қалуы жобаланатын объектінің жобалауына қате алып келуі мүмкін автоматтандырылған жобалау жүйесі болып табылады. Жобаланатын объектідегі бұндай қате адамдарға зияндық алып келуі мүмкін.

Қауіпсіздікке жоғары талаптар бар жүйенің тағы бір мысалы – психикалық денсаулықты қорғауды басқару жүйесі (MHC-PMS). Осы жүйенің жұмыс істемей қалуы денсаулық жағдайы тұрақты емес емделушінің тиісті және уақыттылы күтімді ала алмай қалғандығымен қоймай, өзіне және қоршаған ортаға зияндық алып келуі мүмкін.

Жүйенің сенімділігі мен қауіпсіздігі бір-біріне байланысты, бірақ сенімді жүйе қауіпсіз немесе керісінше қауіпті болуы мүмкін. Бағдарламалық қамсыздандыру жүйені пайдалану нәтижесінде жазатайым оқиғаға ұшыратуы мүмкін. Сенімді, бірақ қауіпсіз болып-болмауы міндетті емес БҚ жүйесінің 4 себебі бар.

1. Біз ешқашан БҚ жүйесінің тоқтаусыз және жұмыс істемей қалуға төзімді болатынына 100% сенімді бола алмаймыз. Дер кезінде анықталмаған ақаулықтар көпке дейін анықталмай, ұзақ тоқтаусыз жұмыстан кейін көп жылдардан кейін шығуы мүмкін.
2. Спецификацияның жоспары белгілі қиын жағдайда жүйені тиісінше пайдалану сипатын бермей, толық болмауы мүмкін (Жүйедегі жұмыс істемей қалулардың үлкен пайызы (Бем және басқалары, 1975 ж, Эндрес 1975, Лутц 1993 ж, Накайо және Куме 1991 ж) жүйедеге жобалаудың қателіктері болғаннан гөрі, айрықшалаудың нәтижесі болуы мүмкін ... талаптары бар күрделіліктер – жүйені тестілеу сәтіне және олардың құрауыштарының өзара әрекеттері дейін болған қауіпсіздікке байланысты БҚ-ғы қателіктердің негізгі себебі.
3. Апараттық қамсыздандыруда жұмыс істемей қалуға БҚ жүйесінің топшылауға болмайтын салдары мен болжанбайтын ортада БҚ қызметін көрсету болып табылады. Компоненттер нақты жұмыс істемей қалуға жақын болғанда қате көп шығып, БҚ бақылай алатын сигналдарды бере бастайды.
4. Жүйе операторлары өздігінен дұрыс емес болып табылмайтын, кейбір ситуацияларда жүйенің жұмыс істемеуіне алып келетін деректерді енгізе алады. Бірде әуе кемесінің шассиінде қызық жағдай болған көрінеді. Техник кемені басқару бойынша бағдарламалық қамсыздандыруға сигнал тетігін басып, шассиді көтерген. БҚ техниктің нұсқаулықтарын мінсіз орындаған. Дегенмен жүйе аталған команданы кеме ауаға көтерілмегенге дейін орындамайтындай етіп құрылуы тиіс еді.

Қауіпсіздікке жоғары талаптары бар жүйелерді талқылау үшін арнайы терминдер пайда болды. Қауіпсіздікке байланысты пайдаланылатын осындай терминдерді түсіну өте маңызды. *11.6-суретте* инсулиндік помпаларды басқару жүйелерінен алынған кейбір маңызды терминдердің мысалдары көрсетілген.

Термин	Анықтама
Жазатайым оқиға (немесе оқиға)	Жоспарланбаған оқиғалар немесе адамдардың жарақаттануына және қазасына, мүліктерге немесе қоршаған ортаға зияндық келтіруге алып келген оқиғалар тізбегі, Инсулинді мөлшерден тыс пайдалану осындай жазатайым оқиғаның себебі болуы мүмкін.
Қауіп	Жазатайым оқиғалардың шығу (туындауы) ықтималдығы бар шарт немесе осындай жазатайым оқиғаның туындауына әсер етуі мүмкін шарт. Қандағы қанттың деңгейін өлшейтін анықтағыштың жұмыс істемей қалуы қауіпке себеп болуы мүмкін.
Зияндық	Оқиға нәтежесінде пайда болған шығындарды өлшеу шаралары. Зияндық жазатайым оқиға нәтижесінде болған адамдар қазасы санынан бастап, мүліктің болмашы зияндығына дейін түрленіп отырады. Инсулинді мөлшерден тыс пайдалану - денсаулыққа айтарлықтай зиян келтіргеннен бастап, инсулинді помпаны пайдаланушы адамды өлу жағдайына дейін алып барады
Қауіптілік деңгейі	Белгілі қауіптіліктің болуы нәтижесінде келтірілуі мүмкін ең үлкен зияндықты бағалау. Қауіптілік дәрежесі көптеген адамдар қаза болуы мүмкін апаттылықтан бастап, болмашы зияндық келтіруге дейін түрленіп отырады. Қауіптілік дәрежесінің орынды бағасына сәйкес бір адам өлуі мүмкін жағдайда «қауіптілік өте жоғары» болып табылады
Қауіптілік ықтималдығы	Қауіптілікті туындатуы мүмкін оқиғалар ықтималдығы. Ықтималдық дәрежесі әдетте ерікті болады, бірақ «мүмкіннен», «мүмкін емеске дейін» түрленеді (мысалы, 100-ге 1 қауіптіліктің туындау мүмкіндігі) қауіптілік туындауы мүмкін ситуацияда ешқандай елестетін ситуациялар болмайды. Мөлшерден тыс болуы мүмкін инсулинді сорғы анықтағышының жұмыс істемей қалу ықтималдығы төмен болады.
Қауіп	Бұл жүйе қауіпке алып келуі мүмкін ықтималдығының шарасы. Қауіп қауіптілік ықтималдығын бағалау, қауіптілік дәрежесін, сондай-ақ, қауіптілік жазатайым оқиғаға алып келуі мүмкін ықтималдығы арқылы бағаланады. Инсулинді қайта мөлшерлеу қаупі орташадан төменге дейін болуы мүмкін.

11.6-сурет. «Қауіпсіздік» тақырыбы бойынша терминология

Қауіпсіздікті қамтамасыз ету кілті жазатайым оқиғалардың болмауын және олардың салдарының барынша аз болуын қамтамасыз ету болып табылады. Оған бір-бірін толықтыратын үш әдіспен қол жеткізуге болады.

1. Қауіптен аулақ болу. Жүйе қауіптен аулақ болатындай етіп құрылған. Мысалы, кескіш жүйеде оператор әр түрлі түймені қатар басып, екі қолын біруақытта пайдаланатындай етіп, құрылған. Бұл бір жағынан оператор қолының кесу траекториясында болмауға себеп.
2. Қауіптілікті анықтау және жою. Жүйе жазатайым оқиға туындағанға дейін қауіпті анықтап, жоюға болатындай етіп құрылған. Мысалы, химиялық зауытты басқару жүйесінде жоғары қысым анықталуы мүмкін. Осындай қысымды түсіріп, апаттың туындау ықтималдығын жою үшін клапанды ашу.
3. Зияндықты шектеу. Жүйе жазатайым оқиға нәтижесінде туындауы мүмкін зияндықты азайтатын қорғаныс функцияларын қосы алады. Мысалы, әуе кемесінің қозғалтқышы автоматты өрт сөндіргішті қоса алады. Өрт болған жағдайда оны ол әуе кемесіне тарағанға дейін жиі сөндіріп тұруға болады.

Жазатайым оқиғалар көп жағдайда біруақытта бірнеше дұрыс емес заттардың болғандығынан туындайды. Перроудың Қиын апаттарды талдауы (1984 ж) жүйенің әртүрлі бөлшектеріндегі жұмыс істемей қалулардың тұтас комбинацияларынан болатындығын көрсеткен. Кіші жүйелердегі жұмыс істемей қалулардың күтпеген комбинациялары өзара әрекеттесу жағдайына алып келіп, бүкіл жүйені жұмыс істетпей қойған. Мысалы, ауаны кондиционерлеу (желдету) жүйесіндегі жұмыс істемей қалу қызып кету жағдайына алып келіп, содан кейін ол аппараттық қамтамасыз етуге дұрыс емес сигналды берген. Перроу сондай-ақ, жұмыс істемей қалулардың барлық мүмкін комбинацияларын топшылап білу мүмкін емес дейді. Сондықтан оқиға күрделі жүйелерді пайдаланудың ажырамас бөлігі болып табылады.

Кейбір адамдар осы аргументті БҚ басқаруға қарсы пайдаланды. БҚ күрделілігіне байланысты жүйенің әр түрлі бөлшектері арасында үлкен өзара әрекеттер бар. Бұл өзара үлкен әрекет кезінде жүйенің жұмыс істемей қалуына алып келуі мүмкін жұмыс істемей қалу комбинацияларының көп болатындығын білдіреді.

БҚ арқылы бақыланатын жүйелер электрмеханикалық жүйелерге қарағанда шарттың үлкен жиынтығын басқара алады. Оларды салыстырмалы түрде жеңіл бейімдеуге болады. Олар бастан-ақ, сенімділігі жоғары, физикалық шағын және салмағы ауыр емес компьютердің аппараттық қамтамасыз етуін пайдаланады. БҚ пайдаланылатын жүйелер ақылды қорғаныс блокировкасын қамтамасыз ете алады. Олар қауіпті жағдайларда адамдарға қажет болатын, уақытта айтарлықтай азайтатын басқару стратегиясын қолдайды. БҚ басқару жүйе жұмыс істемей қалуы мүмкін үлкен ситуацияларды туындатса да, ол сондай-ақ, үздік бақылау мен қорғанысқа ие. Сондықтан жүйе қауіпсіздігін жақсартуға жақсы үлес қоса алады.

Барлық жағдайда жүйе қауіпсіздігі бойынша өлшемді сезу маңызды. 100 пайызға сенімді жүйені құру мүмкін емес және қоғам жазатайым оқиғалардың салдары озат технологияларды пайдаланудан шығатын игіліктерге тұрып, тұрмайтынын шешуі тиіс. Бұл сондай-ақ, барлық ұлт үшін қауіптерді азайту

мақсатында шектелген ұлттық ресурстарды пайдалану жоспарында қабылдануы тиіс әлеуметтік және саяси шешім.

11.4. Қорғаныс

Қорғаныс кездейсоқ немесе әдейі болатын өзін сыртқы шабуылдан қорғайтын жүйенің қабілетін көрсететін жүйе атрибуты болып табылады. Бұндай сыртқы шабуылдар болуы мүмкін. Себебі, жалпы пайдаланудағы көптеген компьютерлер желілерге қосылған және бөтен адамдарға қолайлы. Шабуыл ретінде вирустарды орнату мен троян аттары, жүйе сервистерінің санкциясын пайдалану немесе жүйелерге санкцияланбаған өзгерістерді енгізу немесе оның деректері. Егер Сіз шынымен жүйені қорғағыңыз келсе, онда оны желіге қоспағаныңыз дұрыс. Онда жүйенің қорғанысына байланысты проблемалар санкциясы бар пайдаланушылар жүйеге зияндық келтірмеумен шектеледі. Дегенмен, практика жүзінде желіге қосылу ірі жүйелер үшін орасан басымдық береді, сондықтан желіні Интернеттен ажыратып тастау экономикалық тиімсіз болып табылады.

Термин	Анықтама
Актив	Құндылығы бар және қорғанысты қажет етеді. БҚ жүйесінің өзі немесе осы жүйеде пайдаланылатын деректер актив бола алады.
Әсерге ұшырағыштық	Болуы мүмкін шығындар немесе компьютерлік жүйеге келтірілген зияндық. Бұл деректерді жоғалту немесе оған зияндық келтіру немесе қорғаныс бұзылғанан кейін деректерді қалпына келтіру қажет болса уақыт пен жұмсалған күшті жоғалту
Осал жері	Зиянды келтіру немесе жоғалту үшін пайдаланылуы мүмкін компьютерленген жүйенің осал жері
Шабуыл	Жүйенің осал жерлерін пайдалану, тұтас алғанда ол сырттан енгізіледі және жүйеге әдейі зияндықты келтіруді білдіреді.
Қауіп	жоғалтуды немесе зияндықты келтіру мүмкін жағдайлар. Сіз оны жүйенің осал жері, яғни әсерге ұшырағыштығы деп ойлап қалуыңыз мүмкін
Бақылау	Жүйенің осалдық дәрежесін түсіре алатын қорғаныс шарасы. Криптографиялық қорғаныс енуді бақылаудың әлсіз жүйесінің осалдық дәрежесін азайтатын бақылау мысалы болып табылады

11.7-сурет. «Қорғаныс» тақырыбы бойынша терминология

Кейбір жүйе үшін қорғаныс жүйе тәуелділігінің ең маңызды шарасы болып табылады. Әскери жүйелер, электрондық колммерция жүйелері, сондай-ақ өңдеуді және құпия ақпараттарды алмастыратын жүйелер қорғаныстың үлкен деңгейіне ие болатындай етіп жобалануы тиіс. Мысалы, авиабилеттерді брондау жүйесіне

ену мүмкін емес, бірақ оның қолайсыздық алып келіп, билеттерді беру уақытын кешіктіріп жататын кезі болып жатады. Ал жүйе қорғалмаған болса, бұзушы бүкіл бронды жойып жіберіп, барлық авиажелінің жұмысын тоқтатуы мүмкін.

Тәуелділігіне қарай басқа аспектілерде «Қорғаныс» түсінігіне байланысты арнайы терминдер бар. Пфлигермен (Пфлигер және Пфлигер 200 ж) сипатталған кейбір маңызды терминдердің анықтамасы *11.7-суретте, 11.8-суретте, 11.7-суретте* сипатталған қорғаныс концепциясы көрсетілген және психикалық денсаулықты қорғауды басқару жүйесінен алынған төмендегі сценарийлердің тиістілігі көрсетілген (MHC-PMS):

Клиника қызметкері пайдаланушының аты мен құпиясөзі арқылы MHC-PMS жүйесіне кіреді. Жүйе құпиясөз сөздің сегіз әріптен құралуын талап етеді, бірақ кез келген құпиясөзді тексерусіз орнатуға жол береді. Қылмыскер белгілі спортшының жүйкесіне байланысты ем алып жатқанын біліп, кейін жұлдызды бопсалау үшін ақпараттарға жасырын енуді алғысы келеді

Термин	Анықтама
Актив	Емделіп жатқан пациенттердің медициналық карточкасы немесе осы клиникада емделген пациенттердің медициналық карточкасы
Әсерге ұшырағыштық	Өзінің жеке деректерін сенгісі келмейтін болашақ пациенттердің әлеуеттік қаржылық шығыны. Спорт жұлдызының соттық шағымнан қаржылық зияндығы. Беделден айрылу
Осал жері	Пайдаланушыларға құпиясөздерді табуға мүмкіндік беретін құпиясөздердің әлсіз жүйесі. Пайдаланушылардың аттары
Шабуыл	Заңды пайдаланушы ретінде әрекет ететін бұзушының әрекеті
Қауіп	Санкцияланбаған (рұқсат берілмеген) пайдаланушы заңды пайдаланушының идентификациялық деректерін (логинді және құпиясөзді) табу арқылы жүйеге мүмкіндік алады.
Бақылау	Жеке аттары немесе сөздіктен табуға болатын пайдаланушыларға құпиясөздерін пайдалануға мүмкіндік бермейтін құпиясөздерді тексеру жүйесі

11.8-сурет. «Қорғаныс» тақырыбы бойынша терминдер мысалы

Қамқор туысқан болып кейіп танытып, жүйке ауруханасындағы медбикелермен сөйлесе отырып, олар арқылы жүйеге және жеке деректерге қалай енуге болатынын біледі. Бейджиктерге қарап, жүйеге енуге рұқсаты адамдардың атын анықтап алып, аттарды пайдалана отырып жүйеге кіруге тырысады.

Кез келген желілік жүйеде қорғанысқа қауіп төндіретін үш негізгі қауіп бар:

1. *Жүйе құпиялығының және оның деректерінің қаупі. Аталған қауіп осындай ақпараттарды алуға рұқсат берілмеген (санкцияланбаған) адамдарға немесе бағдарламаларға ақпараттарды ашуы мүмкін.*
2. *Жүйе тұтастығының және оның деректерінің қаупі.* Осындай қауіптер зияндық келтіріп, БҚ және оның деректеріне зияндық алып келуі мүмкін.
3. *Жүйеге және оның деректеріне Мүмкіндікті шектеу қаупі.* Бұндай қауіптер заңды пайдаланушылар үшін БҚ және оның деректеріне мүмкіндікті шектеу мүмкін.

Әрине, аталмыш қауіптер өзара байланысты. Шабуыл нәтижесінде жүйеге мүмкіндік шектеулі болса, уақыт ағымымен өзгеруі мүмкін ақпараттарды жаңарта аласыз. Бұл жүйе тұтастығының қауіпті жағдайда қалуын білдіреді. Шабуыл болып, жүйе тұтастығы қауіпте қалса, онда ол проблемаларды жою үшін демонтаждалған болуы тиіс. Сондықтан жүйеге мүмкіндік шектелуі мүмкін. Практика жүзінде әлеуметтік-техникалық жүйедегі ең осал жерлер техникалық ақаулықтар емес, адам факторы нәтижесінде пайда болуы мүмкін. Адамдар тез тауып алуға болатын құпия сөздерді таңдап, тауып алуға болатын жерлерге жазып қояды. Жүйе әкімшіліктері бақылау файлдарын немесе ену конфигурацияларын құру кезінде қателіктер жасап, пайдаланушылар қорғаныс БҚ орнатпайды және пайдаланбайды. Дегенмен, *10.5-бөлімде* айтылғандай, Пайдаланушы қатесі ретінде проблемаларды саралау кезінде біз өте сақ болуымыз қажет. Адам факторларына байланысты проблемалар көп жағдайда құпиясөздерді жиі ауыстыруды (пайдаланушылар өз құпиясөздерін қағазға жазып отырумен) немесе конфигурациялардың күрделі механизмдерін қажет ететін жүйелердің дұрыс емес жобалық шешімдерін көрсетеді.

Қорғанысты жақсарту үшін енгізе алатын бақылаулар сенімділік және қауіпсіздік үшін пайдаланылатындармен салыстырмалы.

1. Осалдықтан аулақ болу. Шабуылдардың сәтсіздігін қамтамасыз етуге бағытталған бақылаулар. Осы жерде қолданылуы тиіс стратегиялар қауіпсіздік проблемаларынан аулақ етуі тиіс. Мысалы, қауіпсіздікке жоғары талаптармен әскери жүйелер тыс байланыстыр мүмкін болу үшін жалпы пайдалану желілеріне қосылмаған. Сіз сондай-ақ, аулақ болуға негізделген бақылау ретінде криптографиялық қорғаныс туралы ойлауыңыз мүмкін. Шифрленген деректерге кез келген санкциясыз ену бұзушымен оқылмауы тиіс дегенді білдіреді. Практика жүзінде криптографиялық қорғанысты бұзу өте қымбат және уақытты көп алады.
2. Шабуылды анықтау және бейтараптандыру. Шабуылдарды анықтауға және бейтараптандыруға бағытталған бақылаулар. Осы бақылауларға жүйенің жұмысын бақылап, оның қызметінің қарапайым емес сызбасын тексеретін бақылаулар кіреді. Осындай қарапайым емес сызбаларды табу кезінде

қандай да бір шара қолданылуы мүмкін. Мысалы, жүйе бөлшегінің сөніп қалуы, бөлек пайдаланушылар үшін мүмкіндікті шектеу

3. Әсерді шектеу және жүйені қалпына келтіру. Проблемаларды жойғаннан кейін жүйені қалпына келтіре алатын бақылаулар. Онда бақылауларға автоматтандырылған резвертеу стратегиялары, ақпараттарды дубляждау, сондай-ақ, жүйені сәтті бұзуға байланысты болған шығыстарды өтейтін сақтандыру саясаты кіреді.

Орынды қорғаныс деңгейісіз жүйенің қолайлығына, сенімділігіне және қауіпсіздігіне сенімді бола алмаймыз. Қолайлылық, сенімділік және қауіпсіздікті қамтамасыз ету әдістері басында орнатылған жұмыс БҚ сондай БҚ болуы тиіс дегенді болжайды. Егер жүйе және БҚ бұзуға тартылса (егер БҚ өзгертілген болса, немесе оған құрт түсіп кетсе), онда жүйе сенімді және қауіпсіз бола алмайды.

Жүйе әзірлемесіндегі қателіктер қауіпсіздік жүйесінде іліктің пайда болуына алып келуі мүмкін. Егер жүйе күтпеген жерде енгізілген деректерге жауап бермейтін болса немесе массив шектері тексерілмейтін болса, онда бұзушылар жүйеге енуді үшін осал жерлерді пайдалануы мүмкін. Қауіпсіздікке байланысты ірі инциденттер – Бірінші интернет құрты (Спаффорд, 1989 ж.) және 10 жыл бұрын пайда болған «Code Red» құрты (Бергхель, 2001 ж.) осындай осал жерлерді пайдаланған. С бағдарламасына пассивтерін шекарасын тексеру кірмейді. Сондықтан жүйеге санкцияланбаған мүмкіндікті беретін кодпен бірге жадының бір бөлігін көшіріп беруі мүмкін.

НЕГІЗГІ ТҰЖЫРЫМДАР

- Компьютерлік жүйенің жұмыс істемей қалуы үлкен экономикалық шығындарға, ақпараттарды айтарлықтай жоғалтып алуға, физикалық зияндыққа немесе адамдар өміріне қауіп алып келуі мүмкін.
- Компьютерлік жүйенің тәуелділігі пайдаланушының жүйеге сену дәрежесін көрсететін жүйенің қасиетін білдіреді. Тәуелділіктің ең маңызды шаралары болуы, сенімділік, қауіпсіздік және қорғаныс болып табылады.
- Жүйенің болуы пайдаланушылардың сұрауы бойынша оларға қызмет ететін жүйені білдіреді. Сенімділік жүйенің қызметті бастапқы көрсетілген көлемде көрсететінін білдіреді.
- Қабылданатын сенімділік жұмыс кезінде пайда болуы мүмкін қателіктер ықтималдығына байланысты.
- Бағдарламада белгілі қателіктер болуы мүмкін, бірақ практика жүзінде олар өз пайдаланушылары үшін сенімді бола алмайды. Олар осындай қателіктері бар жүйе функцияларын ешқашан пайдалана алмауы мүмкін.
- Қауіпсіздік жүйесі адамдарға немесе қоршаған ортаға зияндық келтірмей, қалыпты жұмыс режимінде жүйенің жұмыс істеу қабілетін білдіретін жүйенің қасиетін білдіреді.

- Қорғаныс жүйесі өзінің сыртқа шабуылдардан қорғау қабілетін көрсетеді. Қорғаныс жүйесіндегі жұмыс істемей қалулар пайдаланушылар жұмысы үшін жүйенің барын жоғалтуға, жүйеге немесе оның деректеріне зияндықты келтіруге, немесе рұқсат етілмеген тұлғалардың пайдасына ақпараттардың шығып кетуіне алып келуі мүмкін.
- Қорғаныстың орынды деңгейін қамтамасыз етілмесе, жүйенің бары, сенімділігі мен қауіпсіздігі тыс жердегі шабуыл кезінде қауіпке ұшырауы мүмкін. Жүйенің қауіпсіздігіне немесе қорғанысын қамтамасыз ету қиын, себебі, олар жүйедегі жұмыс істемей қалуларға байланысты қауіптерге тартылуы мүмкін.

ҚОСЫМША ӘДЕБИЕТТЕР

'The evolution of information assurance'. An excellent article discussing the need to protect critical information in an organization from accidents and attacks. (R. Cummings, *IEEE Computer*, **35** (12), December 2002.) http://dx.doi.org/10.1109/MC.2002.1106181.

'Designing Safety Critical Computer Systems'. This is a good introduction to the field of safety-critical systems, which discusses the fundamental concepts of hazards and risks. More accessible than Dunn's book on safety-critical systems. (W. R. Dunn, *IEEE Computer,* **36** (11), November 2003.) http://dx.doi.org/10.1109/MC.2003.1244533.

Secrets and Lies: Digital Security in a Networked World. An excellent, very readable book on computer security which approaches the subject from a sociotechnical perspective. Schneier's columns on security issues in general (URL below) are also very good. (B. Schneier, John Wiley & Sons, 2004.) http://www.schneier.com/essays.html.

ЖАТТЫҒУЛАР

11.1. Бағдарламалық қамтамасыз етудің сенімділігі әлеуметтік-технологиялық жүйелердің көпшілігінде неге маңызды екенінің алты себебін көрсету.

11.2. Жүйенің сенімділігінің қандай аспектілері неғұрлым маңызды?

11.3. Неге сенімділікті қамтамасыз ету шығындары экспоненциалды ұлғайтады, мұнымен бірге, талаптар сенімділікті ұлғайтады?

11.4. Өз жауабыңызды негіздей отырып, сенімділіктің қандай атрибуттары келесі жүйелер үшін неғұрлым критикалы болып табылатынын көрсетіңіз:

- Мыңдаған клиенттермен ISP ұсынылатын интернет-сервер
- Қолжетімдігі төмен хирургиялық операцияларда қолданылатын компьютермен басқарылатын скальпельдер
- Жасанды жерсерік арқылы іске қосылатын аппараттың айналу бағытын бақылау жүйесі
- Интернет базасында жеке қаржыны жүргізу жүйесі.

11.5. Бағдарламалық қамтамасыз етудің қауіпсіздігі бойынша критикалық жүйесімен бақыланатын алты тұтынушылық тауарды анықтаңыз.

11.6. Сенімділік пен қауіпсіздік өзара байланысты болса да, олар сенімділіктің әр түрлі атрибуттары болып табылады. Бұл атрибуттардың арасындағы ең маңызды айырмашылықты сипаттаңыз, неге сенімді жүйе қауіпсіз немесе керісінше бола алмайтынын түсіндіріңіз.

11.7. Ісіктерді сәулелендіру жолымен емдейтін медициналық жүйеде туындауы мүмкін қауіпті атаңыз, мұндай қауіптердің апаттарға алып келуін болдырмайтын бағдарламалық қамтамасыз етудің бір қасиетін көрсетіңіз.

11.8. Компьютер қауіпсіздігін қамтамасыз ету тұрғысынан шабуыл мен қауіп арасындағы айырмашылықты түсіндіріңіз.

11.9. MHC-PMS көмегімен мысал ретінде жүйеге төнетін үш қатерді анықтаңыз (11.8-суретте көрсетілген қатерге қосымша). Осы қатерлерге негізделген шабуылдың ықтималдығын төмендету үшін белгілі түрде орналастырылатын басқару элементтерін ұсыныңыз.

11.10. Компьютерлік қауіпсіздік сарапшысы ретінде сіз азаптау құрбандары құқығын насихаттайтын ұйыммен келіссөздер жүргізе бастадыңыз және сізден ұйымға көмек көрсету ретінде американдық компанияның компьютерлер жүйесіне рұқсатсыз енуді өтінді. Бұл аталған компанияның саяси тұтқындарды азаптау құралдарын сататынын растауға немесе жоққа шығаруға көмектеседі. Осындай сұрақ көтеретін этикалық диллемаларды және мұндай сұранысқа қалай әрекет етуге болатынын талқылаңыз

ПАЙДАЛАНЫЛҒАН ӘДЕБИЕТТЕР ТІЗІМІ

Adams, E. N. (1984). 'Optimizing preventative service of software products'. *IBM J. Res & Dev.,* **28** (1), 2–14.

Berghel, H. (2001). 'The Code Red Worm'. *Comm. ACM,* **44** (12), 15–19.

Boehm, B. W., McClean, R. L. and Urfig, D. B. (1975). 'Some experience with automated aids to the design of large-scale reliable software'. *IEEE Trans. on Software Engineering.*, **SE-1** (1), 125–33.

Ellison, R., Linger, R., Lipson, H., Mead, N. and Moore, A. (2002). 'Foundations of Survivable Systems Engineering'. *Crosstalk: The Journal of Defense Software Engineering*, **12**, 10–15.

Ellison, R. J., Fisher, D. A., Linger, R. C., Lipson, H. F., Longstaff, T. A. and Mead, N. R. (1999a). 'Survivability: Protecting Your Critical Systems'. *IEEE Internet Computing*, **3** (6), 55–63.

Ellison, R. J., Linger, R. C., Longstaff, T. and Mead, N. R. (1999b). 'Survivable Network System Analysis: A Case Study'. *IEEE Software*, **16** (4), 70–7.

Endres, A. (1975). 'An analysis of errors and their causes in system programs'. *IEEE Trans. on Software Engineering.*, **SE-1** (2), 140–9.

Laprie, J.-C. (1995). 'Dependable Computing: Concepts, Limits, Challenges'. FTCS- 25: 25th IEEE Symposium on Fault-Tolerant Computing, Pasadena, Calif.: IEEE Press.

Littlewood, B. (1990). 'Software Reliability Growth Models'. In *Software Reliability Handbook*. Rook, P. (ed.). Amsterdam: Elsevier. 401–412.

Lutz, R. R. (1993). 'Analysing Software Requirements Errors in Safety-Critical Embedded Systems'. RE'93, San Diego, Calif: IEEE.

Mills, H. D., Dyer, M. and Linger, R. (1987). 'Cleanroom Software Engineering'. *IEEE Software*, **4** (5), 19–25.

Nakajo, T. and Kume, H. (1991). 'A Case History Analysis of Software Error-Cause Relationships'. *IEEE Trans. on Software Eng.*, **18** (8), 830–8.

Perrow, C. (1984). *Normal Accidents: Living with High-Risk Technology*. New York: Basic Books.

Pfleeger, C. P. and Pfleeger, S. L. (2007). *Security in Computing, 4th edition*. Boston: Addison-Wesley.

Spafford, E. (1989). 'The Internet Worm: Crisis and Aftermath'. *Comm. ACM*, **32** (6), 678–87.

12.

Сенімділік және қауіпсіздіктің талаптары

Мақсаттары

Бұл тараудың мақсаты – функционалды және функционалды емес сенімділікті және қауіпсіздіктің талаптарын қалай анықтау керек екенін түсіндіру. Осы тарауды оқып шыққаннан кейін, сіз:

- қауіпсіздікді сақтап, сенімділік және қауіпсіздік талаптарын орындауда бірдейлестіру мен анализ жасауда басқарылатын кауіп-қатерді қалай пайдалану керек екенін;
- қауіпсіздік анализін жасауда және қауіпсіздік техникасының талаптарын қолдануда «бас тартудың тал-шыбықтарын» қалай пайдалану керек екенін;
- сенімділік маманданымының көрсеткіштерімен танысып және сенімділіктің өлшемді талаптарын анықтауда олардың қалай пайдаланатынын;
- күрделі жүйеде қажет болатын қауіпсіздік талабының түрлі үлгілерін;
- жүйенің формалдық, математикалық техникалық талабының игерушілігінің артықшылығы мен ыңғайсыздықтарын білетін боласыз.

Мазмұны

1993 жылы қыркүйекте ұшақ Польшаның Варшава әуежайында найзағай ойнап тұрғанда қонды. Қонғаннан кейін тоғыз секунд бойы компьютермен жұмыс істейтін тежеуіш жүйесінің тежеуіштері жұмыс істемей қалды. Тежеуіштің жүйесі ұшақ жерге қонғанын түсінбеді де, қауіпсіздік жүйесінің жабдығы ұшақта қарама-қарсы әуестік жүйесінің ұңғылауын тоқтатқан, себебі ұшақ ауада болса, ол өте қауіпті жағдай болып есептеледі. Ұшақ көтермеге соғылып, тұтас жанып кетті.

Бақытсыз уақиғаны тергеу жүргізу кезінде белгілі болған жағдай, тежеуіш жүйесінің бағдарламалық қамтамасыздандыруы оның маманданымына сәйкес жұмыс істеген. Бағдарламада ешқандай қате болмаған. Алайда, маманданым бағдарламалық қамтамасыздандыру толық болмаған себептен, жоғарыда айтылған жағдай орын алған. Бағдарламалық қамтамасыздандыру жұмыс істеді, бірақ жүйе тақырға отырғызды.

Бұл жағдай жүйелі сенімділік тек жақсы зерттемеге тәуелді болатынын ашық көркемдейді. Сол үшін жүйелі талаптарға сүйеніп, бөлшектерге көңіл қойып, тиянақты зерттеу керек, әсіресе, бағдарламалық қамтамасыздандыруға деген сенімділік және жүйенің қауіпсіздігін қамсыздандыруға оқталған арнаулы талап қосылуы керек. Сенімділік және қауіпсіздік талаптарының екі түрлі үлгісі болады:

1. Функционалдық талаптар. Тексеру және қалпына келтіру үшін анықтайтын құрал-жабдықтарға қойылатын талаптар, ол талаптар жүйенің ішіне кіру керек және дербес өзгешеліктерімен кіру керек, тек сонда ғана жүйелі қатеге қарсы тұра аламыз және сыртқы факторлардан қорғаныс болады.
2. Функционалды емес талаптар. Сенімділік пен жүйенің қолжетімділігін анықтайтын талаптар.

Функционалдық сенімділік және қауіпсіздік талабының өндірісі үшін ең бастапқы нүкте – жиі немесе жоғары деңгейдегі іскер домен жөн-жосық, саясат немесе ереже. Биік деңгейдің осы талаптары, абзалы «жарамайтын» талаптар болатын сияқты. Керісінше, бірқалыпты функционалдық талаптармен сәйкес «жарамайтын» талаптар жүйенің мінез-құлқына килігіп кетеді. «Жарамсыз» талаптардың мысалдары:

«Жүйе пайдаланушыларға өздері жасамаған файлдарға өзгерту енгізуге рұқсат беру құқықтарын жасамау керек” (Қауіпсіздік).

«Ұшақ ұшып келе жатқанда жүйе қарама-қарсы әуестіктің режимін қоспауы керек» (Қауіпсіздік).

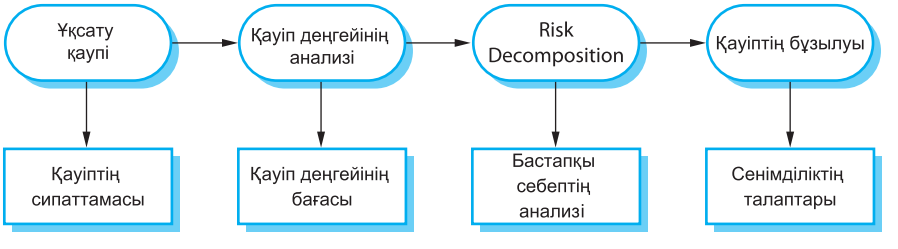
«Жүйе алаңдаушылықты белсендіруде үштен артық дабыл бермеу керек» (Қауіпсіздік).

Осы 'жарамсыз' талаптарды жүзеге асырудан бұрын, бағдарламалық қамсыздандырудың нақты функционалдық талаптарымен қамтамасыз етіліп

жинақталуы керек. Олар бас негіз жүйені жобалау кезінде нақты үлгімен, қандай құрал-саймандармен жүзеге асырылатынын болжай білу керек сияқты.

12.1 Қатер-қауіп басқаруында қойылатын талаптардың мамандандымдары

Сенімділік және қауіпсіздіктің талаптары қорғаныс сақтаудың негізгі талаптары болып қарастырылады. Олар жүйе өзін қалай ішкі ақаулардан қорғау керек екенін, жүйенің ортаға зиян келтіруін тоқтатып, бақытсыз уақиғаны немесе қоршаған ортаның жүйеге зиянды шабуыл жасауын болдырмауды және бұзылған жүйені жөндеуді жеңілдетуін айқындайды. Осы қорғаныстық талаптарды кездестіру үшін, сендерге жүйеде қауіп-қатердің туындауы ықтимал екенін түсіну қажет. Мамандандым талаптарының қауіп-қатер себептерінің оң шешімін табуда ескеретін жайлар: қауіпті уақиғалар болуы мүмкін екенін ескеру, ол қауіпті уақиғалар бұзылуларға әкеліп соғуы мүмкін екенін ескеру, келтірілген зияннның дәрежесін ескеру. Қауіпсіздік және талаптың сенімділігі қауіпті уақиғаның ықтимал себебінің анализінің негізінде тағйындалады. Әзірлеушілер қауіпсіздік негізіндегі мамандандымдары қауіпсіздіктің әдеттегі жүйесіне және қысылшаң маңыздылығына сүйенеді. Тіл табуға тәуекелділік көп шығынның шығуына сай немесе көбірек кездесетін уақиғаларға сай қабылданады. Жартымсыз салдар ғана немесе төтенше сирек кездесетін уақиғалар елеп-ескерілмеуі мүмкін. Қауіпсіздіктің қысылшаң маңызды жүйелерінде бақытсыз уақиғаларға әкеліп соғатын қауіптер; қысылшаң маңызды жүйелерде қауіп-қатер ішкі және сыртқы шабуылдарға әкеліп соғады да, осалдық мүмкіндікті пайдалану амалы туындайды.



12.1-сурет. Тәуекелмен әзірлеудің сипаттамасы

Қауіп-қатерге негізделген тіл табудың ортақ үдерісі (*12.1-сурет*) жүйенің болжамды-тәуекелдік түсінушілігіне душар болуы себепті, осы қауіп-қатерді басқаруға жаралған талапты құру. Аталмыш үдерістің кезеңдері:

1. *Тәуекелді бірдейлестіру.* Әлеуетті тәуекелдер жүйе үшін бірдейлестірілген. Олар жүйе пайдаланылатын ортаға байланысты болады. Қауіп-қатер жүйенің әрекеттестігінің нәтижесі мен сирек шарттардың арасында болатын операциялық ортаға байланысты болады. Мен басында талқылап кет-

кен Варшавада болған бақытсыз уақиға, найзағай жасаған қарсы желдер барысында ұшақты еңкейтіп, екі емес бір доңғалаққа қонуға мәжбүр етті.

2. *Анализ және қауіп-қатерді топтастыру.* Әр қауіп-қатерді оқшау қарастырамыз. Әлеуетте салмақты, сондықтан нанымсыз болып табылатындары одан әрі анализ үшін сұрыпталады. Айтылмыш кезеңінде қауіп-қатерлер серпіліп жойылады, өйткені олар қайта тууы мүмкін емес, себебі олар бағдарламалық қамсыздандырумен анықталмайды (айталық, инсулин сорабының жүйесінде бергішке деген аллергиялық реакция).
3. *Тәуекелдің құрамы.* Тәуекелдің әлеуетті себеп-салдарын табу үшін, әр тәуекел жеке талданып, оған анализ жасалады. Себеп-салдар – жүйенің сәтсіздікке ұшырау себептері. Олар бағдарламалық қамсыздандыруда қате кеткеннен немесе аппаратты айыпұлдың қателерінен немесе жүйенің құрылымында өзіне лайық осалдық кеткеннен болады.
4. *Қауіп-қатерді төмендету.* Бірдейлескен қауіп-қатерді төмендету немесе жою жөнінде ұсыныс-пікір енгізілген, Олар қауіп-қатерге қарсы қорғаныс қабілетін анықтайтын жүйенің сенімділік талаптарына жағдай жасап, осы тәуекелді басқару болады.

Үлкен жүйелер үшін тәуекел дәрежесінің анализі құрылым сатыларынан тұрады (Левесон, 1995), онда әр кезеңі тәуекелдің әртүрлі үлгілерін қарастырады:

1. Тәуекел дәрежесінің алдын ала шамалау анализінде, жүйе ортасындағы басты қауіп-қатерді табуға болады. Олар жүйелі даму үшін пайдаланылатын технологияға тәуелсіз. Тәуекел дәрежесінің алдын ала шамалау анализінің мақсаты – қауіпсіздіктің және сенімділіктің талабының алғы терімін жүйе үшін дамыту.
2. Жүйені зерттеу барысында туындайтын өмірлік топтаманың тәуекел дәрежесінің анализі негізгі жобалықтың шешім жүйесінде туындайды. Әртүрлі технология және жүйелі сәулеттердің өзінің жекеменшікті ерекше қауіп-қатерлері болады. Айтылмыш кезеңде қауіп-қатерден сақтану үшін сіз талаптарды кеңейтуіңіз керек.
3. Жүйелі қолданбалы интерфейстің оператордың қателесуінен туындайтын қауіп-қатерлер тәуекелдің дәрежесінің пайдалануын талдауға байланысты болады. Интерфейстің қолданбалы дизайнында бекітім қабылданғаннан кейін, одан әрі талаптар үстеу болу үшін қосымша қорғаныс талаптарын қосу керек.

Бұл фазалар өте қажет, себебі жүйелі енгізуде толық ақпаратсыз сенімділік пен қауіпсіздік туралы шешім қабылдау мүмкін емес. Қауіпсіздік және сенімділіктің талаптары технологиялық талғам және жобалық танымдарының айрықша мәселесін көтереді. Жүйелі тексерістерге, мүмкін, жан-жақты компоненттер дұрыс жұмыс істеуіне кепілдік болу үшін де қажет. Ал қауіпсіздіктің талаптарын, керісінше, өзгертуге тура келеді, себебі олар стандартты жүйемен қамсызданып, қауіпсіздіктің тетіктерімен қайшылықта болады.

Қауіпсіздік менеджментіне арналған IEC стандарты

IEC (Халықаралық электрохникалық комиссия) комиссиясы қорғаныс жүйелеріне арналған қауіпсіздік менеджментінің стандартын (мысалы, қауіпті жағдай туған кезде, іске қосу қорғанысына арналған жүйелерін) белгілеп шығарды. Қорғаныс жүйесінің мысалы қызыл сигнал арқылы өткенде пойызды автоматты түрде тоқтататын жүйе болып табылады. Бұл стандарттың құрамына қауіпсіздік спецификациясының үдерісіндегі жан-жақты нұсқаулық кіреді.

http://www.SoftwareEngineering-9.com/Web/SafetyLifeCycle/

Айталық, қауіпсіздіктің талабына сай пайдаланушылар өзін жүйеге сәйкестендіруі керек, құпия сөз арқылы емес түйінді мәселе сөз арқылы белгіні пайдалана білу қажет. Құпия сөздерге қарағанда, шартты белгінің түйінді сөздері қауіпсіздеу деп саналады. Олар зиянкестер үшін автоматталған іріктеу немесе шартты белгінің құпия сөзін болжалап ойлап тапқанға қарағанда күрделі. Алайда, егер нағыз жүйелі тексерісте шартты белгілерді құпия сөз арқылы пайдалану негізделген болса, сол қауіпсіздіктің айтылмыш талабы қуатталмайды. Ол жүйенің қосымша атқаратын қызметін қосуы үшін қажетті, өйткені шартты белгіні түйінді сөз арқылы емес құпия сөз арқылы пайдалана отырып, өсіңкіреген қауіп-қатерді теңестіруге болады.

12.2. Қауіпсіздіктің талаптары

Қауіпсіздік сынағынан өткен жүйелерде жүйенің қоршаған ортасын бұзып және адамдар жарақат алып немесе адамның ажалына әкеліп соғатын жағдайлар тууы мүмкін. Айтылмыш ортада қауіпсіздіктің талабының негізгі мақсаты талапты бірдейлестіруінің мынадай жүйелі бұзылудың мүмкіндігін ең төмен деңгейде ұстау. Қауіпсіздіктің техникалық талаптары ең алдымен, қауіпсіздік талаптары және олардың әдеттегі жүйелі операцияларға ешқандай қатысы жоқ. Олар жүйе жабық болған кезде де қауіпсіздік сақталу керек екенін анықтай алады. Қауіпсіздік талабының тұжырымында, қауіпсіздік пен функционалдықтың арасында теңгерім болуға тиіс және әдеттен, тыс қауіпсіздік болмау керек.

Өте қауіпсіз жүйенің құрылысында егер ол тиімді жұмыс істемесе, ешқандай мағына жоқ. *10-тараудағы* талқылаудан білетініміз қауіпсіздікке қысылшаң жүйелер мамандандырылған терминологияны пайдаланады, онда қауіп-қатер нәтижесінде бірнәрсе бұзылып, оның нәтижесінде адамдар жарақаттанып не болмаса ажалға ұшырап жатса, жүйе қауіпті күйге кіреді. Сол себептен қауіпсіздіктің маманданымы әдетте, арада туатын қауіптерде немесе сол қауіпке апарып соғатын уақиғаларда топтанады.

Әрекеттер *12.1-суретте* тәуекелдің сарапшылығында жалпы негізделген, қауіпсіздіктің маманданымының үдерісінің картасына келесі бейнемен безейді:

1. *Тәуекелді бірдейлестіру.* Қауіпсіздіктің маманданымында, ол – жүйеге қауіп-қатер төнгенде бірдейлестіру барысында қауіптің сәйкестендірме үдерісі.
2. *Тәуекел дәрежесінің анализі*. Ол – қауіптің бағалау үдерісі, олардың қайсысы ең қауіпті немесе ең ықтимал болып табылатынын бағалайды. Олар қауіпсіздіктің техникасының талаптарына сай басымдыққа қарай орналыстырылу керек.
3. *Қауіп-қатердің құрамы*. Ол үдеріс қауіпті уақиғаны табуға тиімді. Қауіпсіздіктің талаптарында, айтылмыш үдеріс «тәуекелдің анализі» атымен белгілі.
4. *Қауіп-қатердің төмендету*. Бұл үдеріс тәуекелдің анализінің нәтижесінде негізделген және қауіпсіздікт техникасының талабының бірдейлестіруіне келтіреді. Олар қамсыздандыруға тиесілі болуы мүмкін, қауіп туғызатын немесе бақытсыз уақиғаға келтіретін жағдай туса, жүйенің бұзылуы ең кем мөлшерде болады.

12.2.1. Қауіпті бірдейлестіру

Қауіпсіздікке қысылшаң негізгделген жүйелерде негізгі қауіп-қатерлер апатқа ұшырау қатерінен туындайды, Қатердің түрлі үлгілерін, мысалы, физикалық, биологиялық, радиоактивті электр қауіп тудыратын және серверден қате кету қаупін қарастыра отырып, бұдан әрі сендер сәйкестендірме мәселемен шұғылдана аласыңдар. Қауіп тудыратын қолайсыз жағдайларды анықтап, табу үшін, осы топтардың әрқайсысы жеке талдаудан өтеді. Айбаттың барлық ықтимал әрекет-амалдары сәйкестендіруден өтуі қажет.

Мен жоғарыда мысал ретінде айтып кеткен инсулин сорабының жүйесі қысылшаң қауіпсіздік жүйе болып табылады, себебі, сәтсіздік бұл арада жүйелі пайдаланушыға жарақат немесе тіпті, ажалға әкеліп соқтыруы мүмкін. Осы машинаны пайдалану барысында болатын бақытсыз уақиғалар, қан құрамындағы қант мөлшерін қадағалайтын пайдаланушыларға кері әсерін тидіруі мүмкін (көзге, жүрекке және бүйректерге); қателіктер пайдаланушының дене және тән қасиеттерін күйзеліске ұшыратып, баска аурулардың пайда болуына әсер етіп, аллергиялық реакцияға әкеліп соғуы мүмкін.

Инсулин сорабының жүйесіндегі кейбір қауіпті жағдайлар:

- инсулиннің аса көп мөлшерде есептелуі (сервистік қате);
- инсулиннің жеткіліксіз мөлшерде есептелуі (сервистік қате);
- мониторинг жүйесінің аппаратты құрал-саймандарының жұмыс істемей қалуы (сервисті қате);
- батареяның бәсеңдеп, қуат көзінің таусылуы (қоректенудің қатесі);

- басқа медициналық жабдықтардың электрлік кедергілерінің әсері, мысалы, кардиостимулятор (қоректенудің қатесі);
- қондырғыны дұрыс орнатпағаннан болатын бергіштің және жетектің жаман байланысы (физикалық қате);
- емделушінің денесінде аппарат бөлігі жұмысының үзіле беруі (физикалық қате);
- аппараттың дәрі-дәрмекті егу кезінде пайда болатын инфекциялар (биологиялық қате);
- аппаратта қолданылатын материалға немесе инсулинға деген аллергиялы реакциялар (биологиялық қате).

Қауіпсіздіктің техникасының сарапшы және кәсіби кеңесшілерімен шұғылданушы тәжірибелі инженерлер қауіптерді тәжірибенің алғашқы бас негізінде және қолданбалы сала анализін пайдалана отырып, бірдейлестіреді. Идеялармен пайдалануды білетін адамдар тобы «ми шабуылы» атты жұмыстың әдістерін пайдаланып, мақсаттарын жүзеге асырады. Мысалы, инсулиннің сорабының жүйесі үшін дәрігер, медициналық физиктарды, тіпті, инженер және бағдарламалық қамсыздандырудың әзірлеушілерінің басын қосуға тура келді.

Бағдарламалық қамсыздандыру мен тоқулы қауіптер әрдайым жүйелі операцияны орындауынан немесе мониторингтің және қауіпсіздік жүйесінің бас тартуынан болады. Айталық, батарея зарядтарының деңгейлерін өлшеу үшін және ерекше шарттарды білу үшін мониторинг және сақтандыру жүйелері құрылымның ішіне орналастырылған.

12.2.2. Қауіптің сарапшылығы

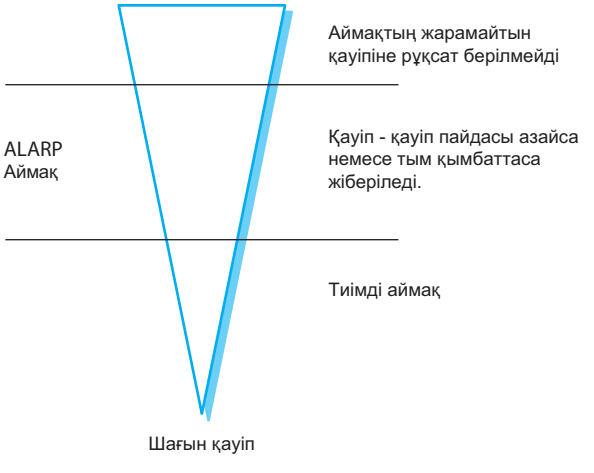
Қауіптің сарапшылығының үдерісі бір орынға жинастырылу түсінушілігінде, қауіп-қатер туу мүмкіндігі апат немесе қақтығыс зардап салдарына байланысты. Қауіп-қатер жүйеге немесе қоршаған ортасына қаншалықты қиын әсер ететінін түсіну үшін сендер бұл анализді жасауларың керек. Бұл анализ түпті әдістің талғамы үшін басқарма қауіппен байланысты тәуекелмен қамсыздандырылған.

Қауіпті жағдайда анализдің және топтастырудың нәтижесі қабылдауға болатын бекітім болып табылады. Тәуекел бұзылу салдарының мүмкіндігін ескеретінін біз тәуекелдің терминдерінен біле аламыз. Қауіп-қатердің үш санаты болады, біз оларды қауіптің сарапшылығын да пайдалана аламыз:

1. *Адамның өміріне қауіп төндіретін, адам төзгісіз қауіп-қатерлер* (қауіпсіздікке қысылшаң жүйелерде). Жүйені дайындаған кезде ешқандай қауіп-қатер тудырмайтын жағдайлар жасалып, ал қауіптер туған күнде де адам өміріне зиянын тигізбейтіндей қылып әзірлеу керек. Егер инсулин сорабының мәселесін алатын болсақ, инсулин мөлшері шамадан тым артық кетсе, адам төзгісіз қауіп-қатер болады.

2. *Іс жүзінде тәуекелдің тиімді деңгейі (ППУ)-неғұрлым азырақ зардап салдарға соқтыратын қауіп-қатерлер*. Жүйені әзірлеген кезде, бақытсыз уақиғаны болдырмау мүмкіндігін, әрине, бағасы мен жеткізудегі шығындарды ескере отырып, ұлғайту керек. Тәуекелдің ППУ инсулинның сорабы үшін – аппаратты құрал-саймандар мониторингі жүйесінің бұзылуы. Оның зардабы – дозаның қысқа мерзімді артық берілуі. Мұндай жағдай бақытсыз уақиғаға келтірмейді.
3. *Қабылдауға болатын қауіп-қатерлер, сәтсіз уақиғалар аздаған бұзылуға әкеліп соғады*. Тәуекелдің мүмкіндігін төмендету үшін жүйелі жобалаушылар шығынды, жеткізудің уақытын немесе функциялық емес жүйелі белгілерінің барлық ықтимал қадамдарын алқындырулары керек. Егер инсулинның сорабын мысал ретінде қарастырсақ, қолайлы тәуекел – бас пайдаланушының аллергиялы реакцияға ұшырау қауіпі. Негізінде ол адамның аз ғана денесі тітіркеніп, түршігуі мүмкін, ендеше бұл қауіп-қатерді төмендету үшін аса қымбат материалдар мен қымбат бағалы құрылымдарды қолданудың қажеті жоқ.

12.2-сурет (Брэзендэйл және Белл, 1994). Қауіпсіздікке қысылшаң жүйелер үшін әзірленген осы сурет үш саланы көркемдейді. Диаграмманың пішіні бұл тәуекелдің қамсыздандыруының шығындары қақтығыс немесе бақытсыз уақиғаларға келтірмейді дегенді айқындайды. Қауіп-қатермен күресу үшін жасалған жүйелі жобаның құны үшбұрыштың енімен таңбаланған. Қауіп-қатердің ең жоғарғы шығындары диаграмманың шыңында, ең аз (төмен) шығын үшбұрыштың шыңында көрсетілген.



12.2-сурет. Тәуекел үшбұрышы

Сала шекараларының арасы *(12.2-суретте)* техникалық емес, әлеуметтік және саяси факторлар тәуелді болады. Ұзақ уақыт бойы қоғам тәуекелге деген пейілді төмендетіп, ақырында, шекаралар төмен ауыстырылды. Тәуекелді қабылдаудың қаржы шығындары жоспарлағаннан едәуір төменірек болу керек еді, бірақ,

қоғамдық пікір бөлінген қаражаттың толық қамтылуын талап етуі мүмкін, өйткені жүйелі апаттың мүмкіндігін төмендету қосымша қаражат талап етеді. Мысалы, компанияларға ластануға қарсы толық бағалы жүйені қондырғанша, ластанудың зардабын жою әлдеқайда тиімдірек (өмірде аз кездесетін жағдай). Алайда, жұрттың СМИ қабаттасушылығына байланысты, мұндай шешім нұсқа болып табылмайды. Мынадай уақиғалар тәуекелдің топтастыруына әкеліп соғады.

Қауіптілікті бағалау қауіптіліктің мүмкін екенін бағалауға әкеледі. Мұндай окиға өте сирек кездесетін жағдай болғандықтан, бұл өте қиын үдеріс болып есептеледі, демек, инженерлерде олармен күресудің тәжірибесі болмауы мүмкін. “Ықтимал”, “күмәнді”, “сирек”, “биік”, ”орташа”, ”аласа” атты арнаулы терминдерді пайдалана отырып, айбаттың мүмкіндігінің арнаулы топтастырулары бар екенін білеміз. Статистикалық анализ жасау үшін бақытсыз уақиға немесе қақтығыс жайлы жеткілікті мағлұматтар болса ғана барлық шарттарды орындап, анықтауға болады.

Анықталған қауіптілік	Қауіптілік ықтималдылығы	Сәтсіздіктің қаталдылығы	Есептелген тәуекел	Қабылдану
Инсулин дозасының артылуы	Орташа	Жоғары	Жоғары	Төзімді емес
Инсулин дозасының кемуі	Орташа	Төмен	Төмен	Жарамды
Техникалық қамсыздандыруды бақылайтын жүйенің сәтсіздігі	Орташа	Орташа	Төмен	Дыбыс
Қуаттың сәтсіздігі	Жоғары	Төмен	Төмен	Жарамды
Есептеу машинасы қате орнатылуы	Жоғары	Жоғары	Жоғары	Төзімді емес
Есептеу машинасының ақырын бұзылуы	Төмен	Жоғары	Орташа	Дыбыс
Жұқпалы аурулардың машина арқылы таралуы	Орташа	Орташа	Орташа	Дыбыс
Электрикалық араласу	Төмен	Жоғары	Орташа	Дыбыс
Аллергиялық реакция	Төмен	Төмен	Төмен	Жарамды

12.3-сурет. Инсулин сорғышының тәуекелдерінің жіктелуі

12.3-сурет алдыңғы бөлімде бірдейлестірілген инсулин жеткізу жүйесіндегі қауіп тәуелділігінің топтауын көрсетеді. Мен инсулинді дұрыс есеппен мөлшерлемейтін (аз мөлшерде немесе өте кем мөлшерде) қауіпті бір бірінен ажыраттым. Инсулиннің кем құйылғанға қарағанда артық кетуі әлдеқайда қауіпті. Инсулиннің көп мөлшерде кетуі танымдық дисфункцияға, есінен тануға, ақыр соңында ажал әкеледі. Инсулин кем мөлшерде түссе, қанда қант деңгейі жоғарылап кетеді. Қысқа мерзімде адамды шаршатып тұрады, бірақ ол жағдай өте қауіпті болып есептелмейді, ал егер ұзақ мерзімге созылса, онда жүректің, бүйректің және қатерлі көз ауруларына әкеліп соғады.

12.3-суреттегі 4-9 қауіптер бағдарламалық қамсыздандырумен байланысты болып табылмайды, бірақ бағдарламалық қамсыздандыру қауіпті жағдайды анықтап табуға міндетті түрде қатысуы керек. Аппаратты құрал-саймандарының бағдарламалық қамсыздандырудың бақылау жүйесі әр уақытта тексеріп және әлеуетті мәселелер жайлы ескертіп отыруға тиіс. Ескерту бақытсыз уақиға туындамай тұрып, қауіпті жағдайды дер кезінде анықтауға мүмкіндік береді. Электрдің қуат көзінің кідірісі (аккумулятор батареяларын мониторинг жасағанда анықталады) – аппаратты дұрыс қондырмаудың (қандағы қант мөлшерін өлшейтін құрылғыға мониторинг жасағанда анықталады) себебі болып табылатын қауіп-қатерлердің мысалы болып табылады.

Жүйедегі мониторинг үшін бағдарламалық қамсыздандыру, әрине, қауіпсіздікпен байланысты. Қауіпті жағдайды табуда кеткен қате апатқа әкеліп соғуы мүмкін. Мониторинг жүйесінің бұзылуы болып жатып, бірақ жабдықтар дұрыс жұмыс істеп жатса, онда бұл жеңіл-желпі бұзылу. Алайда, өлшеушінің жүйесі бұзылып, іркіліс берсе және ол бұзылыс табылмай жатса, ол жағдайда зардап ауыр болады.

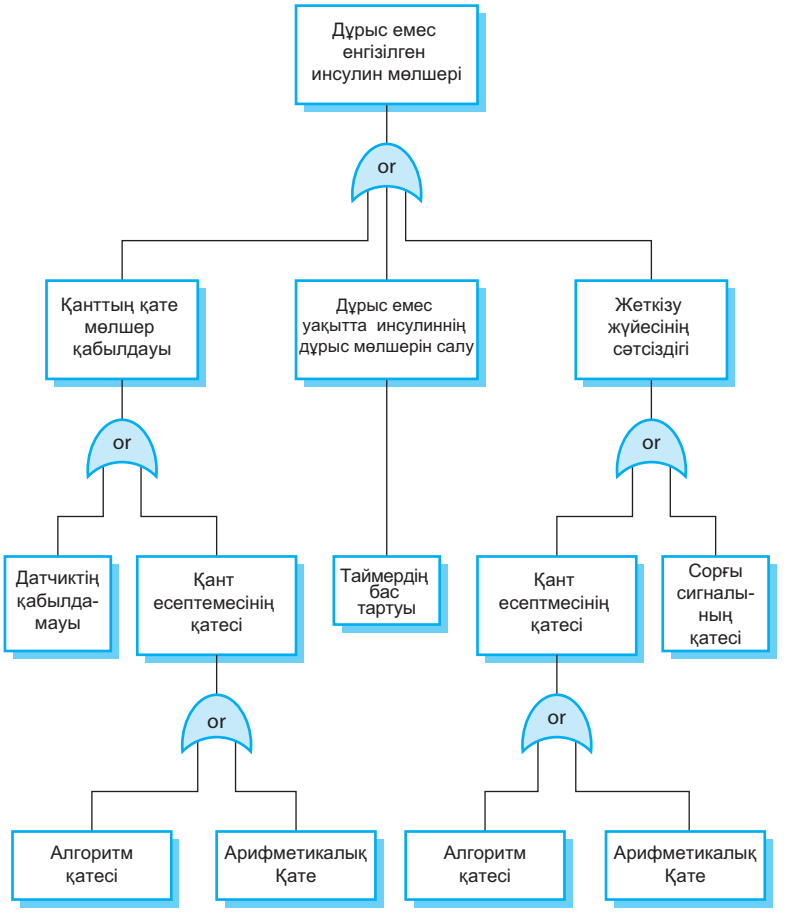
12.2.3. Қауіптің анализі

Қауіптің анализі – қауіпсіздікке қысылшаң жүйеде қауіптің басқа себептерін кездестіру үдерісі. Сендердің мақсаттарың – қандай уақиғалар немесе уақиғаның әрекеті жүйелі қауіп ақауына әкеліп соқтыратынын білу. Оны жүзеге асыру үшін, сіз анализдің «бәсеңдейтін», немесе «өрлемелі» әдісін пайдалануыңыз керек. Дедукциялық, («бәсеңдеу») әдістері неғұрлым ыңғайлырақ болып табылады, олар қауіптен басталады және жүйенің ықтимал ақауларына дейін өңдейді. Индуктивті («өрлемелі») әдістер ұсынылған жүйелі ақаулардан бастайды да, сәтсіздіктен кейін қандай қауіптер болуы мүмкін екенін бірдейлестіреді.

Ажыратудың немесе қауіпсіздік анализінің әртүрлі әдістері ұсынылған болатын. Оларды Стори (1996) жинақтаған. Оның ішіне әдістерді шолу және бақылау тізбелері, формалдық әдістері, мысалы, Петри желісінің анализі («Петерсон», 1981) және ақаулық ағашының анализі («Левесон және Столзи», 1987; Стори, 1996) кіргізілген. Осы кітапта бұл әдістердің барлығының маңызды мәселелерін көтеруге мүмкіндік болмағандықтан мен қауіп-қатер анализінде неғұрлым жиірек

кездесетін әдістерге тоқталамын. Бұл техниканы арнайы мағлұматтарсыз едәуір жеңіл түсінуге болады.

Ақаулық ағашының анализін жасау үшін, сіз алдымен бірдейлестірілген қауіптерден бастайсыздар. Әр қауіпті жағдай үшін, сол жағдайдың себеп-салдарын табу үшін сіз қарама-қарсы бағытта жұмыс істеуіңіз керек. Қауіпті ағаштың түбіне салып, қауіп-қатерге әкеліп соғатын жүйенің жағдайын бірдейлестіресіздер. Кейін әр жағдай үшін сіздер осы тәрізді үдерісті ақаудың бас мәселесін тапқанша ажыратуды жалғастырып жұмыс жасайсыздар. Бас әрекеттің нәтижесінен туатын қауіптерге қарағанда, жалғыз бір себептен туатын қауіптер негізінде азырақ апатқа әкеліп соқтырады.



12.4-сурет. Себеп құрылымының мысалы

12.4-сурет. Ақаулық ағашының бағдарламалық қамсыздандырумен байланысты қауіптер үшін инсулинді жеткізетін жүйеде инсулинді дұрыс мөлшерлемейтін ақауларға әкеліп соғады. Бұл жерде «жеткіліксіз мөлшер» және «тым артық мөлшер» бір қауіпке бірігеді. Айтылған ақаулықтар тал-шыбығының ақаулықтарының са-

нын қысқартады. Әрине, сіз бағдарламалық қамсыздандырудың осы қауіпке қалай назар аудару керек екенін міндетті түрде анықтай білуіңіз керек. Сіз инсулинның артық кеткен мөлшерін кем мөлшерден ажырата білуіңіз керек. Мен жоғарыда айтып кеттім, мөлшерден қате кетсе, оның зардабы әртүрлі – мөлшер тым артық кетсе, ол өте қауіпті жағдай.

12.4-суретте көрсетілген**:**

1. Инсулинның қате мөлшеріне әкеліп соғатын үш түрлі жағдай болады. Ең алғашта кан құрамындағы қант деңгейі дұрыс өлшенбегеннен инсулин талабы дұрыс есептелмейді. Жеткізу жүйесі инсулин енгізу командасына дұрыс жауап бермей қалуы мүмкін. Баламалы жағдай, инсулин мөлшері дұрыс есептелген, бірақ ол аса ерте немесе аса кеш қойылған.
2. Қандағы қант деңгейінің қате өлшенуіне қате ағашының сол тармағы жауапты. Қате кету себептері қандағы қант мөлшерінің кіріс механизмінің дұрыс істеп тұрғанын көрсететін аспаптың бұзылуы немесе қант өлшеудің есептеулері дұрыс болмауынан болады. Қандағы қант деңгейі адам терісінің өткізу қабілетіне байланысты болады. Қате есептелім арадағы дұрыс емес алгоритмнің немесе арифметикалық қатенің салдарынан туады (мысалы, сандардың арасындағы үтірді дұрыс қоймай себептері).
3. Орталық тармақ ағаш уақыт мәселесімен байланыста болады және ол жүйе таймерінің бұзылу нәтижесінде орын алатын жағдай.
4. Ағаштың оң тармағы жеткізу жүйесінің бұзылуына қатысты осы сәтсіздіктің ықтимал себептерін зерттейді. Олар инсулин талабының қате есептелімінен туған немесе инсулинді жеткізетін сорапқа дұрыс белгілерді жіберуден бас тартқан жағдайда туады. Дұрыс емес есептелім алгоритмнің немесе арифметикалық қатенің сәтсіздігінен болады.

Ақаулықтың тал-шыбықтары аппараттың құрал-саймандарының әлеуетті мәселелерін бірдейлестіруде қолданылады. Аппаратты құрал-саймандардың ақаулығының тал-шыбықтары зейін талабының мәніне бағдарламалық қамсыздандырумен қамтамасыз етеді, тек сонда ғана осы мәселелерді анықтап тауып, оны жөндеуге мүмкіндік болады. Айталық, инсулинның мөлшерін қайта-қайта реттеудің қажеті жоқ. Сол себептен, процессордың алымдылығы диагностикалық және өзін-өзі тексеріп отыратын бағдарламаны басқаруды қамтамасыз етеді. Аппараттың құрал-саймандарының қателері, мысалы, бергіш, сорап немесе таймердің қателері табылуы мүмкін, ал олар болса тиянақты нәтижені емделушіге жасаудан бұрын, алдымен ескертулер шығарады.

12.2.4. Қауіп деңгейін төмендету

Ықтимал қауіп-қатерлер мен олардың басты мәселелері бірдейлестірілгеннен кейін, сендер қауіп-қатерді басқарып және бақытсыз жағдайларды болдырмауға

кепілдік жасайтын қауіпсіздік техникасының талаптарын ала аласыңдар. Ықтимал стратегияның үш түрі болады:

1. *Қауіптің алдын алу.* Жүйе қауіп-қатерді болдырмайтын етіп әзірленген.
2. *Қауіп-қатерді тауып, оның көзін жою.* Жүйе әзірленген кезде қауіп-қатерді уақытында тауып, оны бақытсыз уақиға болмай тұрып бейтараптандыру алдын ала ескерілген.
3. *Шығынды шектеу.* Жүйе бақытсыз уақиғаның зардабы неғұрлым аз болатындай әзірленген.

Негізі, қысылшаң жүйенің жобалаушылары осы амал әрекетін пайдаланады. Қауіпсіздікке қысылшаң жүйеде адам төзгісіз қауіптер өңделініп, оның болу себептерін азайтатын және жүйенің қауіпсіздігінің сақтық көшірмесін қамсыздандыратын өңдеу жұмыстары жүргізілу керек. Мысалы, химиялық зауыттың басқармасының жүйесінде, реактордағы артық қысымды тауып, одан арылуға тырмысады. Алайда, онымен қатар қысымды қадағалайтын және қысым өте жоғары көтеріліп кеткенде сақтандыру клапанын ашатын тәуелсіз сақтандыру жүйесі болады.

Инсулинның жеткізуінің жүйесіндегі 'сенімді күй' – инсулин енгізілуі тоқтаған кезде оның да тоқтап қалуы. Қысқа уақыт ішінде мұндай қауіп диабетпен ауыратын адамның денсаулығына еш зиянын тигізіп үлгірмейді. Инсулинның қате мөлшерін есептейтін бағдарламалық қамсыздандырудың ақаулықтары үшін, келесі «шешімдер» әзірлену керек:

1. *Арифметикалық қате.*Арифметикалық есептелімінен кеткен қатеден болатын ақаулық. Маманданым алдын ала барлық ықтимал арифметикалық қателерді болжап және өңдеуіші ықтимал қатенің әрқайсысы үшін өңдеу ерекшелігі косылу керек екенін болжап, бірдейлестіруі керек. Маманданым қателердің әрқайсысы үшін іс-әрекеттерін баяндау керек. Жасырын қауіпсіз әрекетпен жеткізу жүйесін ағытып тастау және дыбысты ескерту белгісін белсендіру болып табылады.
2. *Алгоритмдік қате.* Бұл өте қиын жағдай, себебі бағдарламаның өңделуге қажетті ешкандай ерекшелігі жоқ. Қатенің осы үлгісі – инсулинның қажетті мөлшерін алдында енгізілген мөлшермен салыстырған кезде ғана табылатын қате. Ол мағыналардан жоғары болса, онда мөлшердің дұрыс есептелмегені. Жүйе тізбектелудің мөлшерін бақылай біледі. Көп қате санынан кейін жүйе ескерту жасап, инсулин мөлшерін шектейді.

Инсулинның сорабы жүйесінің бағдарламалық қамсыздандыруының қауіпсіздігіне қойылатын талаптар *12.5-суретте* көрсетілген. Ол қолданбалы талаптар және олардың жүйелі талабының маманданымында толық айтылған. *12.5-суреттегі, 3- және 4-кестелердегі* сілтемелер құжаттарға енгізілген талаптардың кестесі болып табылады да, бұл жерде көрсетілмейді.

SR1: Инсулин дозасының белгілі бір науқасқа рұқсат етілген мөлшерінен артық берілмеуі.

SR2: Инсулин дозасының белгілі бір науқасқа бір күндік мөлшерінен артық берілмеуі.

SR3: Жүйе кемінде сағатына 4 рет өзін-өзі тексеріп отыру керек.

SR4: Жүйе *3-кестеде* анықталған барлық ерекшеліктерге дайын болуы керек.

SR5: Жүйенің сәтсіздікке ұшырағаны жағдайында арнайы дыбыс шығып, *4-кестедегі* көрсетілген хабарламалардың біреуі дисплейге шығарылады.

SR6: Егер дыбыс шықса, инсулин тасымалы тоқтатылады. Егер адам сәтсіздік мәселесінің шешімін тапса, инсулин тасымалы қайта жалғастырылады.

12.5-сурет. Қауіпсіздік талаптардың мысалдары

12.3. Сенімділікке қойылатын талаптар

Мен *10-тараудың* басында айтқандай, жүйенің толық сенімділігі аппаратты құралдардың сенімділігіне, бағдарламалық қамтамасыз етудің сенімділігіне және жүйелік операторлардың сенімділігіне бағынышты болады. Жүйелік бағдарламалық қамтамасыз ету бұл жағдайды назарына қабылдауы керек. Мұнда тағы да сенімділіктің талабы есепке алынады, аппаратты құралдардың істен шығуларынан кейін дәрігерлік жан сақтау және оператор қателігінен кейін табылуға көмек үшін, сол бағдарламалық қамтамасыз етудің іркілісіне өтемді талаптарды қосады. Сенімділіктің қауіпсіздіктен айырмашылығы – сенімділік өлшенетін жүйелі белгі. Егер қажетті сенімділік мақсатына жететін болса, қорыта келгенде, біз талап етілетін сенімділіктің деңгейін анықтауға, ұзақ уақыт жүйенің әрекеті ағымында бақылай аламыз. Мысалы, сенімділіктің талабымен жүйе аптада бір реттен артық қайта жүктеуге тиісті емес шарт ретінде болуы мүмкін.

Мысалы, сенімділіктің талабымен жүйе аптада бір реттен артық қайта жүктеуге тиісті емес шарт ретінде болуы мүмкін. Мұндай іркіліс туған сайын, ол тіркелуі мүмкін және сіз қажетті сенімділіктің деңгейіне жеткен-жетпегенін тексере аласыз. Сіз басқа жағдайда өз сенімділік талабыңызды өзгертесіз немесе негізгі жүйелік мәселелерді шешу үшін өзгеріске сауал ұсына аласыз.

Жағымсыз жағдайлардан сақтанып, қауіпсіздік немесе қорғаныстың қажетті «деңгейін» анықтаудың орнына қорғаныс та, қауіпсіздік те олардан қашқақтайды. Тіпті, бүкіл жүйенің циклінде бір ғана мұндай жағдайды болдыруға болмайды, егер дегенмен, мұндай жағдай туып жатса, онда жүйелі өзгерістер енгізілуі тиіс. «Жүйелік қателер жылына 10-нан артық зақымданудан аспау керек « деген сияқты тапсырма қоюға рұқсат етілмейді, өйткені тек бір зақымдану болды ма, іле-шала жүйелік мәселе дұрысталып шешілуі керек.

Сондықтан сенімділік талаптарының екі түрі болады:

1. Функционалдық емес талаптар – жүйені қалыпты пайдалану кезінде қателердің санын анықтайтын талап немесе жүйе қол жетпес жағдайдағы уақыт. Бұл сенімділіктің есептік талаптары.
2. Жүйені анықтайтын функционалдық талаптар және бағдарламалық қамтамасыз етудің функциясы. Олар жоламай қашқақтап, бағдарламалық қамтамасыз етуде қателік жіберіп немесе сол қателіктерді дер уақытында тауып, сонымен қатар бұл қателер жүйелік сәтсіздікке әкелмейтініне кепілдік береді.

Сенімділіктің есептік талаптары байланған функционалдық жүйелік талаптарға алып келеді. Кейбір қажетті сенімділіктің деңгейіне жету үшін, функционалдық және жүйенің конструктивтік талаптары болуы мүмкін қателерді анықтауы және қабылдануы керек болатын өлшемдерді анықтауы керек, өйткені бұл қателер жүйелік жаңылуларға әкелмейтініне кепілдік береді.

12.1-суретте сенімділіктің спецификациясының үдерісі ортақ басқарылатын тәуекелде негізделген:

1. *Тәуекелді бірдейлендіру.* Осы кезеңде, сіз экономикалық ысыраптарға әкеліп соғатын жүйелік сәтсіздіктердің түрлерін сәйкестендіресіз. Мысалы: электронды коммерцияның жүйесіне қол жеткізу мүмкін емес делік, демек, клиенттер өз тапсырыстарын жасай алмайды немесе мәліметтерді бұзатын іркіліс туады, соның кесірінен қор көшірмесінен жүйелік дерекқорды қалпына келтіру үшін және өңдеген келісімдерді қайтадан іске қосу үшін уақыт керек болады. *12.6-суретте* қауіп-қатерді сәйкестендіру үшін бастама болып табылатын қателіктердің тізімі көрсетілген.
2. *Тәуекел дәрежесінің талдауы.* Бұл талдаудың ішіне шығынның бағасы және бағдарламалық қамтамасыз етудің қателерінің әртүрлі зардаптары кіреді, сонымен бірге одан әрі талдау үшін тәуекелдің биік деңгейінің қателерін іріктейді.
3. *Тәуекелді жасаушы*. Бұл жерде сіз маңызды және болуы мүмкін жүйелік іркілістердің себептеріне талдау жасайсыз. Алайда, бұл талаптың кезеңінде мүмкін емес, себебі себеп жүйелік жобалаудың шешімдеріне бағынышты болады. Сірә, сізге жобалау қызметі кезінде және әзірлеу кезінде қайта оралуға тура келеді.
4. *Тәуекелді төмендету*. Бұл кезеңде, сіз, сәтсіздіктердің әртүрлі түрлерін қолайлы ықтималдықтар болатын сенімділік есептік техникалық талаптарын жасауыңыз керек. Әрине, олар сәтсіздіктердің шығындарын ескеруі керек. Сіз әртүрлі жүйелік қызметтер үшін әртүрлі ықтималдықтарды пайдалана аласыз. Сіз сенімділіктің функционалдық талаптарын жасай аласыз. Бірақ жүйелік жобалаудың шешімі қабылданып жатқанда күтуге тура келеді. Алайда, *12.3.2-бөлімде* айта кететін жағдай, кейде есептік техникалық та-

лаптарды құру қиынға түседі. Сіз тек қана сенімділіктің функционалдық талаптарын сәйкестендірудің жағдайын жасай аласыз.

Сәтсіздіктің түрі	Баяндама
Қызметтің жоғалуы	Жүйе қолжетімсіз болып, өзінің қызметтерін дұрыс орындай алмайды. Бұл қызметтерді критикалық немесе критикалық емес деп ажыратуға болады. Критикалық емес қызметтердің қолжетімсіз болуы жүйені критикалық қызметтерден артық зақымдатады.
Қызметтің бұрыс жұмыс атқаруы	Жүйе пайдаланушыларға дұрыс қызмет атқармайды. Бұл да маңызды немесе маңызды емес болып ажыратылады.
Жүйенің / деректердің бұзылуы	Жүйенің сәтсіздігі жүйенің немесе деректердің бұзылып зақымдануына алып келеді. Бұл көбінесе басқа сәтсіздіктермен қатар жүреді.

12.6-сурет. Жүйе сәтсіздіктерінің түрлері

12.3.1. Сенімділіктің көрсеткіштері

Жалпы айтқанда, сенімділік жүйелік қатенің болуы мүмкін екенін, жүйе осы көрсетілген операциялық ортада пайдаланылатынын айқындайды. Мысалы, егер 1000 келісімнің біреуі сәтсіздікке ұшыраса, онда сіз істен шығу ықтималдығы 0, 001%-ға тең деп көрсете аласыз. Әрине, бұл жерде әр 1000 операциялардың біреуі жаңылысқа ұшырайды деп білуге болмайды. Бұл, егер сіз N мәмілені қадағалайтын болсаңыз, онда істен шығып бұзылу сандары N-ға тең болу керек екенін білдіретінін білесіз. Сіз істен шығулардың әртүрлері үшін анықтап ала аласыз немесе әртүрлі жүйенің бір бөліктері үшін анықтай аласыз. Сіз сыни құрамдас бөліктердің сын көзімен қарамау құрамдас бөліктеріне қарағанда істен шығу ықтималдығы төменірек деп ойлауыңыз мүмкін.

Сенімділіктің анықтамасы үшін пайдаланатын екі маңызды көрсеткіштер және қолжетімділік жүйесінің байланған төлсипаттарын бірдейлендіру үшін пайдаланатын бір қосымша көрсеткіш болады. Көрсеткішті таңдау қолдану аймағындағы талаптарға және қазіргі уақытта анықталынған талаптар жүйесіне бағыныштыкеледі. Мысалы:

1. *(ВНПТ) талабы бойынша сәтсіздіктің ықтималдығы*. Егер сіз бұл көрсеткішпен пайдаланатын болсаңыз, онда жүйе арқылы қызмет көрсетуге сұраныс жүйелік іркіліске әкеліп соғатынын анықтайсыз. Осылай, 5 0.001-ші ВНПТ 1/1,000 іркілістің болу мүмкіндігін білдіреді.
2. *(ТВН) сәтсіздіктердің пайда болуының қарқыны*. Бұл көрсеткіш жүйеде болуы мүмкін қателердің санын баяндайды, олар белгілі уақыт кезеңдерінде байқалған жүйелік қателердің мүмкіндік санын баяндайды (мысалы, 1 сағат)

немесе жүйелік орындаудың санын көрсетеді. Жоғарыдағы мысалда ТВН-1/1,000. ТВН аналогы – сенімділікті кейде көрсеткіш ретінде пайдаланатын бұзылуға дейінгі орташа уақыт (СВДО). СВДО – бақылаланатын жүйелік қателердің арасындағы уақыт бірліктерінің орташа саны. Егер қатенің пайда болуы орташа есеппен сағатына екі сәтсіздікке тең болса, онда қателіктің тууына 30 минут уақыт кетеді.

3. *(ДОСТ) қолжетімділігі*. Жүйенің қолжетімділігі өзінің талап бойынша қызмет көрсетуін бейнелеп көрсетеді. ДОСТ – сұранысқа сай жүйенің дұрыс жұмыс істеу ықтималдығы. Сондықтан, ДОСТ = 0.9999 жүйенің 99,99% қол жететін ағымында операциялық уақыт ретінде болатынын білдіреді. *12.7-сурет* қолжетімділіктің әртүрлі деңгейлерін көрсетеді.

Қолжетімділік	Түсініктеме
0.9	Жүйе 90% уақыт бойы қолжетімді болады. Бұл дегеніміз, 24 сағатта (1240 минут), жүйе 14,4 минут бойы қолжетімсіз болады.
0,99	24 сағатта жүйе 14,4 минут бойы қолжетімсіз болады.
0,999	24 сағатта, жүйе 84 секунд бойы қолжетімсіз болады.
0,9999	Аптасына жүйе жуық шамамен 1 минут бойы қолжетімсіз болады

12.7-сурет. Қолжетімділік сипаттамасы

ВНПТ талапқа сай іркіліс мүмкін маңызды жүйелік қатеге әкелетін жағдайларда сенімділік көрсеткіш ретінде пайдалануы керек. Бұл талаптар жиіліктен тәуелсіз қолданылады. Мысалы, химия реакторын бақылайтын және реактор қызып кеткенде оны тоқтататын жүйе қорғанысы ВНПТ-ны қолдануы керек. Әдетте, қорғаныстың жүйесіне талап сирек болады, өйткені жүйе – соңғы қорғаныс шегі, қорғаныстың қалған әдістері бәрі осындайда пайдалану нәтижесіз екенін көрсетті. Сондықтан (1000 талаптарға 1 сәтсіздік) бұл көрсеткіш өте қауіпті көрсеткіш болып көрінеді, егер бүкіл цикл бойына екі немесе үш талап қойылса, онда сіз мүмкін жүйелік қатені мүлдем көрмей қалуыңыз мүмкін. ТВН-жүйелерге талаптар мерзімді емес үнемі болатын жағдайларда ғана пайдалану ең жақсы көрсеткіш болады. Мысалы, транзакциялардың үлкен санын жұмыстанатын жүйеде ТВН күніне 10 іркіліс көрсетуге болады.

Күніне орташа есеппен 10 келісімшарттар ойдағыдай болмай, оларды өзгерту керек екенін мақұлдауымыз керек. Және сіз ТВН- да 1000 операцияға тоқтап қалу санын көрсете аласыз. Егер сәтсіздіктердің арасындағы абсолют уақыты маңызды болса, сіз сенімділікті бұл сәтсіздіктердің арасындағы орташа уақыт ретінде анықтай аласыз. Мысалы, егер сіз (автоматтандырылған жобалау жүйе сияқты) ұзақ мерзімді транзакциялармен жүйе үшін қажетті сенімділікті анықтасаңыз, сіз сәтсіздікке қарым-қатынастың орташадан үлкен аралыққа дейін сенімділікті анықтауыңыз керек. СВДО қолданушы олардың нәтижесін үнемдемей үлгімен

жұмыс істеп жатқан кезде, орташа уақытқа қарағанда әлдеқайда ұзындау болуы керек. Бұл қолданушы кез келген сессияда жүйелік қатеге байланысты жұмыс жоғалтпайтынын білдірер еді.

Жүйенің сенімділігін бағалау үшін, сіз оның іс-әрекеті туралы мәліметтерді алуыңыз керек. Талап етілетін мәліметтерге кіретіндер:

1. Сауалдардың жүйелік қызметтеріне қатар алған жүйелік істен шығулардың саны. Бұлар ВНПТ өлшеу үшін пайдаланылады.
2. Жүйелік сәтсіздіктердің арасындағы уақыт немесе келісімдер саны және оған толық жұмсалған уақыт немесе келісімнің жалпы саны қосылады. Бұл ТВН мен СВДО-ның өлшемдерін алу үшін қолданылады.
3. Жөндеудің уақыты немесе қызмет көрсетудің ысырабына алып келетін жүйелік қатеден кейін қайта жіберілуі. Бұл қолжетімділіктің өлшеуінде пайдаланады. Қолжетімділік қателердің арасындағы уақытқа бағынышты болады, бірақ сонымен бірге талап етілетін операцияға жүйені кері қайтару үшін берілген уақытқа да бағынышты келеді.

Пайдалану үшін алынған уақыт бірліктері: күнтізбелік уақыттың көрсеткіші, процессор жұмысының уақыт бірлігін көрсеткіш немесе дискретті бірлікті көрсеткіш (мысалы, транзакциялардың саны). Көп уақыт ішінде ештеңе жасамайтын жүйелердің ішінде, жүйе жұмыс істемей тұрған кездегі уақытты алып тастау үшін сіз процессордың жұмыс уақытын көрсететін көрсеткіштерді пайдалануыңыз керек (телефон коммутация жүйесінде пайдаланылады) .

Сіз үздіксіз режимде жұмыс істейтін жүйелерге күнтізбелік уақыттың көрсеткіштерін қолдануларыңыз керек, бұл категорияға: мониторинг жүйесі, сигнализация және технологиялық үдерістерді басқарудағы жүйенің басқа түрлері кіреді. Банкоматтарды қолданатын немесе ұшу билеттерін брондайтын операцияларды өңдейтін жүйелер тәулік уақытына негізделген жүктеме айнымалыларға ие болады. Яғни ТВН-ның транзакцияларының саны осы жағдайда пайдаланылатын «уақыт бірліктері» болуы мүмкін, демек, ТВН N мың келісімнің сәті түспеген келісімдерінің саны болады.

12.3.2. Функционалды емес сенімділіктің талаптары

Сенімділіктің функционалдық емес талаптары – қажетті сенімділік пен жүйенің қолжетімділігінің есептік техникалық талаптары, олар алдыңғы бөлімде айтылған көрсеткіштердің бірінің көмегімен есептеп шығарылған. Сенімділіктің және қолжетімділіктің есептік спецификациясы, көп жылдар бойы жүйелердің қауіпсіздігіне сыни қатысты пайдаланды, бірақ сыни жүйелерде өте сирек пайдаланылады. Алайда, қазіргі күнде компаниялардың көбі тәулік бойы күні-түні жұмыс істеп қызмет көрсететін жүйелерді талап етеді, сондықтан бұндай әдістерді пайдалану күннен күнге көбейіп барады.

Сенімділіктің есептік техникалық талаптарын алудың бірнеше артықшылықтары бар:

1. Сенімділіктің деңгейін таңдаудың үдерісі мүдделі адамдардың шынымен қандай көмекке мұқтаж екенін түсіндіруге көмектесе алады. Бұл жүйелік іркілістің әртүрлі түрлерінің болатынын түсінуге көмектеседі және сенімділіктің биік деңгейіне жету өте күрделі де қымбат екенін анық түсінуге көмектеседі.
2. Бұл негіздеу жүйені тексеруді қашан тоқтату керек екенін түсініп, баға беруге көмектеседі. Сіз жүйені өзінің қажетті сенімділіктің деңгейіне жеткен кезде тоқтата аласыз.
3. Бұл – жүйенің сенімділігінің жақсартуы үшін қолайлы әрлендірулердің әртүрлі стратегиялары бағасының құрал-жабдықтары. Сіз әр стратегияның қажетті сенімділіктің деңгейлеріне әкеле алатыны туралы пікір жасай аласыз.
4. Егер реттегіш жүйені пайдаланудан бұрын оны мақұлдауы керек болса (мысалы, ұшудың қауіпсіздігі үшін ұшақтың оған жауап беретін маңызды жүйелері реттелуі тиіс), онда жүйенің сертификациясы үшін маңызды қажетті сенімділіктің мақсатын дәлелдеу орындалды деп есептейміз.

Сіз қажетті жүйелік сенімділіктің деңгейін орнату үшін жүйелік қатені тергеуде пайда болатын байланған ысыраптарды қарап шығуыңыз керек. Бұл – ақшалай ғана зияндар емес, сонымен бірге кәсіп абыройының ысырабы. Абыройдың ысырабы клиентураның ысырабын білдіреді. Бір жағынан жүйелік қатеден қысқа мерзімді ысыраптар біршама ұсақ болса, ұзақ мерзімді ысыраптар әлдеқайда маңыздырақ болады. Мысалы, егер сіз ғаламтор дүкенінің жүйесіне қол жеткізетін болып жатқанда оған қол жеткізу мүмкін емес екенін көрсеңіз, сіз өзіңізге керегін жүйе қол жететін жағдайға жеткенше күтіп отырмай басқа жерден табуға тырысасыз. Егер бұндай жағдай үнемі болып тұрса, онда сіз бұл жерден сатып алмайсыз.

ВНПТ, ТВН, сондай-ақ ДОСТ көрсеткіштерін және сенімділіктің анықтамасын, мәселе көрсеткіштерді пайдалана отырып, сол сенімділікті қайта бағалау үшін биік шығындарды жаңғыртуға, қорыта келгенде апаруы және бекітуі мүмкін. Себебі – мүдделі адамды және есептік техникалық талаптарға жаттығу тәжірибесін ауыстыруға қиын. Бұл мүдденің есептік көрсеткіші техникалық талаптарды ауыстыруы және тәжірибесі қиын болып келуінде. Олар (1, 000 талаптарға 1 қате) 0.001-ші ВНПТ-ға қатысты жүйені сенімді жүйе деп ойлауы мүмкін. Егер менің түсіндіруім бойынша қызмет көрсетуге талабы ерекше болса, ол сенімділіктің деңгейі іс жүзінде өте биік деңгейге ілінеді.

Егер сіз сенімділікті көрсеткіш ретінде анықтасаңыз, сенімді деңгейге жеттіңіз бе, соны бағалау керек. Сіз бұл бағалауды жүйелік тестілеудің бөлігі ретінде қарастырасыз. Жүйенің сенімді тестілеуін арттыру үшін статистика бойынша сіз көп қателіктер арқылы өтуіңіз керек. Мысалы, егер сізде (10,000 талаптарға 1 қате) 0.0001-ші ВНПТ бар болса, онда сізге тестілеуді қайта жобалау қажет болады, жүйенің 50 немесе 60 қателіктерінен гөрі, жүйеде талап бойынша бірнеше

қателіктер ғана байқалуы қажет. Іс жүзінде өңдеп және осындай мөлшерде тест енгізу мүмкін емес болып көрінеді. Сондықтан сенімділіктің артық санап теруі жүйенің тестілеуін биік шығындарға алып келеді.

Сіз жүйенің қолжетімділігін анықтаған кезде келесі мәселелермен қақтығысуыңыз мүмкін. Бізге ең жоғары деңгейдегі қолжетімділік керек сияқты болып көрінгенмен, өте көп жүйелерде тұрақсыз талаптардың үлгілері (мысалы, бизнес-жүйелер көбіне, жұмыс уақытында қолданылады), жалғыз жоғарғы өлшем қажетті тұтынушылықты көрсетпейді. Сіз жүйе пайдаланылған кезде ғана жағдайы биік қолжетімділікке мұқтаж боласыз. Әрине, жүйенің түріне байланыса тұра, қолжетімділіктің арасында ешқандай нақты 0.999 бен 0.9999-дың іс жүзінде айырмашылығы болуы мүмкін емес.

Сенімділік пен қолжетімділіктің өте биік деңгейге жеткенін анықтау мүмкін емес болатын жағдайлар артық спецификацияның негізгі мәселесі болып табылады. Мысалы, жүйе қауіпсіздік үшін өте маңызды жағдайларда қосымша бөлімдердің қолданылуына арналған делік, сол үшін жүйе жұмыстың барлық кезеңінде ауыстырылып қосылмауы керектігі талап етіледі. Бізге жүйенің 1,000 көшірмелерін орнату керек делік, ал жүйе бір секундта 1,000 рет орындалады. Болжам бойынша жүйенің қызмет ету мерзімі 10 жыл.

Жүйелік орындаудың жалпы саны шамамен $3*10^{14}$. Қателердің пайда болуының қарқыны (бұл кейбір беріктік қорды ескереді) 10^{15}-ке 1 болуы керек екенін анықтаудың ешқандай мағынасы жоқ, себебі сіз сенімділіктің бұл деңгейін бекіту үшін, жүйені әжептәуір ұзақ тексере алмайсыз. Сенімділіктің өте биік деңгейін анықтауға және бекітуге болатынын білу үшін мекемелер реалист болу керек. Сенімділіктің биік деңгейлері жүйеде міндетті түрде қажет (телефонның ауыстырылып қосылу жүйесі немесе жүйелік қателер үлкен экономикалық шығындарға әкеліп соғатын кездер).

Сірә, олар іскер немесе ғылыми жүйелер үшін қолайсыз да болар. Мұндай жүйелерде сенімділіктің талаптары сыпайы болады, қиыншылық және аса қымбатшылық шақырмайды, өйткені кеткен қателердің құны – өңдеудің аялдауы.

Сенімділік жүйесінің артық спецификациясынан құтылу үшін іске қосып қабылдауға болатын бірнеше қадамдар болады:

1. Әртүрлі қателердің үлгілері үшін қолжетімділіктің талабын және сенімділігін анықтаңыз. Болмашы іркілістерге қарағанда маңызды қателіктердің пайда болмауына ықтималдық жасап, қамтамасыз ету керек.
2. Қолжетімділіктің және көрсетілетін қызметтердің әртүрлері үшін талаптарды анықтаңыз. Ең сыни қызметтерге әсер ететін қателер тек жергілікті әсер болуына қарағанда мүмкіндігі өте аз болып қаралуы керек. Сіз сыни жүйелік қызметтер мен сенімділіктің ең есептік спецификациясын шектеуді шеше аласыз.
3. Сізге шынымен бағдарламалық қамтамасыз етудің жүйесінде биік сенімділік немесе сенімділіктің ортақ жүйелік мақсаттарын басқа амал-тәсілмен алуыңыз қажет екендігін шешкеніңіз жөн. Мысалы, сізге жүйенің өнімін тексеру үшін және қатені түзету үшін керекті үдерістер қоластыңызда болу

керек. Сіз ол деректердің тетіктерін пайдалана отырып, қателіктерді анықтай аласыз. Өнім өндірген жүйеде сенімділік биік деңгейде болу қажеттілігі сол кезде жоғалады.

Соңғы тармақтағы суреттемені мысал ретінде қарастыратын болсақ, қолма-қол ақшаны үлестіретін ақша үлестіргіштің жүйесі сенімділіктің талабымен ақшаны үлестіргенін және клиенттерге басқа қызмет көрсеткенін қарап шығуымыз керек. Аппаратты қаржылармен мәселе болған жағдайда немесе ақша үлестірушіні бағдарламалық қамтамасыз етуде олар клиенттер есебінің шотындағы теріс жазбаларға әкеліп соғады. Ақша үлестіруші банкоматтың бағдарламалық қамтамасыздығын және сенімділіктің өте биік деңгейін анықтау барысында көп қателіктерден құтылуға болады.

Алайда, шот операциясының қателіктерін анықтап, оларды жөндеуде банктерде көпжылдық тәжірибе бар. Қате болып жатқан кезеңді дер кезінде табу үшін, олар бухгалтерлік әдістерді пайдаланады. Сәтсіздікке ұшыраған мәмілелердің көпшілігі, банктерге зиянын тигізбей, тек болмашы тұтынушы қолайсыздықтарын тудырып тұрғанда өзгертіле алады. Ақша үлестірушілердің желілерін басқаратын банктердің айтуы бойынша қорғаныс жасауда қатені жеңілдету үшін үлкен шығын шығарғанша сол қатені түзету жеңілге түседі.

Банк және банктің клиенттері үшін ақша үлестірушілерді желінің қолжетімділігі кейбір мәмілелер қателерінің ең төменгі санына қарағанда маңыздырақ, Қолжетімділіктің тапшылығы желіні қалпына келтірудегі қарама-қарсы қызметтерге, тұтынушы қанағаттанбаушылығына, техникалық шығындарға үлкенірек сұранысты және т.б. білдіреді. Сондықтан транзакция шығаратын жүйелер үшін (банктер, электронды коммерция) сенімділіктің спецификация орталығы әдетте, жүйенің қолжетімділігін анықтауда болады.

Ақша үлестірушілер желінің қолжетімділігін анықтау үшін, сіз жүйелік қызметтерді сәйкестендіруіңіз және олардың әрқайсыларына қажетті қолжетімділігін анықтауыңыз керек. Бұл:

- клиент шотының дерекқорына қызмет көрсету;
- ақшаны қолма-қол алғандарға банкомат көрсеткен жеке қызметтер шот жайлы ақпарат береді және т.б.

Осы жағдайда, дерекқорға қызмет көрсету ең маңызды болып көрінеді, себебі бұл қызмет көрсетудегі сәтсіздік бүкіл желідегі ақша үлестірушілерде іркіліс бар деген сөз. Биік деңгейге қол жеткізу үшін сіз бұны анықтап алуыңыз керек. Бұл жағдайда қолжетімділіктер үшін дерекқор ақпараттар саны (болжамды қызмет көрсету және жаңғырту мәселелерін елемейтін) жуықтап алғанда, дерекқор үшін қолжетімділіктің қолайлы саны шамамен, таңертеңгі сағат 7-ден кешкі сағат 11-ге дейін 0.9999 болуы мүмкін. Яғни, тұрыс уақыты 1 аптада бір минуттан да аз дегенді білдіреді, іс жүзінде неғұрлым кем тұтынушыларға әсер етіп, аздаған қолайсыздықтарға әкеледі деп есептеледі.

Жеке ақша үлестіруші банкомат үшін толық қолжетімділік механикалық сенімділікке және деректерге бағынышты болады, осы қолжетімділік қолма-қол ақшаны тауысатыны айғақ. Толық қолжетімділік жеке ақша үлестіруші үшін механикалық сенімділікке бағынышты болады және осы қолжетімділік қолма-қол ақшаны мүмкіндігінше тауысатыны айғақ. Сірә, мұндай факторларға қарағанда бағдарламалық қамтамасыз ету мәселесінің неғұрлым аз әсері болады. Сондықтан ақша үлестірушіні бағдарламалық қамтамасыз етуге қолжетімділіктің деңгейі қолайлы болып көрінеді. Ақша үлестірушінің бағдарламалық қамтамасыз етуінің толық қолжетімділігі 0.999 болып анықталуы мүмкін, машина бір күн ішінде бір минут пен екі минуттың арасында қол жетпес болуы мүмкін болатындығын білдіреді.

Сенімділіктің спецификациясына негізделген іркілістерге мысал келтіретін болсақ, инсулин сорғысында бақылауды қамтамасыз ететін сенімділік талабының бағдарламасын қарап шығамыз. Бұл жүйе қанға күніне сан рет инсулин жеткізіп тұрады және қолданушының қанындағы глюказаны сағатына бірнеше рет бақылайды. Себебі жүйенің пайдалануы аумалы және ВНПТ (талап бойымен сәтсіздіктің ықтималдығы) сенімділіктің ең маңызды көрсеткіші сәтсіздіктің зардабы. Инсулин сорғысында болуы мүмкін сәтсіздіктің екі түрі бар:

1. Ығысу немесе машинаны калибрлеу сияқты қолданбалы әрекетпен қалпына келтірілетін бағдарламалық қамтамасыз етудің өтпелі сәтсіздігі. Қателердің бұл түрлері үшін, мысалы, 0.002, ВНПТ-ның ең төменгі мағынасы қолайлы болады. Бұл жасалған машинада әр 500 талапқа бір сәтсіздік болуы мүмкін екенін білдіреді. Бұл шамамен, 3.5 күнде бір сәтсіздік, өйткені қанда қант жуық шамамен, сағатына бес рет тексеруден өтеді.
2. Бағдарламалық қамтамасыз етуді өндіріп-даярлаушылармен қайтадан қондыруды талап ететін тұрақты сәтсіздіктер. Сәтсіздіктің ықтималдығы әлдеқайда төмен болуы керек. Ең төменгі сан – жылына шамамен бір рет, қорыта келгенде, ВНПТ 0,00002-ден артық болмауы керек.

Алайда, инсулинді жеткізуден бас тартуда тікелей қорғаныстық мағыналар болмайды, сенімділік коэффициенттері орнына мұндай коммерциялық факторлар сенімділік коэффициенттерінің орнына талап етілетін сенімділіктің деңгейін басқарады. Сервистік шығындар өте жоғары, өйткені қолданушыларға өте тез жөндеу және алмастыру қажет. Жөндеуді қажет ететін тұрақты қателіктердің санын шектеу жасап-шығарушыларды да қызықтырады.

12.3.3. Сенімділіктің функционалдық спецификациясы

Сенімділіктің функционалдық спецификациясы анықтайтын шектеулер бірдейлендірудің талабынан тұрады және жүйелік сенімділіктердің мүмкіндігін туғызатын ерекшеліктері болады. Сан ретінде анықталынған бұл функционалдық

талаптар сенімділігі жүйе үшін өте қажетті келеді, сенімділіктің деңгейіндегі жетістікке кепілдік береді.

Жүйе үшін сенімділіктің үш түрлі функционалдық талаптары болады:

1. *Талаптарды тексеру*. Бұл талаптар бірдейлендіруді өткізеді, жүйеге кіріспе параметрлерді тексереді, өйткені ауқымы диапазоннан шет жататын немесе теріс кепілдік беретін кіріспе параметрлерінің жүйені өңдеуге дейін олардың мәлім болатынын айқындайды.
2. *Қалпына келтірудің талабы.* Бұл талаптар іркілістен кейін өз қалпына келу үшін жүйеге көмек беруге бағытталған. Әдетте, бұл талаптар жүйенің көшірмесі мен мәліметтерді сақтауға және жүйенің сервистерін іркілістен кейін қалпына келтіру тәсілін көрсетуге бағытталады.
3. *Сақтық қор талаптары.* Бұл талаптар бір бөліктің істен шығуы қызмет көрсетудің толық ысырабына алып келмеген жүйенің артық функцияларын анықтайды. Толық ақпарат астамын келесі тарауда оқыңыз.

Одан басқа сенімділіктің талабы сенімділік үшін жасалған үдерістің талабынан тұрады. Бұл талаптардың қажеттілігі жүйенің даму үдерісінде жүйедегі қателер санын азайту үшін тиімді тәжірибе кепілдік беретініне пайдалану үшін өте қажет. Функционалдық сенімділіктің кейбір мысалдарын және үдерістің талаптарын *12.8-суреттен* қараңыз.

RR1: Барлық операторлардың аралығы алдын ала анықталуы керек. Барлық операторлар тек қана осы аралықта болуы тиіс.
RR2: Науқастардың барлық деректері 2 серверде сақталуы тиіс. Егер бір сервер сәтсіздікке ұшыраса, деректер екінші серверде сақталып қалады.
RR3: Бұзылуды бақылайтын жүйе N-нұсқалы бағдарламалау тілінде әзірлену керек.
RR4: Жүйе Аданың қауіпсіз жиынында әзірленеді және статикалық анализдермен тексеріледі.

12.8-сурет. Функционалды сенімділік талаптарының сипаттамасы

Сенімділіктің функционалдық талаптарын алуда ешқандай да қарапайым ереже-тәсіл жоқ. Сыни жүйелерді өндейтін ұйымдарда, әдетте, болуы мүмкін сенімділік талаптары туралы ұйымдастырылған білім болады және олар қалай жүйенің нақты сенімділіктеріне әсер ететіні туралы мағлұматтар бар. Бұл ұйымдар жүйенің белгілі түрлеріне мамандандыра алады (мысал, теміржол басқару жүйелері), сондықтан сенімділіктің талабын көптеген басқа жүйелерде де қайтадан пайдалануға болады.

12.4. Қауіпсіздіктің спецификациясы

Қауіпсіздік талаптарының спецификацияларында ортақ қауіпсіздік техникасының талаптарымен ортақ жерлері болады. Оларды сандай анықтау өте тиімсіз, ал қауіпсіздік талаптары көбінесе, «қабылдауға жарамайтын» талаптар болып келеді, керекті жүйелік функционалдықты емес, істеуге болмайтын жүйелік мінез-құлықты анықтаған талаптар болып көрінеді. Алайда, қауіпсіздік – көп мәселелерге қорғаныс жағдайын жасағанша ынталандырушы мәселе, мысалы:

1. Сіз қорғанысты қарап отырып, жүйе орнатылған қоршаған орта жаулардың ортасы емес екенін байқайсыз. Ешкім және ештеңе, қақтығыс шақыруға азаптандырмайды. Сіз жүйеге қарай шабуыл жасау қасақана болжау екенін және шабуыл жасаушыда жүйелік әлсіздіктердің бар болуы мүмкін екенін түсінесіз.
2. Егер қауіпсіздіктің бұзылысына әкелетін жүйелік қателер болса, сіз ол қателерді тауып және ол қателерді түзетуіңіз керек. Қасақана шабуылдар жүйелік іркіліске әкеліп соққанда, онда шабуылшы жасалған қателіктердің іздерін қалдырмауы мүмкін, сондықтан іркілістің себебін табу үдерісі қиындап қалуы мүмкін.
3. Жүйенің өшірілуі немесе жүйелік қызмет көрсетуінің нашарлауы әдетте, қателерден аман болу үшін қауіпсіздікке қатысты қолайлы әрекет болып көрінеді. Алайда, «қызмет көрсетуді қабыл алмау» жүйеге шабуыл жасауға арналған шабуыл болуы мүмкін, оның мақсаты – жүйені өшіру. Жүйенің өшірілуі шабуыл табысты болғанын білдіреді.
4. Қауіпсіздікке қатысты оқиға ақылды жаумен жасалмаған. Шабуыл жасаушы шабуылдардың топтамасында жүйенің қорғаныс қабілетін зерттеп, шабуылды өзгерте отырып, жүйе туралы сұрақтарға көп жауап ала алады.

Бұл айырмашылықтар қауіпсіздік талабы қауіпсіздік техникасының талабына қарағанда әдетте, көлемдірек болу керек екенін білдіреді. Қауіпсіздік техникасының талабы көптеген фунционалдық жүйенің талаптарына әкеледі, жайымсыз оқиғаларға және іркілістерге қарсы қорғанысты қамтамасыз ететін функционалдық жүйелік талаптардың қатарына алып келеді. Бұл мәселелер туған жағдайда негізінен олар мәселенің тексерісімен және оған қарсы қойылатын шаралармен беймазаланады. Қауіп-қатерге қарсы тұратын жүйеде кездесетін әртүрлі қауіпсіздік талаптарының түрлері болады. Файрсмит (2003) жүйелік спецификацияға қосылған қауіпсіздік талаптарының 10 түрін сәйкестендірді:

1. Теңестіру талаптары жүйе өз қолданушыларын олармен өзара іс-әрекет жасаудың алдында сәйкестендірілу керектігін анықтайды.
2. Танып-айыру талаптары қолданушыларды сәйкестендірудің қалай болғанын анықтайды.

3. Рұқсаттың талабы артықшылықты және белгіленген қолданушыларды қол жеткізуге рұқсат алуын анықтайды.
4. Қолсұқпаушылықтың талабы жүйе вирустарынан, құрттардан және соған ұқсас қатерлерден қалай сақтану керек екенін анықтайды.
5. Бүтіндіктің талабы мәліметтердің бүлінуін қалай қорғау керек екенін анықтайды.
6. Басып кірудің табылуын анықтайтын талап, жүйеге жасалған шабуылды табу үшін қандай тетіктер пайдалануы керек екенін анықтайды.
7. Мүлтіксіздік талаптары мәміледегі осы мәмілеге өз қатысы барлығын теріске шығара алмайтынын анықтайды.
8. Жеке өмір сүрудің талабы жасырын құпия сақтауды анықтайды.
9. Қауіпсіздіктің тексеріс талабы жүйелік пайдалану қалай тексерілетінін анықтайды.
10. Қауіпсіздік жүйесіне техникалық қызмет көрсету механизміндегі рұқсат етілген өзгерістерді кездейсоқ бұзылудан қалай сақтау керек екенін анықтайды.

Қауіпсіздіктің тәуекел менеджменті

Қауіпсіздік бұл құқықтық мәселе болып табылады және бизнестер қауіпсіздік жүйесіне қатыспау туралы шешім жасай алмайды. Алайда, қауіпсіздік саласының кейбір аспектілері бизнес мәселелеріне жатады. Бизнес кейбір қауіпсіздік шараларын орындамауына қатысты шешім жасай алады және ол осы шешімінің салдарынан болуы ықтимал барлық шығындарды өтейді. Тәуекел менеджменті қандай мүліктің қорғалуына және оларды қорғауға қанша қаражаттың жұмсалуын белгілеуге қатысты мәселелерді шешу үдерісі болып табылады.

http://www.SoftwareEngineering-9.com/Web/Security/RiskMan.html

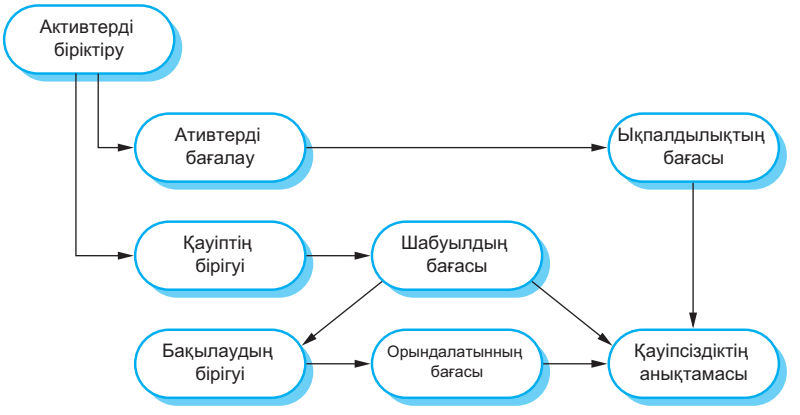
Әрине, сіз әр жүйеде қауіпсіздік талаптары бұл түрлерінің бәрін көре алмайсыз. Ерекше талаптар жүйенің тұрпаттарына, күткен қолданушыларға және қолдану жағдайларына бағынышты болады.

12.1-бөлімінде жүйенің қауіпсіздік талаптарын бірдейлестіруде қарастырылған тәуекел дәрежесінің талдауы және бағаның үдерісі көрсетілген. Мен жоғарыда айтып кеттім, бұл үдеріске тиелі үш кезең болады:

1. *Тәуекел дәрежесінің алдын ала жобалап жасалған анализ.* Бұл кезеңде толық жүйелік талаптар туралы шешім немесе енгізудің технологиясы және жүйенің жобасы жайлы шешім жасауға болмайды. Бұл – баға үдерісінің мақсаты жүйе үшін қауіпсіздік талабын негізінен алу.
2. *Тәуекел дәрежесінің өмірлік циклінің анализі*. Бұл өмірлік циклді бағалау дизайнды таңдағаннан кейін жүйелік өмір циклінде орын алады. Қосымша

қауіпсіздік талаптары жүйені жобалау құрылысында пайдаланылатын технология және енгізудің шешімдері ескеріледі.

3. *Тәуекел дәрежесінің қолдану кезіндегі талдауы.* Бұл баға қолданушылардың операциялық жүйеге шабуыл жасаған кезде біреудің көмегімен не болмаса көмексіз пайда болатын қауіп-қатерлерді ескереді.



12.9-сурет. Қауіпсіздік талаптардың тәуекелін бағалау үдерісі

Қауіпсіздік талаптарының спецификациясында пайдаланылатын тәуекел дәрежесі бағасының үдерістері және анализі. Олар *12.1-бөлімінде* талқыланған спецификация үдерісінің нұсқалары болып көрінеді. Қауіпсіздік талаптарының тәуекелінде негізделген үдеріс *12.9-суретте* көрсетілген. Бұл суретін *12.1-суретінен* айырмашылығы бар болып көрінеді, бірақ мен онда әр кезең иерархия кезеңіне сәйкес келгенін дәлелдеп көрсеттім. Үдерістің кезеңдері:

1. *Активті бірдейлестіру*. Мұнда қорғаныс қажет ететін жүйелік активтер бірдейлестірілген. Жүйенің өзі немесе жүйелік функциялардың ерекшеліктері актив сияқты, дәл солай жүйемен байланысқан активтер (тәуекелді бірдейлестіру).
2. *Активтер құнын бағалау*. Сіз бұл жерде активтердің құндылығын бағалап, оларды бірдейлестіресіз (тәуекел дәрежесінің анализі).
3. *Әсер етудің бағасы*. Сіз әр активпен ысырап болу мүмкіндігін бағалайсыз. Сіз олардың ақпаратты ұрлау, қалпына келтірудің шығындары және (тәуекел дәрежесінің талдауы) абыройдың ысырабы болуы мүмкін екенін ескеруіңіз керек.
4. *Бірдейлестірудің қатері*. Сіз (тәуекел дәрежесінің талдауы) жүйелік активтерге төнген қауіп-қатерді бірдейлестендіруіңіз керек.
5. *Шапқыншылықтың бағасы*. Сіз барлық төнген қауіптерді шабуылға аударасыз да, ол шабуылдардың шыққан жолдарын талдайсыз. Сіз шабуылдарды талдау үшін (Шнайер, 1999) шабуыл ағаштарын пайдалана аласыз. Бұл қадам қателердің талшыбықтарына ұқсас, себебі сіз 0 қауіп-қатерді ағаштың

тамырынан бастайсыз және болуы мүмкін шабуылдарды сәйкестендіресіз және олар қалай (тәуекелдің жіктеуі) жасалғанын тексере аласыз.

6. *Бақылауды бірдейлестіру*. Сіз активті қорғауда қолданылатын басқарудың амал-әрекеттерін ұсынасыз және активтерді қорғауда пайдалана алатын шифрлау әдісін (тәуекелді төмендету) көрсетесіз.
7. *Орындалатындықтың бағасы*. Сіз техникалық орындалатындықты және ұсынылған басқару құралдарының шығындарын бағалайсыз. Активтерді қорғауда қымбат бағалы құндылығы жоқ әрекеттерді қолданудың қажеті жоқ (тәуекелді төмендету).
8. *Қауіпсіздік талаптарының анықтамасы*. Қауіпсіздік талабын алу үшін әсер етудің, қорқытудың және бақылаудың бағасын берудің білімдері қолданылатын қауіпсіздік талаптарының анықтамасы. Олар жүйелік инфрақұрылым үшін немесе қолданбалы жүйе үшін талап бола алады (тәуекелді төмендету).

Тәуекел дәрежесінің бағасына маңызды салым және басқару үдерісі – қауіпсіздікті ұйымдастыру саясаты. Қауіпсіздіктің ұйымдастыру саясаты барлық жүйелерге жатады және онда не істеуге болады, нені істеуге болмайтынын мазмұндауы керек. Мысалы, қауіпсіздіктің әскери саясатының бір аспектісі – «Оқырмандар тек қана олардың жіктесу деңгейіне сәйкес келетін құжаттарды зерттей алады». Бұл егер оқырман 'жасырын' деңгейде болса, онда ол «құпия» құжаттарға қолжетімділікте болады, ол құжаттар «құпиялы», «ашық», «жасырын» немесе «өте құпиялы» болып жіктеледі.

Қауіпсіздіктің саясаты қауіпсіздікті қамтамасыз ететін жүйені әрқашан сақтау керек болатын шарт орнатады және пайда болуы мүмкін болатын қатерлерді бірдейлестіруге көмектеседі. Қорқыту – іскер қауіпсіздікке қатер төндіретін бірнәрсе. Әдетте, қауіпсіздік саясаты іс жүзінде бейресми құжаттар, олар қай нәрсеге рұқсат етілген, қай нәрсеге рұқсат етілмегенін анықтайды. Алайда, Бишоп (2005) қауіпсіздіктің саясатын формалды тілде үстірт мүмкіндігін талқылайды және автоматталған тексерістерді өндіру үшін саясаттың бағытына сәйкестіктің кепілдігін береді.

Қауіпсіздіктің қатеріне анализ жасаған үдерісті суретпен көркемдеу үшін, «MHC-MPS» ауруханасының писхикалық денсаулыққа қамқорлық көрсететін ақпараттық жүйесін қарап шығыңыз. Мен оны есептеме бөлігі ретінде (*12.10*-және *12.11-суреттер*) алдын ала болатын үдерістерден жасалған дәреженің бағасы ретінде көрсеттім. Бұл алдын ала есептеу қауіпсіздік талаптарының анықтамасында пайдаланылады. Толық бағасын мұнда талқылауға менде орын жоқ, бірақ мен бұл жүйені мысал ретінде қарастырамын.

Сіз ауруханаақпараттық жүйе үшін тәуекелдердің талдауынан қауіпсіздік талабын ала аласыз. Бұл талаптардың кейбір мысалдары:

1. Емделушісі туралы ақпарат дерекқор мәліметтерден бастап жүйелік клиенттегі қорғалатын орынға дейін, сессиясының басында жүктелген болуы керек.

2. Емделуші туралы барлық ақпарат жүйелік клиентте шифрлануы керек.
3. Ақпарат жүктелген дерекқорда болуы керек, клиника сессиясы біткен кезде клиенттің компьютерінен алып тасталынады.
4. Жүйелік дерекқорға енгізілген барлық өзгерістерді тіркеу және бұл өзгерістерді бастаушымен бірге сервер дерекқорынан бөлек компьютерде жеке сақталуға тиіс.

Баға	Көлем	Үзінді
Ақпараттық жүйе	Жоғары. Барлық клиникалық кеңестерді демейді. Қауіпсіздігі өте маңызды.	Жоғары. Клиника өзгерген жағдайында ақшалай шығындарға алып келеді. Жүйені қайта құрастыру шығындары. Бұзылған жағдайында тұтынушы шығындарына әкеледі.
Науқастардың дерекқоры	Жоғары. Барлық клиникалық кеңестерді демейді. Қауіпсіздігі өте маңызды.	Жоғары. Клиника өзгерген жағдайында ақшалай шығындарға алып келеді. Жүйені қайта құрастыру шығындары. Бұзылған жағдайында тұтынушы шығындарына әкеледі.
Науқастардың жеке жазылулары	Әдеттегідей төмен. Кейбір жоғарғы дәрежелі науқастарға жоғары болуы мүмкін.	Тікелей шығындары аз. Бірақ, абыройдың жоғалуына жетелеуі мүмкін.

12.10-сурет. MHC-PMS-тің бағалы есеп беру анализі

Қауіп	Ықшамдылық	Бақылау	Орындалу
Кез келген пайдаланушы жүйедегі басқарушының рөліне ие болып, жүйені қолжетімсіз етеді.	Төмен	Жүйені басқаруға шектелген әрі қауіпсіз мекендерден ғана мүмкіншілік беру	Әзірлену шығындары аз. Арнайы кілтпен шифрланады да, төтенше жағдайларда кілт қолжетімді болады.
Кез келген пайдаланушы жүйедегі жеке ақпараттарға қол жеткізеді.	Жоғары	Барлық пайдаланушылардың жүйеге өзінің жеке кілті арқылы кіруін әзірлеу. Науқастардың жеке мәліметтерінің өзгеруін жазып алу.	Техникалық тұрғыдан қарағанда шығыны көп болады. Қарапайым әзірленеді және бұрынғы қалпына қайта келтірілуді қолдайды

12.11-сурет. Қауіп тәуекелінің есеп беруінің бақылау анализі

Егер дерекқор сервері шабуылға ұшыраса немесе қолжетпес болса, емделуші туралы ақпараттар кеңес алуға кедергі жасамай, әрі қарай жалғастыра беруге болатындай жергілікті машинаға жүктелу керек. Алғашқы екі талап бір-бірімен байланған. Алайда, бұл ақпараттар кешірек келген қолданушылар мәліметтерге қолжеткізе алмайтындай болып алып тасталынуы керек. Төртінші талап – қалпына келтірудің және тексерудің талабы. Бұл өзгерістерді тауып кіргізу мүмкіндігі бар екенін және өзгерулер журналы қайта оңай қалпына келтірілетін бола алатынын білдіреді. Бұл жауапкершілік уәкіл штаттық жүйені теріс қолдамауына кедергі келтіреді.

12.5. Формалдық спецификация

30 жылдан астам бойы, көптеген зерттеушілер бағдарламалық қамтамасыз ету әзірлеуінің үстірт әдістерін пайдалануды қолдауды қалайды. Бағдарламалық қамтамасыз етудің үстірт үлгісін анықтауға болатын формалды үлгілер математикалық тәсілдер болып табылады. Содан кейін бұл үлгілерді талдауға және оны жүйенің үстірт спецификацияның негізі ретінде пайдалануға болады. Негізінен, бағдарламалық қамтамасыз етудің формалдық үлгісінен бастауға болады және игерілген бағдарлама үлгімен сәйкестендіріліп, қателердің нәтижесінде бағдарламалық қамтамасыз етудің іркілісі тумайтынын дәлелдеу керек.

Барлық үстірт формалдық даму үдерістерінің дамуы үшін жүйенің формалды үлгісі спецификация ретінде қызмет көрсететін жүйенің үстірт үлгісі болып көрінеді. Бұл үлгіні жасау үшін, сіз өзін табиғи қалпында ұстайтын диаграммалар, кесте жүйелері және математикалық тілде қолданылатын талаптарды аударасыз. Жүйенің не істеу керек екенін бірмәнді сипаттайтын формалдық спецификация болып табылады. Сіз қолмен істейтін әдістерді немесе аспапты әдістерді қолдана отырып, бағдарламаның мінез-құлқының спецификацияға сәйкес келгеніне көз жеткізе аласыз.

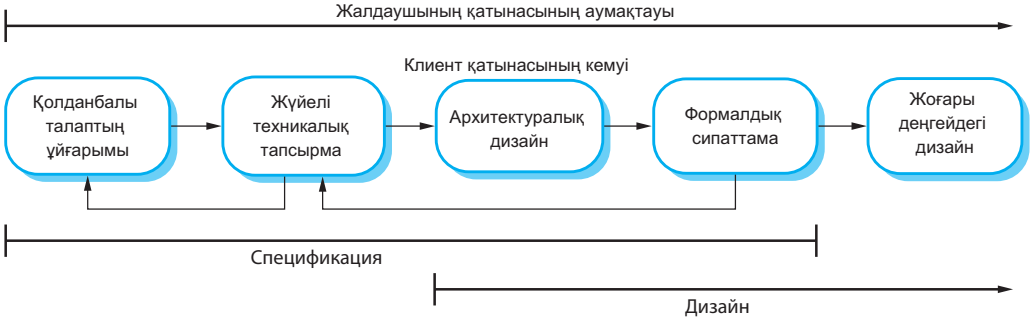
Формалдық спецификация техникалары

Формалдық жүйе спецификациялары жүйе интерфейстерінің үлгілері (алгебралық спецификациялар) немесе жүйе күйінің үлгілері сияқты екі фундаменталды тәсілін пайдалану арқылы берілуі мүмкін. Осы тақырып бойынша қосымша торап тарауын жүктеп ала аласыз. Ол жерде мен осы екі тәсілге мысалдар келтіргенмін. Тарау инсулин сорғысының жүйесіндегі бөліктің формалдық спецификациясы қамтылған.

http://www.SoftwareEngineering-9.com/Web/ExtraChaps/FormalSpec.pdf

Формалды үстірт спецификация әзірлеудің тексерісі және енгізілуі бағдарламалық жасақтамаға ғана негіз емес. Демек, олар жүйедегі анықтама тәсілінің ең дәл тәсілі болып табылады және түсінбеушілік жағдайды қысқартады. Одан басқа, үстірт спецификациясының құрылысы толық талдаудың талаптарын қамтамасыз етеді және бұл талаптардың мәселелерін табудың тиімді тәсілі саналады. Спецификациялардың табиғи тілінде тілдің дәлсіздігінен қателер көрінбей қалады. Бұл жүйенің ресми анықталынған жағдайы емес.

Әдетте, үстірт спецификациялар жүйе әзірлеуге дейін толық анықталатын жоспарлы бағдарламалық қамтамасыз етудің үдерісті шеңберінде жасалады. Жүйелі талаптар және әзірлеу бөлшектерімен анықталады және мұқият талдау жүргізіледі және жүзеге асырудан бұрын тексеріледі. Егер бағдарламалық қамтамасыз етудің үстірт спецификациясы жасалған болса, онда бұл жағдай жүйелік талаптарды анықтағаннан кейін болған, бірақ жүйенің толық жобалауына дейін. Үстірт спецификациясы мен формалдық спецификацияның ортасында, яғни талаптардың толық спецификацияның аралығында қатты кері байланыс тұзағы болады.



12.12-сурет. Жобалаумен әзірлеудің қалыпты сипаттамасы

12.12-сурет бағдарламалық қамтамасыз ету спецификациясының кезеңдерін және бағдарламалық қамтамасыз етудің жоспарлы үдерісінде әрлендіруі бар оның интерфейсін көрсетеді. Үстірт спецификациясы қымбат болғандықтан, сіз жүйенің қызмет етуі үшін шешуші мағынаға ие болған құрамдас бөлік үшін бұл тәсілді пайдалануды шектей аласыз. Сіз жүйенің архитектуралық жобалауында бұл жағдайды анықтай аласыз.

Соңғы бірнеше жылдарда үстірт спецификацияның талдауына автоматты қолдау жасалды. Clarke (Clarke және т.б., 2000) пішінді тексерістің қаржысы жүйе үлгісінің бастапқы негізінде үстірт спецификациясы болатын программалық құралдармен қатар кейбір үстірт спецификациямен бейнеленген ерекшеліктерді ұсынады, мысалы, «қолжетпейтін жағдайлар болмайды» деген сияқты. Пішінді тексеріс бағдарлама спецификацияны жеткілікті талдайды және жүйе үлгіге сәйкес келгенін хабарлайды немесе оған сәйкес келмегенін көрсететін мысалдарды ұсынады. Статикалық талдау тексеріс ұғым үлгілерімен тығыз байланысқан және мен осы тексерістің жүйесіне ортақ тәсілдерді *15-тарауда* айтып өттім.

Үстірт спецификацияларының әзірлеу және үстірт даму үдерісінде де пайдаланудың келесі артықшылықтары болады:

1. Бөлшектердегі үстірт спецификацияның дамуына сай, сіз тереңірек жетілдіре түсесіз және жүйелік талаптарды толық түсінушілікті жетілдіресіз. Егер сіз үстірт дамудың формалдық спецификациясын пайдаланбаған күннің өзінде, талаптардың қателерін үстірт даму үдерісінде анықтап тауып алу (Hall, 1990) үстірт спецификацияның даму пайдасына күшті дәлел екенін көресіз. Әзірлеу үдерісі кезінде талаптардың ерте табылған мәселелері кеш табылған мәселелерге қарағанда, оларды түзету анағұрлым арзандау болады.
2. Спецификация үстірт белгілі семантика тілінде байқалғандықтан, толымсыздықты және үйлеспеушіліктерді іздестіруде автоматты түрде анализ жасау керек.
3. Егер сіз B әдісі сияқты әдісті пайдалансаңыз, сіз дәлдік сақтайтын үстірт спецификациялар тізбегі арқылы формалдық спецификаларды бағдарламаға айналдыра аласыз. Алынған спецификацияның сәйкес келуі бағдарламаның нәтижесіне кепілдік береді.
4. Бағдарламаның спецификациясын тексеруде бағдарламаны тестеу шығындарын азайта аласыз.

Үстірт әдістер бұл артықшылыққа қарамастан бағдарламалық қамтамасыз етудің жаттығу әзірлеуіне шектелген әсерге ие болады, тіпті, сыни маңызды жүйелер үшінде. Демек, әзірлеу кезінде және оны қолдану кезінде жүйені үстірт спецификациялы пайдалануда тәжірибе өте аз. Жүйенің үстірт спецификацияның дамуына қарсы келесі аргументтер ұсынылған:

1. Мәселенің иелері және мәселе аймақтарындағы мамандар үстірт спецификацияны түсіне алмағандықтан, олардың талаптарына дәл жауап бергенін тексере алмайды. Үстірт спецификацияны түсінген бағдарламалық қамтамасыз етудің өндеушілері мүмкін, қолданбалы доменді түсінбейді, сондықтан олар үстірт спецификация жүйе талаптарының дәл шағылысу дәлелі болып табылатынына сене алмайды.
2. Бұл жердегі үстірт спецификацияның жасауындағы сан шығынын әжептәуір жеңіл анықтауға болады, бірақ оны пайдалану нәтижесінде пайда болатын қаражат үнемдеуді бағалау аса қиын. Нәтижесінде, менеджерлер мұндай тәсілді қабылдауға өз жауапкершіліктеріне алғысы келмейді.
3. Бағдарламалық қамтамасыз етудің көп инженерлері көпшілік үстірт спецификация тілдерін пайдалануға оқытылмаған. Демек, олар оны әзірлеу үдерістерінде пайдалануға қарсы.
4. Жаңа тәсілдер мен үстірт спецификацияларды өте үлкен жүйеде түйістіру қиынға түседі. Үстірт спецификация бүтін жүйе емес программалық кодтың сыни бөліктерінің анықтамасы үшін пайдаланады.
5. Үстірт спецификация әзірлеудің иілгіш әдістерімен үйлесімді емес.

Формалдық спецификация шығындары

Формалдық спецификацияны дамыту қымбат жоба, себебі талаптарды формалдық тілге аударуға және спецификацияны тексеруге айтарлықтай көп уақыт кетеді. Тәжірибе жүйе сынағы мен растау әрекеттерінен біраз жинақтаулар жасауға болатындығын көрсетті. Формалды түрде жүйені спецификациялау барлық зерттеу шығындарын өсірмейтіндей көрінеді. Бірақ шығыны жоғары шығындар өзгерісінің теңгерімі зерттеу үдерісінің басында шығынға ұшырап үлгерді.

http://www.SoftwareEngineering-9.com/Web/FormalSpecCosts/

Алайда, осы мақаланы жазудың алдында қорғаныс және қауіпсіздіктің маңызды қосымшаларының қатарын әзірлеуде формалдық әдістер қолданылған болатын. Олар да әзірлеуде және ірі де және күрделі программалық жүйелерде экономикалық жағынан тиімді жұмсала алады және сыни бөліктерді валидациялауда жұмсалады (Badeau and Amelot, 2005; Hall, 1996; Hall and Chapman, 2002; Miller et al., 2005; Wordworth, 1996). Олар статикалық анықтауда қолданатын Ball Microsoft драйверлерінің Ball Microsoft сияқты пайдаланылатын тексерісі жүйесінің (Ball және т.б. 2004; Ball et al., 2006) және сыни инженерлік жүйелерінің (Barnes, 2003) SPARK/Ada негізі болып табылады.

НЕГІЗГІ ТҰЖЫРЫМДАР

- Қауіп-қатерді талдап анализ жасау қауіпсіздік спецификациясының және сенімділіктің талаптарының маңызды бағыты болып көрінеді. Ол қатеге немесе апатқа әкелетін қауіп-қатерлердің анықтамасынан.
- Тәуекел-бағдарланған тәсіл жүйе қауіпсіздігіне қойылатын талаптарды түсіну үшін қолданылады. Сіз потенциалдық қауіп-қатерлерін сәйкестендіріңіз және (қателерді талшыбық әдісімен талдауды қолданып) оларды талдаңыз. Содан кейін ол мәселерден құтылу үшін немесе мәселені жөндеу үшін талаптар қоя аласыз.
- Сенімділікке қойылатын талап жүйеге қойылатын талаптардың спецификациясына анықталуы мүмкін. Сенімділіктің көрсеткіштеріне (ВНТП) істен шығу ықтималдығының талабы (ТВН), жеткіліксіздіктің пайда болуының жиілігі және (ДОСТ) қолжетімділік кіреді.
- Жүйенің талап етілетін сенімділігінің маңызын артық бағаламау, өйткені бұл жағдай әзірлеп-дайындау кезінде керексіз қосымша шығындарға алып келеді және үдерістердің валидациясына әкеледі.

- Қауіпсіздік талабын сәйкестендіру қиындау, өйткені бұзушылар жүйеге шабуыл жасау жоспарының осалдықтарын пайдалана отырып, сәтсіз шабуылдардан басқа да осалдықтар туралы біле алады.
- Қауіпсіздік талаптарының анықтау үшін сіз қорғалған активтерді анықтауыңыз және осы активтерді қорғау үшін әдістер мен қауіпсіздік технологиясы қалай қолданылуы керек екенін анықтауыңыз қажет.
- Бағдарламалық қамтамасыз етудің формалдық әдістері жүйенің спецификацияларына сүйенетін математикалық модель түрінде болады. Үстірт спецификацияның әзірлеудің толық зерттеуді ынталандыруы маңызды артықшылық болады және жүйеге талаптарды талдайды.

ҚОСЫМША ӘДЕБИЕТТЕР

Safeware: System Safety and Computers. This is a thorough discussion of all aspects of safety-critical systems. It is particularly strong in its description of hazard analysis and the derivation of requirements from this. (N. Leveson, Addison-Wesley, 1995.)

'Security Use Cases.' A good article, available on the Web, that focuses on how use cases can be used in security specification. The author also has a number of good articles on security specification that are referenced in this article. (D. G. Firesmith, *Journal of Object Technology,* **2** (3), May–June 2003.) http://www.jot.fm/issues/issue200305/column6/.

'Ten Commandments of Formal Methods . . .Ten Years Later.' This is a set of guidelines for the use of formal methods that was first proposed in 1996 and which are revisited in this paper. It is a good summary of the practical issues around the use of formal methods. (J. P. Bowen and M. G. Hinchey, *IEEE Computer*, **39** (1), January 2006.) http://dx.doi.org/10.1109/MC.2006.35.

'Security Requirements for the Rest of Us: A Survey.' A good starting point for reading about security requirements specification. The authors focus on lightweight rather than formal approaches. (I. A. Tøndel, M. G. Jaatun, and P. H. Meland, *IEEE Software*, **25** (1), January/February 2008.) http://dx.doi.org/10.1109/MS.2008.19.

ЖАТТЫҒУЛАР

12.1. Түсіндіріңіз, неге *12.12-суретте* көрсетілген тәуекелдің үшбұрышында уақыт өткен сайын оның шекаралары өзгеріске ұшырайды?

12.2. Түсіндіріңіз, неліктен қауіпсіздікке негізделген анықтау уақытындағы тәсіл тәуекел әртүрлі түсіндіріледі.

12.3. Инсулин сорғысында жүйесінде қолданушы инені өзгертуі керек және инсулинның жабдықтауын бір қалыпты және мүмкіндігінше жалғыз және ең жоғары мөлшерді өзгертуі және күн сайынғы ең жоғары мөлшерді өзгертуді басқара алады. Үш түрлі қолданбалы қателерді және осы қателерден құтылуға көмектесетін қауіпсіздік техникасының талабын ұсыныңыз.

12.4. Жүйенің қауіпсіздігі бағдарламалық қамтамасыз етуде сыни қатысты рак ауырымен ауыратын науқастарды қарау үшін екі негізгі құрамдас бөліктері болады:

- ісік бар жерге радиация мөлшерін жеткізетін радиациялық терапияның машинасы. Бұл машинамен кіріктірме бағдарламасын қамтамасыз ететін жүйені басқарады.
- әр емделуші үшін емнің ерекшелігін қосқан дерекқор. Емнің талабы бұл дерекқорға енгізілген және ол радиациялық терапияның машинасына автоматты түрде жүктеледі.

Бұл жүйеде пайда болатын үш түрлі қауіп-қатерді сәйкестендіріңіз. Әр қауіп-қатер үшін бұл қауіп-қатерлерді қорғайтын талаптарды ұсыныңыз, олар апатқа әкелетін ықтималдықты азайтады. Сіздің ұсынған қорғанысыңыз қауіп-қатерге қатысты тәуекелді азайтатынын түсіндіріңіз.

12.5. Бағдарламалық қамтамасыз ету жүйесінің кластары үшін сенімділіктің тиісті көрсеткіштерін ұсыныңыз. Өз таңдауыңыздың себептерін түсіндіріңіз. Бұл жүйелерді қалай пайдалануға болатынын болжаңыз және сенімділіктің көрсеткіштері үшін тиісті құндылықтарды ұсыныңыз.

- емделушілерді ауруханаңың қарқынды терапиясының бөлімінде бақылайтын жүйе;
- мәтіндік процессор;
- саудалық автомат басқаратын автоматтандырылған басқару жүйесі;
- автокөлікте тоқтатуды басқаратын жүйе;
- тоңазытқыш қондыру басқаратын жүйе;
- басқарушылық есепті өндіргіш.

12.6. Егер сіздің сегменті үшін жылдамдыққа шек қою шектен шықса, пойыздың қорғанысының жүйесі автоматты түрде тежеуішті шақыртады немесе, егер пойыз сегмент ізіне кірсе, бұл уақытта ол қызыл жарыққа енгізіліп, хабарлайтын іздің сегментіне кіреді (сегмент қауіпті болмауы керек). Сіздің сұрақтарыңызға жауап беру үшін сенімділіктің көрсеткішін таңдаңыз, мұндай жүйе үшін қажетті сенімділікті анықтау керек.

12.7. Пойыздың қорғаныс жүйе үшін қауіпсіздік техникасының екі маңызды талаптары бар:

- пойыз қызыл жарықпен хабарланған із сегментіне кірмейді.
- пойыз іздің бөлімі үшін көрсетілген жылдамдыққа қойылған шектен аспайды.
- Сигнал мәртебесі және бағдарламалық қамтамасыз етуге із сегменті үшін жылдамдыққа шек қойылғанын борттық бағдарлама камсыздандыруына пойыз із сегментіне кіруден бұрын енгізілуін ескере отырып, қауіпсіздік техникасының жүйелік талаптарынан жасалатын бағдарламалық қамтамасыз етудің бес түрлі функционалдық жүйелік талаптарын ұсыныңыз.

12.8. Қауіпсіздік қатерінің алдын ала бағалау қажеттілігі бар екенін және жүйенің дамуы кезінде өмірлік циклдің қауіпсіздігіне қатер төнуді бағалауды түсіндіріңіз.

12.9. *12.11-суреттегі* кестені кеңейтіңіз, сонда басқарудың байланған қаржыларымен қатар MHC-PMS тағы екі қатерді сәйкестендіре аласыз. Басқарудың ұсынылған қаржыларын жүзеге асыратын бағдарламалық қамтамасыз етудің одан әрі қауіпсіздік талаптарын жасау үшін оларды негіздеуді пайдаланыңыз.

12.10. Спецификация мен жүйе қауіпсіздігіне қатысты дамуымен бағдарламалық қамтамасыз етудің өңдеушілері кәсіби жоспарда сертификатталуы керек пе? Өз пікіріңізді түсіндіріңіз.

ПАЙДАЛАНЫЛҒАН ӘДЕБИЕТТЕР

Badeau, F. and Amelot, A. (2005). 'Using B as a High Level Programming Language in an Industrial Project: Roissy VAL'. Proc. ZB 2005: Formal Specification and Development in Z and B, Guildford, UK: Springer.

Ball, T., Bounimova, E., Cook, B., Levin, V., Lichtenberg, J., McGarvey, C., Ondrusek, B., Rajamani, S. K. and Ustuner, A. (2006). 'Thorough Static Analysis of Device Drivers'.Proc. EuroSys 2006, Leuven, Belgium.

Ball, T., Cook, B., Levin, V. and Rajamani, S. K. (2004). 'SLAM and Static Driver Verifier: Technology Transfer of Formal Methods Inside Microsoft'. Proc. Integrated Formal Methods 2004, Canterbury, UK: Springer.

Barnes, J. P. (2003). *High-integrity Software: The SPARK Approach to Safety and Security*. Harlow, UK: Addison-Wesley.

Bishop, M. (2005).*Introduction to Computer Security*. Boston: Addison-Wesley.

Brazendale, J. and Bell, R. (1994). 'Safety-related control and protection systems: standards update'. *IEE Computing and Control Engineering J.*, **5** (1), 6–12.

Clarke, E. M., Grumberg, O. and Peled, D. A. (2000).*Model Checking*. Cambridge, Mass.: MIT Press.

Firesmith, D. G. (2003).'Engineering Security Requirements'.*Journal of Object Technology*, **2** (1), 53–68.

Hall, A. (1990). 'Seven Myths of Formal Methods'.*IEEE Software*, **7** (5), 11–20.

Hall, A. (1996). 'Using Formal methods to Develop an ATC Information System'. *IEEE Software*, **13** (2), 66–76.

Hall, A. and Chapman, R. (2002).'Correctness by Construction: Developing a Commercially Secure System'. *IEEE Software*, **19** (1), 18–25.

Jahanian, F. and Mok, A. K. (1986).'Safety analysis of timing properties in real-time systems'.*IEEE Trans.on Software Engineering*.,**SE-12** (9), 890–904.

Leveson, N. and Stolzy, J. (1987).'Safety analysis using Petri nets'.*IEEE Transactions on Software Engineering*, **13** (3), 386–397.

Leveson, N. G. (1995). *Safeware: System Safety and Computers*. Reading, Mass.: Addison-Wesley.

Miller, S. P., Anderson, E. A., Wagner, L. G., Whalen, M. W. and Heimdahl, M. P. E. (2005).'Formal Verification of Flight Control Software'.*Proc. AIAA Guidance*, Navigation and Control Conference, San Francisco.

Peterson, J. L. (1981). *Petri Net Theory and the Modeling of Systems*. New York: McGraw-Hill.

Schneier, B. (1999).'Attack Trees'.*Dr Dobbs Journal*, **24** (12), 1–9.

Storey, N. (1996).*Safety-Critical Computer Systems*. Harlow, UK: Addison-Wesley.

Wordsworth, J. (1996). *Software Engineering with B*. Wokingham: Addison-Wesley.

13.

Инженерлік іс сенімділігі

Мақсаттары

Осы тараудың мақсаты – оқырманды жүйенің жоғары сенімді зерттемелік үдерістерімен және әдістерімен таныстыру. Тарауды оқып болғаннан кейін сіз:

- артықтық және түрлілік компоненттердің арқасында жүйенің сенімділікке қалай жетуге болатынын;
- бағдарламалық жасақтаманың дамуындағы бағдарламалық қамтамасыз етудің сенімді үдерістерін;
- бағдарламалық артықтық пен түрлілікті жасау үшін әртүрлі архитектуралық стильдерді қалай қолданылатынын;
- сенімді жүйелердің жобалауында пайдалануға қажет бағдарламалаудың жақсы стильдерін түсінетін боласыз.

Мазмұны

Бағдарламалық жасақтама әдістерімен және бағдарламалау тілдерімен жақсартылған менеджментті пайдалану бағдарламалық сенімділікті арттырды. Сонда да, жүйенің жол ашықтыққа немесе теріс нәтижеге келтіретін жүйелік қателері болуы мүмкін. Кейде, осы қателер тек әлсіз ыңғайсыздыққа келтіреді. Жүйелік қамсыздандырушы жүйедегі қателерді түземей-ақ, онда жұмыс істей береді. Алайда, кейбір жүйелердегі қателер адамның ажалына немесе байыпты экономикалық жағдайдан немесе беделден айырылуына себеп болуы мүмкін. Осындай сенімділіктің жоғары деңгейлі маңызды жүйелері «қысылшаң жүйелер» болып есептеледі.

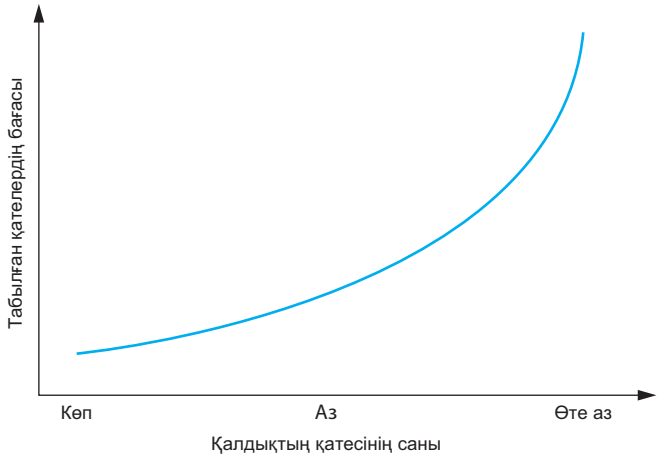
Қысылшаң жүйеге: үдерісті басқаратын жүйелер, қате жүйелерді жабатын жүйелерді, дәрігерлік жүйелерді, телекоммуникациялық коммутаторлардың ұшуымен басқарылатын жүйелерді жатқызамыз. Бағдарламалық қамсыздандырудың сенімділігін жақсарту үшін зерттеменің арнаулы аспаптармен әдістемелер қолдануы мүмкін. Осы аспаптармен әдістемелер жүйенің зерттемесінің бағасын көтереді, бірақ жүйелік қателерді жоғалтуын азайтады.

Қысылшаң және қысылшаң емес жүйелердің сенімділігін көтеру үшін пайдаланылатын сенімділіктің әзірлеу әдістері қолданылады. Бұл әдістер сенімді бағдарламалық жасақтаманың дамуында пайдаланатын үш қосымша тәсілдерді қолданады:

1. *Тоқтап қалуын болдырмау.* Жүйе орындалуында тоқтап қалу санын төмендету үшін бағдарламалық жасақтаманы әзірлеуді және енгізу үдерісін, бағдарламалық жасақтама тәсілдерін пайдалану қажет. Ол дизайнда және бағдарламауда қателеспеуге көмектеседі. Егер қателер аз болса, онда жүйе орындалғанда тоқтау мүмкіндігі аз болады.
2. *Тоқтап қалуын табу және жою.* Тексеру және растау үдерістері бағдарламада тоқтап қалмауын операциялық пайдалануға жазылғанға дейін немесе тауып алғанға дейін жасақталады. Тоқтап қалуын табу үшін жүйе сенімділігін енгізуін растау, қысылшаң жүйелердің кең тексеруін растауды қажет етеді. Бұл тақырып *15-тарауда* талқыланады.
3. *Тоқтамаушылық.* Жүйелік тоқтап қалуын болдырмас үшін, жүйенің құрылысының орындалуына кедергі жасайтын себебін және тоқтап қалуының күтпеген қылығын табу керек. Тоқтамаушылықта жай тәсілдемелерді орындалу кезінде негізделген кіріктірме тексермені барлық жүйелерде қолдануға болады. Бірақ тоқтамаушылықтың мамандандырылған әдістерін (тоқтамаушылық сәулет жүйесін пайдалану) әдетте, тек қана жоғары деңгейі жүйелік қатыстықпен сенімділік өте биік қажет болғандығына пайдалынады.

Қолданылатын әдістер тоқтауды болдырмауға, тоқтауды табуға және тоқтамаушылықты азайтуға келтіреді. Қалған тоқтауларды табу және бағдарламалық жасақтама жүйесінде жою бағасы күрт өседі, себебі бағдарламадағы тоқтаулар табылып, жойылады (*13.1-сурет*). Сондықтан бағдарламалық жасақтама сенімді

болғандықтан, өте аз тоқтауларды табу үшін көбірек уақыт пен күш салу керек. Кейбір жағдайларда, қысылшаң жүйесінде қосымша күш салу шығындарды ақтамайды.



13.1-сурет. Сәтсіздікке себепші факторларды жоюдың әзірлеу шығындарына қатынасы

Нәтижесінде, бағдарламалық жасақтамасын жасақтайтын компаниялар жойылмайтын тоқтаулардың болатынын құптайды. Тоқтаулардың деңгейі жүйенің түріне тәуелді. Дайын өнімдерде тоқтамаулар деңгейі өте жоғары келеді, сондықтан қысылшаң жүйелерде тоқтаулар біраз төмен болады.

Тоқтауларды қабылдау себебі, егер жүйе жұмыс істемей қалса, қатені тауып, жою салдарына төлеген арзан болады. Бірақ, *11-тарауда* айтылғандай, ақаулы бағдарламалық жасақтаманың өнімі тек экономикалық шешім емес. Жүйелік тоқтаулардың әлеуметтік және саяси жарамдылығына да назар аудару қажет.

Әуе көлігінде, дәрігерлік және қаржылық реттемелі саласында ұшақ жүйелер, дәрігерлік жүйелермен қатар қысылшаң жүйелерде қолданылады. Бұл облыстарда, ұлттық үкіметтерде қолданылылатын ережелерді анықтайды және компаниялар бұл ережелерді қолдану үшін, реттейтін мекемені тағайындайды. Іс жүзінде, бұл реттегіш бағдарламалық жасақтаманың сындық жүйелеріне сенуге болатынын және бұл жүйелердің сенімділік көрсететінін анық дәлелдеу қажет.

Қорыта келгенде, сындық жүйелерді әзірлеу үдерісі тек сенімді жүйенің жасауына тиісті; сонымен бірге ол жүйе сенімді болып табылады, бұл реттегіш дәлелдемеулер беру керек. Мұндай дәлелдемелерді өндіріс сындық жүйелердің әзірлемесіне шығындардың биік үлесін тұтынады және осылай сындық жүйелердің қымбат болатынына себепші фактор болып табылады. Мен өндірістің қауіпсіздігінің және сенімділік жағдайларының сұрақтарын *15-тарауда* талқылаймын.

13.1. Артықтық және әртүрлілік

Артықтық және әртүрлілік – кез келген жүйенің сенімділігінің жоғарылауына қолданылатын стратегиялар. Егер жүйенің бөлігі тоқтаса, бұл жүйенің резервті қабілеттілігін жұмсау мүмкіндігін «артықтық» деп атайды. Әртүрлілік жүйенің әртүрлі артық компоненттердің бірдей тоқтамайтынын білдіреді.

Біз күнделікті өмірлерімізде сенімділікті үлкейту үшін артықтықты және әртүрлілікті пайдаланамыз. Мысал ретінде, көпшілік адамдар жарық шамдарын тезарада жөндеп алу үшін, өз үйлерінде артық жарық шамдарын ұстайды, бұл артықтық болады. Әдетте, біз үйді қауіпсіздендіру үшін бір немесе оданда көп құлыптар пайдаланамыз (артықтық) және пайдаланылатын құлыптар әртүрлі болады (әртүрлілік). Егер қаскүнем құлыптардың бірін бұзудың әдісін тапса, ол басқа құлыптарды бұзудың әдісін табуы мүмкін, содан кейін ғана ол кіре алады. Біз компьютерлерде резервтік көшірме жасауымыз керек, біздің мәліметтер артық көшірмелер жасай алады. Дискінің тоқтауынан құтылу үшін, мәліметті резервтік көшірмелерде, түрлі сыртқы құрылғыларда сақтауымыз керек.

Ariane 5 жарылысы

1996 жылы, «Еуропалық ғарыш зерттеу басқармасының» Ariane 5 зымыраны ұшу алаңында көтерілгеннен кейін 37 секундта жарылды. Ақаулықтар бағдарламалық жүйелердің сәтсіз орындалуынан болды. Онда сақтық көшірмелеу жүйесінің бар болғанымен, ол екіге бөлінбеді және сондықтан сақтық көшірме компьютері тура сол тәртіппен команданы қате орындады. Зымыран мен Жердің жасанды серігінің пайдалы жүктемесі жойылды.

http://www.SoftwareEngineering-9.com/Web/DependabilityEng/Ariane/

Сенімділік үшін жасалған бағдарламалық жасақтаманың жүйесі басқа қамтамасыз ететін артық құрамдас бөліктер сияқты артықтық жүйелік құрамдас бөліктерді қоса алады. Егер негізгі құрамдас тоқтаса, онда олар жүйеге қосылады. Егер бұл артық құрамдастар әртүрлі болса, яғни басқа құрамдастар сияқты емес, онда көшірілетін құрамдастарда ортақ қателер жүйелік тоқтауларға алып келмейді. Артықтық жүйеге қажетті емес тексерістің қосымша кодексін қосумен қамтамасыз етіледі. Бұл кодекс тоқтаудан бұрын түрлі қателерді табуы мүмкін. Бұл жүйеге жұмыс істеу кепілдігін беру үшін, қалпына келтіретін тетіктерін іске қосу керек. Әдетте, қолжетімділік ең маңызды талаптардың бірі болып көрінетін жүйелерде қосалқы серверлер пайдаланады. Егер бұл сервер істен шығып кетсе, онда қосалқы серверлер автоматты түрде қолданысқа енеді. Кейде, жүйеге шабуылдар әсерін тигізбейтініне кепілдік беру үшін, бұл серверлер әртүрлі болады және әртүрлі жүйелер арқылы басқарыла алады. Әртүрлі операциялық жүйелерді пайдалану бағдарламалық жасақтаманың әртүрлілігінің және артықтығының мысалы

болып көрінеді. Мен *13.3.4-бөлімде* бағдарламалық жасақтаманың әртүрлілігін толығырақ талқылаймын.

Операциялық тәуелді үдерістер

Бұл тарауда тәуелді зерттеу үдерістері туралы талқылау жүргізіледі. Бірақ жүйе тәуелділігіне салымшы ретіндегі маңыздылығы тең дәрежеде жүйенің операциялық үдерістері жүреді. Бұл операциялық үдерістерді жобалауда адами факторларын есепке алуыңыз қажет және олардың жүйені пайдалану барысындағы жіберіп алуы ықтимал қателеріне қатысты жауапкершілік туралы жадыңызда сақтағаныңыз жөн. Тәуелді үдерістер адами фактордан болатын қателерді болдырау мақсатында жобалануы тиіс, ал бағдарламалық жасақтама қателерді тауып, оларды түзетілуіне мүмкіндік беруі қажет.

http://www.SoftwareEngineering-9.com/Web/DependabilityEng/HumanFactors/

Әртүрлілікті және артықтықты сенімді үдерістерге жету үшін пайдалануға болады. Үдерістің әрекеті бағдарламалық жасақтаманың тексерісі сияқты, жалғыз үдерісті немесе әдісті кепілдендіреді. Бағдарламалық жасақтаманың әзірлеу үдерісінің уақытында адамдардың жасаған қателерін, бағдарламалық жасақтаманың қателерін және үдерістің тоқтау мүмкіндігін азайтады. Бұл кез келген техникаға басқа әдіс өткізілген қатені таба алар ма дейтін қосымша әдістер. Одан басқа, әртүрлі команда мүшелері үдерістің сол қызметін қолдана алады (мысалы, бағдарламаның тексерілуі). Адамдардың даралық, тәжірибе және біліміне байланысты міндеттері әртүрлі, бұл артықтықтың түрі жүйені түрлі пікірлермен қамтамасыз етеді.

Біз бағдарламалық жасақтаманың әртүрлі жетістіктігін *13.3.4-бөлімде* талқылаймыз. Әртүрлілік және артықтық жүйе күрделі және түсінуге қиындық туғызады. Код тексеру және жазу құрамдас бөліктің тоқтауын табу және баламалы құрамдас бөліктерге бақылауларды ауыстыру үшін жүйеге қосымша функционалды қосу керек. Сірә, бұл қосымша күрделілікті бағдарламашылар қатені білдіреді және жүйені тексеретін адам бұл қателерді табады.

Кейбір адамдар бағдарламалық артықтыққа және әртүрлілікке жоламайды. Олардың ойынша жақсы тәсіл – бағдарламалық қамтамасыз етуін растау және тексеріс үдерістер өте қатал рәсімдерді пайдалану (Parnas et al., 1990). Бағдарламалық жасақтамада артық құрамдас бөліктердің дамуы қажет, сондықтан жинақтарды тексеру және бекіту керек.

Коммерциялық, қауіпсіздік сыни жүйелерде екі тәсіл пайдаланады. Мысалы, 340 аэробустың ұшу аппараты және бағдарламалық жасақтаманың басқару жүйесі артық және әртүрлі болады (Storey, 1996). Нөмірі *777-Боингтегі* ұшу бағдарламалық жасақтаманың басқаруы аппараты артық қаржыларға негізделген, бірақ әр компьютер экстенсивті бекітілген бағдарламалық жасақтамамен басқарылады. Нөмірі *777-Боингтің* басқару жүйесінің жобалаушылары оңай басқаруды көздеген.

13.2. Сенімді үдерістер

Бағдарламалық жасақтаманың сенімді үдерістері – сенімді бағдарламалық жасақтаманы өндіру үшін жасалған бағдарламалық жасақтаманың үдерістері. Сенімді үдерісті пайдаланатын компания бұл үдеріс тиісті бейнесімен бұйрық бергенін және тіркелген дамудың тиісті әдістеріне сыни жүйелерді даму үшін пайдалануы мүмкін. Сенімді үдеріске капиталдың салынуы үшін, дәлелдеулі бағдарламалық жасақтаманың жақсы үдерісінде қате аз болатын бағдарламалық жасақтамаға әкеледі және олардың орындалуында тоқтамайтын болып көрінеді. *13.2-сурет* бағдарламалық жасақтаманың сенімді үдерістеріне кейбір төлсипаттарын көрсетеді.

Үдеріс сипаттамасы	Баяндама
Құжаттандырылған	Үдеріс барлық іс-әрекеттерді құжаттандыруды қолданады.
Стандартты	Бағдарламаны әзірлеу және құжаттандыру стандарттары әрқашан қолжетімді болуы керек.
Тексерулі	Үдеріс оған қатысы жоқ адамдарға да түсінікті болуы керек. Ол адамдар үдерістің стандарттарға сай келетінін сынап, үдерісті жақсартатын кеңестерді береді.
Әртүрлі	Үдеріс әртүрлі тексеріс және қабылдау іс-әрекеттерін артығымен қолдану керек.
Мықты	Жүйенің көптеген сәтсіздіктер мен пайдаланушылардың әрекеттерінен кейін өзінің бұрынғы қалпына қайта оралу мүмкіншілігі әзірлену керек.

13.2-сурет. Тәуелді үдерістердің сипаттамалары

Сенімді үдерісті пайдаланатын дәлелдеу, ең тиімді программалау тәжірибесі бағдарламалық жасақтаманың дамуына қолданылған реттегіштің сенімділігін қажет етеді. Жүйенің өңдеушілері реттегіште үдерістің үлгісін көрсетеді, сонымен бірге бұл үдерісті дәлелдейді. Реттегіш үдерістің барлық қатысушыларымен дәйекті түрде пайдаланатынын және дамудың әртүрлі жобаларында пайдалану мүмкіндігіне сенуі керек. Бұл үдеріс қайталанатын және нақты анықталған болуы керек:

1. Нақты белгілі үдеріс – белгілі үлгісі бар, бағдарламалық жасақтаманың өндірісінің үдерісіне жағдай жасау үшін пайдаланатын үдеріс. Үдерістің үлгіде қажетті қадамдар мен бұйрық бергенін көрсету үшін уақытында мәліметтерді жинауы керек.
2. Қайталалатын үдеріс – жеке түсіндіруді және пікірді қажет етпейтін үдеріс. Бұл, үдеріс әртүрлі жобаларда және әртүрлі команда мүшелерінде қайталануы мүмкін. Дамуының айналымы уақытында өңдеушілердің то-

бында маңызды өзгерістер жиі болады, бұл әсіресе ұзақ айналымы болатын сыни жүйелер үшін маңызды.

Сенімді үдерістер артықтықты және әртүрлілікті сенімділікке жету үшін пайдаланады. Олар бірдей мақсатпен әртүрлі әрекеттерді жиі пайдаланады. Мысалы, бағдарламадағы қатені табу үшін оны тексертеді. Тәсілдердегі қателерді табу үшін бір-біріне қарағанда, бір аспапты қолданып, толықтырады.

Сенімді үдерістерде пайдаланатын әрекет, анық жасалатын бағдарламалық жасақтама түріне бағынышты болады. Жалпы, бұл әрекеттер жүйеге табылу қателерді енгізудің болдырмауын ыңғайлатады, қателерді алып тастайды және үдеріс туралы ақпараттарды жинайды. Сенімді үдеріске қосыла алатын әрекеттердің мысалдары:

1. Талаптарды толық және біртіндеп тексеру үшін.
2. Талаптарының өзгерістерін басқаруы үшін кепілдік беру және өзгерістерінің әсер талаптардың барлық өңдеушілерді түсініктемелермен басқару.
3. Бағдарламалық жасақтаманың математикалық моделі құралған және анализ жасалған үстірт спецификациясы. Мен үстірт спецификацияның артықшылықтарын *12-тарауда* талқыладым. Ең маңызды артықшылық – жүйелік талаптарды өте толық талдау. Бұл талдау талаптардың шолуларындағы өткізген талаптардың мәселесін табады.
4. Бағдарламалық жасақтама жобалауының үлгілері график түрінде берілген және талаптарының арасында байланысқан, бұл үлгілер анық тіркелген жүйелікте пішінделген.
5. Жүйенің әртүрлі сипаттамаларын қарап шығу және әртүрлі адамдармен бағдарламаның әрлендіру және тексерулерін жасау. Тексеру әрлендірудің және бағдарламалаудағы ортақ қателерінің бақылау тізімдерімен жиі басқарылады.
6. Бағдарламаның бастапқы кодта атқарылған автоматталған тексерістерінде статикалық талдауы. Олар программалық қателіктерді немесе білместікті көрсететін аномалияларды іздейді. Мен статикалық талдауды *15-тарауда* талқылаймын.
7. Жүйелік тесттердің толық теруі жасалған тесттермен жоспарлау және басқару. Бұл тест жүйелік талаптардың сақтандыруына ілігетінін және тест үдерісін дұрыс қолданғанын көрсетеді, тест үдерісін мұқият басқаруымыз керек.

Үдерістің әрекеті жүйелік дамуларда және тестілеуде шоғырланатын сияқты дәл осылай, сапаның және өзгеріс басқару үдерістердің айқын басқаруы тиіс. Бір жағынан, сенімді үдерісте белгілі әрекеттер бір компаниядан басқа компанияға көшкенге дейін өзгертіліп кетсе де, тиімді сапада және өзгерістерді басқаруға қажетті.

Тиімді басқару үдерістер (*24-тарауды* көріңіз) өнімнің және үдеріс стандарттарының қатарын орнатады. Бұл стандарттар сақтағанын көрсету үшін үдеріс туралы ақпаратты қамтитын әрекеттерді қосады. Мысалы, бағдарламаның

тексерулерін орындау үшін белгіленген стандарт болады. Инспекция стандартын сақтайтынын көрсету үшін инспекция тобының жетекшісі үдерістің құжаттамасына жауапты болады.

Ұзақ мерзімге төзу қауіпсіздігінің айналымы

Халықаралық электротехникалық комиссиясы қорғаныс жүйелерінің инжинирингіне арналған (IEC 61508) үдеріс стандартын жасап шығарды. Бұл стандарт қауіпсіздік инжинирингі мен жүйе инжинирингі арасындағы айырмашылықтарды айқындайтын ұзақ мерзімге төзу қауіпсіздігінің айналымы негізінде жасалды. IEC 61508 ұзақ мерзімге төзу қауіпсіздігінің айналымының бірінші деңгейі жүйенің ауқымын анықтайды, жүйенің болуы ықтимал қауіпті айқындайды және олардан болатын тәуекелдер дәрежесін бағалайды. Бұл амалдар қауіпсіздік талаптарына арналған спецификацияға сай жүргізіледі және осы қауіпсіздік талаптарының түрлі қосалқы жүйелерге орналастырылуы орындалады. Қауіпсіздікке маңыздылығы жоғары жүйенің дамуына қолданылатын маңыздылығы жоғары жүйелер инжинирингіне арналған арнаулы техникаларға мүмкіндік беретін қауіпсіздікке маңызы жоғары функционалдылық ауқымының кеңеюін шектеу басты мақсат болып табылады.

http://www.SoftwareEngineering-9.com/Web/SafetyLifeCycle/

25-тарауда талқыланған өзгерістерді басқару жүйесінің өзгерістері мен қабылданған өзгерістердің іс жүзінде ұқсатқандығына кепілдік беріп, бағдарламалық жасақтаманың жоспарланған шығарылымдары жоспарланған өзгерістер қосқанын растайды. Жүйенің құрылымға қосылған теріс құрамдас бөліктері, бағдарламалық жасақтамасының бір мәселесі болды. Атқарылатын жүйе әзірлеудің үдерісі тексерілмеген құрамдас бөліктерден тұратын жағдайға уақытында әкеледі. Бұл болмаған кепілдік беру үшін кескінін басқаруын рәсім өзгерістерді басқару үдерістің бір бөлігі сияқты анықталған болуы керек.

3-тарауда икемді тәсілдер туралы айтылған, сенімді үдерістер үшін (Boehm, 2002) кең таралған. Икемді тәсілдер бағдарламалық жасақтаманың дамуында, жасалған құжаттамаларда шоғырланбаған. Олар өзгеріс пен сапаны басқаруға жиі әжептәуір бейресми тәсілді қолданады. Реттегіштерге және басқа, сыртқы жүйелік мүдделі адамдарға түсінікті құжаттаманы жасайтын жоспарларда негізделген сенімді жүйенің дамуына арналған тәсілдерінің артықшылығы бар. Алайда, икемді тәсілдерді артықшылық сыни жүйелерге бірдей қолданады. Осы икемді әдістердің қолдануының жетістіктері туралы анықтамалар бар (Lindvall) және сыни жүйелердің жобалауына жарасатын икемді әдістердің нұсқаулары дамиды.

13.3. Жүйенің сенімді сәулеті

Мен айтқандай, жүйелердің сенімді дамуы сенімді үдерісте негізделуі керек. Сонда да, сізге сенімді жүйелерді жасау үшін сенімді үдеріс керек болуы мүмкін, ол

сенімділікке кепілдік беру үшін жеткіліксіз. Одан басқа, сізге сенімділік үшін жүйелік сәулетті жасау керек, әсіресе, тұрақты қызмет қажетті болғанда. Бұл сәулеттің жасалуы артық құрамдас бөліктермен тетіктерден тұру керек болғанын білдіреді, бұл құрамдастар бақылауды бір құрамдас бөліктен басқаға ауыстырып қосуға мүмкіндік береді.

Барлық жол бойы жұмыс істейтін ұшақтардағы жүйелер, телекоммуникациялық жүйелер және басқаратын сыни жүйелер тұрақты қызмет сәулеттерде керек болуы мүмкін. Pullum (2001) ұсынған сәулеттің әртүрлі түрлерін суреттейді және Torres-Pomales бағдарламалық жасақтаманың тұрақты қызмет әдістерін қарастырады (2000).

Сенімді архитектурасының ең оңай өткізуі көшірілетін серверлерде болады, бұл серверлер бір есепті шығарады. Өңдеуге сауал әр сауалды нақты серверге нұсқайтын басқарудың серверлік құрамдас бөлігіне бағытталады. Бұл құрамдас сервердің жауаптарын да зерттеп отырады. Жауаптың жоқтығымен әдетте, мәлім болатын сервердің істен шығуы жағдайында ақауы бар сервер жүйеден өшіріледі. Өңделмеген сауалдар өңдеу үшін басқа серверлерге қайтадан жіберіледі.

Бұл көшірілетін серверлік тәсіл, жұмыс транзакциялардың көшірмесі, жеңіл сақталуға болатын транзакцияларды өңдеу жүйелерінде кең пайдаланады. Транзакцияларды өңдеудің жүйесі мәліметтерді бір рет қана жаңартатын жүйеден жасалған. Егер қосалқы сервер әдеттегідей төмен есептер үшін пайдаланылса, бұл аппаратты құралды қолдану тиімді әдіс болуы мүмкін. Негізгі серверде мәселелер болғанда, оның өңдеу жұмысы ең биік басымдылықты берген резервтегі көшіру серверге кетеді.

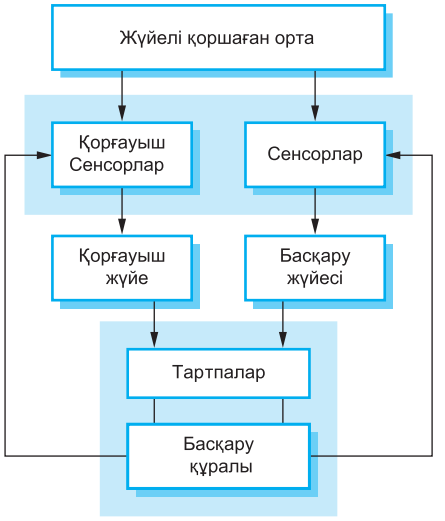
Көшірілетін серверлер әртүрлілікті емес, артықтықты қамтамасыз етеді. Аппаратты қаржылар әдетте, бірдей және олар бағдарламалық жасақтаманың сол нобайымен басқарылады. Сондықтан олар бағдарламалық жасақтаманың іркілістерімен және бір машинада жасалған аппараттармен жұптаса алады. Олар бағдарламалық жасақтаманың нобайы бәрі бір уақытқа алған, бағдарламалық жасақтаманы әзірлеуінің мәселелерімен жұптаса алмайды. Бағдарламалық жасақтама, жүйе ақаулардың өңдеуі үшін түрлі тәсілдерді қосуы керек және ол үдеріс *13.1-бөлімде* толығырақ талқыланады.

Әртүрлілік және бағдарламалық жасақтаманың артықтығы көпшілігінде әртүрлі сәулеттік стильдерден құрала алады. Мен бұл бөлімнің соңында осыған сәйкес кейбір мысалдарды сипаттаймын.

13.3.1. Қорғаныс жүйе

Қорғаныс жүйе – басқа жүйелермен байланған мамандандырылған жүйе. Әдеттегідей, бұл – химия өндірісінің үдерісін немесе пойыздағы жүйелерді басқаратын жүйелер. Пойыздағы жүйе қорғаныстық, қызыл сигналға өтетін пойыз жүйенің мысалы болуы мүмкін. Егер бұл осылай болатын болса, онда пойыздың тежеуіші автоматты түрде қолданылады және пойыз қозғалысы баяулайды. Егер

көрсеткіштері басқару жүйесі алмаған мәселені тапса, қорғаныстық жүйелер өз ортаны тәуелсіз бақылайды және үдерістің қорғаныстық жүйесін активтендіреді.



13.3-сурет. Жүйені қорғау сәулеті

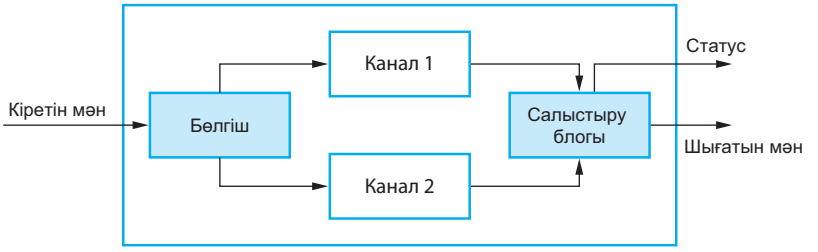
13.3-суретте қорғаныс жүйесінің аралығында өзара байланыс суретпен көркемделген. Қорғаныс жүйесін бақылайды. Егер атқарғыш механизмдерге әмір берілсе, жүйе жабылсын немесе қорғаныстың басқа тетіктері тартылсын. Байқасаңыз, көрсеткіштерінің екі теруі бар. Бір теру бақылаудың қалыпты жүйесі үшін пайдаланады, ал қорғаныстың жүйесі үшін басқа теру бар. Көрсеткіштің істен шыққан жағдайында бұл қорғаныстық жүйе операцияны жалғастыруға рұқсат ететін көшірме.

Жүйеде де артық атқарғыш механизмдер бола алады. Қорғаныстық жүйе әлеуеті қауіпті күйден (жүйенің өшіруі) қауіпсіз күйге жүйенің ауыстыруы үшін талап етілген ғана сыни функционалдар қосылады. Негізгі жүйеге кішірек қателік жасамайтын сәулет, мысалы, бұл ғана қажетті функционалдықта болатын резервтік жүйе. Мысалы, американдық ғарыш кемесінен бағдарламалы басқару резервтік жүйе, сізді ‘үйге әкелу’ функциясымен әуә кемесін жерге қондыруға мүмкіндік беретін резервтегі жүйе болады.

Мұндай сәулеттің артықшылығы – қорғаныстық жүйенің бағдарламалық жасақтамасы оңай әзірленуі мүмкін. Қорғаныстық жүйедегі жалғыз функция операцияның бақылауын қадағалайды және төтенше жағдайдың жағдайында қауіпсіз күй әкелетін жүйе ретінде кепілдік береді. Сондықтан қателер табылады. Бағдарламалық жасақтаманың спецификациясын дұрыс тексеруге болады. Мысалы, талап бойымен істен шығу ықтималдығы аз (0.001) ие болғанда қорғаныстық жүйесіне сенімді кепілдік береді. Қорғаныстық жүйедегі сәтсіздіктің ықтималдығы 1/1,000-ге тең болатыны жүйедегі қателердің өте сирек болғанын білдіреді.

13.3.2. Өздігінен тексерілетін сәулеттер

Егер мәселе табылса, өздігінен тексерілетін сәулет – жүйенің өз қызметін бақылау үшін жүйелік сәулетке және шешім қабылдауға арналған. Бұл жеке арналарға есептеулерді жүргізуге жетеді және бұл есептеулердің нәтижелерін теңестіреді. Егер нәтиже бірдей болса және уақыт жететін болса, онда жүйе дұрыс жұмыс істейтіндігіне қорытынды жасалады. Егер нәтиже әртүрлі болса, онда істен шығуы мүмкін. Бұл жүйеде болған жағдайда жүйедегі маңызды үдерістердің істен шығуы, ілгеріленеді. Бұл *13.4-суретте* көрсетілген.



13.4-сурет. Өзін-өзі бақылау сәулеті

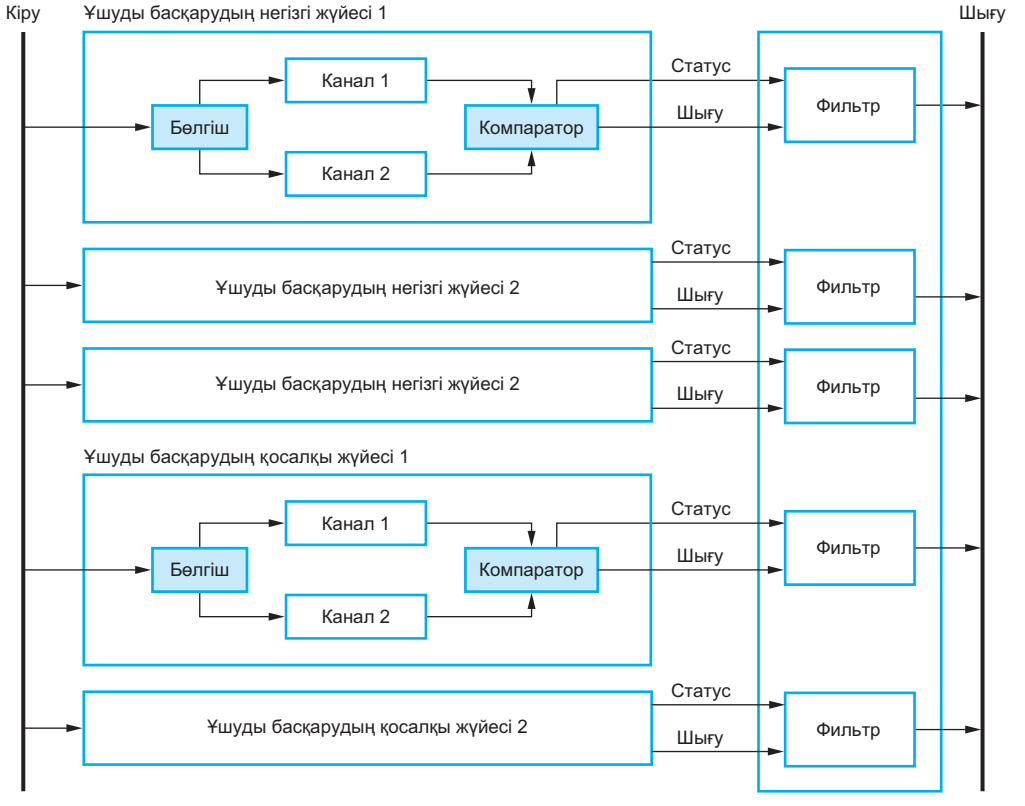
Қателерді табуға тиімді болатын және аппаратты бағдарламалық жасақтама жүйесінің өзін-өзі бақылауға алуын жасауы керек, ол үшін:

1. Әр арна қажетті есептеулерді орындау үшін процессордың әртүрлі пайдаланатынын мүмкіндік білдіретін немесе әртүрлі өндіруге алынған жүйе құрайтын микросызықтарды жасайды. Бұл есептеуге ықпал ететін процессорлардың ортақ қателерін әрлендіруінің ықтималдылығын кішірейтеді.
2. Әр арнада пайдаланылатын бағдарламалық жасақтама құрайды. Басқа жағдайда, программалық қателік те әр арнада бір мезгілде пайда бола алады. Мен *13.3.4-бөлімде* бағдарламалық жасақтаманың түрлі қиындықтарын талқылаймын.

Бұл сәулет өзімен-өзі пайдаланатын бағдарлама үшін маңызды, бірақ қолжетімділік маңызды емес, болған істің есептелуінің дұрыс болуы маңызды. Егер әр арнаның жауаптары ерекшеленсе, жүйе қарапайым өшіріледі. Көп дәрігерлік және сенімді емдік-диагностикалық жүйелері маңызды, өйткені теріс жүйелік жауап емделушінің теріс еміне әкелуі мүмкін. Алайда, жүйе қарапайым жағдайда қатені өшіреді, бірақ емделушіге жүйе қолайсыз.

Биік қолжетімділік талап етілген жағдайда, сіз тексерістің бірнеше жүйесін бірге пайдалануыңыз керек. Сіз қатені білінетін, ауыстырып қосатын бірлікте сақтайсыз және екі арна біртіндеп жауапты өндірген жүйенің бірі тергеуді таңдайды. Мұндай тәсіл тексерістің бес компьютерлерін пайдаланатын ұшақтың

340 топтамаларында, аэробус үшін ұшу-басқару жүйесінде пайдаланады. *13.5-сурет* – ұйым суретпен көркемдейтін ықшамдалған диаграмма.



13.5-сурет. Ұшақтың ұшуын бақылайтын жүйенің сәулеті

Ұшуды басқаратын компьютерлердің әрқайсысы аэробустың ұшуды басқару жүйесін қатарлас есептейді. Егер мәртебе қатеге көрсетсе, онда өнім аппараттық сүзгіге қатысады, компьютер сол өнімді өшіреді. Сол кезде, өнім баламалы жүйеден алынады.

Аэробус жүйесінің жобалаушылары түрлі әдіс-айлаларды қолданады. Мысалы:

1. Ұшуды басқаратын негізгі компьютерлер ұшуды басқаратын екінші жүйелерінен әртүрлі процессорды пайдаланады.
2. Негізінде әр арнада пайдаланатын микросызбалар теріледі және екінші жүйелер әртүрлі жасаушымен беріледі.
3. Ұшуды екінші басқару жүйелерінде бағдарламалық жасақтама бұл сыни функционалдығымен қамтамасыз етіледі.
4. Әр арна үшін бағдарламалық жасақтама екінші жүйеде де дамыған, әртүрлі бағдарлама әзірлеу тілдері қолданылады.

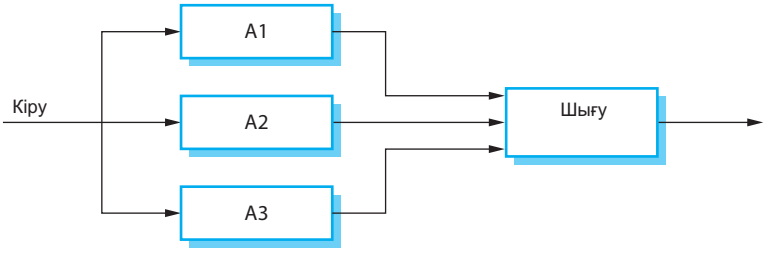
5. Әртүрлі бағдарлама жасау тілдерінде екінші және негізгі жүйелер пайдаланады.

13.3.3. N-нобайлы бағдарламалау

Өздігінен тексерілетін сәулет – көп нобайлы программалау артықты және әртүрлі бағдарламалық жасақтама үшін пайдаланатын жүйелер. Көп нобайлы бағдарламалаудың ұғымында (TMR) үш еселі модулдік сақтау ұғымының құрылыс жүйесінің (*13.6-сурет*) аппаратты құралдардың істен шығуынан шыдайтынын көрсететін көп пайдаланатын аппаратты жүйелер алынды.

TMR аппаратты қаржылар жүйеде (немесе кейде артық) үш рет көшіріп алынады. Әр блоктың істен шығу белгісі – дауыс. Егер екі немесе артық болса, бұл жүйе өзі басынан бастап бәрін салыстырады және онда мағына бұны шығарады. Егер бұл қондырғылардың бірі тоқтаса және сол нәтижені иелемеген болса, басқа бөлімше нәтижені өндірмейді. Менеджер ақаулы бірлік автоматты түрде қалпына келтіруге қателескен талаптану үшін қызмет көрсетуден бірлікті суырып алуы керек, бірақ бұл мүмкін емес, жүйе автоматты түрде қайтадан қалыптасады. Жүйе екі жұмыс бірліктерімен жұмыс істеуді сол кезде жалғастырады.

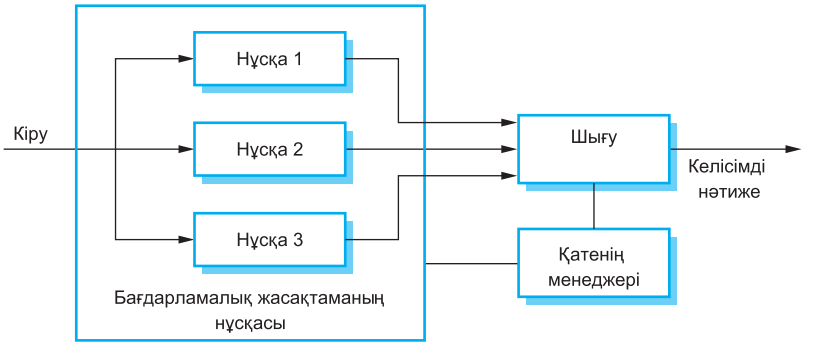
Әрине, құрамдас бөліктерде әрлендірудің ортақ қатесі болуы мүмкін және бәріне жауап береді. Аппаратты құралдардың бірліктерін пайдалану, ортақ спецификацияға барып тұр, бірақ жасалғандары және әртүрлі жасаушылармен салынған, мұндай ортақ режимді сәтсіздіктің мүмкіндігін кішірейтеді. Әрлендіру дәл сол жасайтын әртүрлі командалардағы ықтималдықтың өндірісте қатесі аз.



13.6-сурет. Енгізілген ақпаратты үшке бөліп әзірлеу

Ұқсас тәсіл бағдарламалық жасақтама N жүйесінің түрлі нобайларын (Avizienis, 1985; Avizienis, 1995) параллель орындалатын қателік жасамайтын бағдарламалық жасақтаманы пайдалану. *13.7-суреттме*, теміржол сигналдық жүйелерін, ұшақ жүйелерін пайдалануға бағдарламалық жасақтаманың тоқтамаушылық тәсілі және қорғаныстың реактор жүйелері суретпен көркемделген.

Ортақ спецификацияны қолданып, бағдарламалық жасақтаманың жүйесіне ұқсатуға болады. Бұл нобайлар жеке компьютерлерде атқарылған. Олардың өнімі дауыс беру жүйесін салыстыра пайдаланған. Үш жүйенің нобайы қалай дегенмен қол жеткізу үшін істің екі нобайы жалғыз сәтсіздіктің жағдайында болған.



13.7-сурет. N-нұсқалы бағдарламалау

N нұсқа бағдарламалау қолжетімділік биік деңгей талап еткен жүйелердегі сәулеті тексеріске қарағанда қымбат болуы мүмкін. Бұл бағдарламалық жасақтаманы әзірлеудің шығындарына өте қымбат болып келеді. Бұл тәсіл қауіпсіздіктің сыни қателеріне қарсы шара қолданатын қорғаныстың жүйесін қамтамасыз ету үшін нәтижеде қайта орамсыз жүйелерді ғана пайдаланады.

13.3.4. Бағдарламалық жасақтаманың әртүрлілігі

Жоғарыда айтылған қателік жасамайтын сәулеттердің жетістігі үшін бағдарламалық жасақтаманың әртүрлілігі ойда болады. Бұл спецификацияның (немесе спецификацияның бір бөлігі, қорғаныс жүйе үшін) соның өзіндегі түрлі енгізулері тәуелсіз шартқа негізделеді. Оларда таралған қателерді белгілі бір уақыт ішінде оған қосуға болады. Бұл бағдарламалық жасақтама әзірлеудің үдерісі әртүрлі әмірлермен жазылды, таралған адасуларды мүмкіндігінше кішірейтіп, спецификацияның қатесін түсіндіреді.

Жүйені қамтамасыз ететін компания, арналған әртүрліліктің айқын саясатын жүйелік нобайлардың арасындағы айырмашылықты қосуы мүмкін. Мысалы:

1. Әрлендірудің талаптарын әртүрлі әдістерімен пайдалануы. Мысалы, әрлендіру, тағы басқа әмір бір әмірге мүмкін объектіге бағдарланған әрлендіру бағытталған функционалдығын жасайды.
2. Енгізу бағдарламалары әртүрлі тілде жазылуы керек. Мысалы, үш нобайлар жүйесінде бағдарламалық жасақтаманың нобайына жазу үшін, Ада, C ++ және Java пайдаланылады.
3. Әртүрлі құралдарды пайдалану талап етеді және жүйе үшін жобалау орталары қолданады.
4. Әртүрлі алгоритмдер анық талап ететін кейбір енгізудің бір бөліктерінде пайдаланады.

Өндеушілердің әр тобы (Avizienis, 1995) жүйелік талаптарды спецификациядан алынатын (кейде V-spec аталатын) толық жүйелік спецификациямен жұмыс істеуі керек. Бұл тәптіштеу әжептәуір толық болуы керек, спецификацияда ешқандай да екіұштылықтар болмайтынына кепілдік беру керек. Дәл осылай жүйенің функционалдығы, толық спецификацияны анықтай салыстыру үшін жүйелік өнім жасау керек.

Жақсы болғанда, жүйенің түрлі нобайы ешқандай да тәуелділіктер алынбауы керек және абсолютті әртүрлі тәсілдермен сәтсіздікке ұшырауға тиісті әдістер қолдану керек. Егер бұл дұрыс болса, түрлі жүйесін әр міндеттемелері арта түскен арна толық сенімділікке алынады. Егер әр арнада талап бойымен сәтсіздіктің ықтималдығы бар болса, онда үш (барлық тәуелсіз арналармен) арналармен толық POFOD жүйе миллион болып көрінеді.

Іс жүзінде, алайда, арнаның толық тәуелсіздігінің жетістігі мүмкін емес. Тәуелсіз ұжымдар дизайнер қатені дәл сол жиі жасағанын эксперименталь түрде көрсеткен (Brilliant, et., 1990; Knight and Leveson, 1986; Leveson, 1995). Бұған бірнеше себеп бар, олар:

1. Әртүрлі командалар мүшесі жиі және білім алатын тәсілді қолданған. Бұл олар дәл осыны қиын түсінуге болжағышқа алғанын білдіреді және облыс арналған сарапшылармен хабарламада ортақ қиыншылықтарды сынайды. Сірә, олар қатені дәл сол тәуелсіз жасайды және алгоритм дәл соны жобалайды, бұл сол мәселені шешу үшін қолданады.
2. Егер талап теріс болса, онда бұл қателер жүйенің әр енгізуінде қайтарады немесе олар жүйенің ортасы туралы қателерде негізделеді.
3. V-spec сыни жүйесі − әмірлерге толық мазмұндама өзін тиісті хабардың жүйелеген сияқтыға ілігетін жүйеге талап негізделген толық құжаттар. Бағдарламалық жасақтаманың өндеушілеріне қателерін түсіндіру үшін облыс болу керек. Ортақ сипаттамалы қателерді азайтудың бір тәсілі – жүйе тәуелсіздігінің толық сипаттамасын әзірлеу және әртүрлі тілдердің сипаттамасын анықтау. Өндеушілердің бір тобы үстірт спецификация, басқа мемлекеттік жүйелік үлгіден жұмыс істей алатынын және үшінші табиғи тілдің спецификациясын ұсынады. Бұл спецификацияның түсіндіруінің кейбір қателері құтылуға көмектеседі, бірақ спецификацияның қателері мәселені жәмпейлемейді. Бұл да қателерді талаптарды аудармада мүмкіндік беріп, жүйесіз техникалық талаптарға енгізеді.

Эксперименттерді талдауда, (1997) Hatton, үш арналарды жүйе қорытындыға қарағанда, тоғыз есе сенімдірек келді. Сірә, ол алына алған сенімділікті жақсарту қорытындыға, жалғыз ғана версияның ресурстарын үлкенірек санды арнауға тура келді, бұл сәйкес келе алмады және қорыта келгенде нұсқалардың жалғыз тәсілдеріне қарағанда, тәсілдің нұсқасы сенімді жүйелерден астам нәтижелерге әкелді.

Бірде, айқын емес, алайда, дамудың қосымша шығындарын жүйе көп нұсқалық сенімділікті жақсартуынан тұрады. Қосымша құндар көп жүйелер

үшін ықтимал болмауы мүмкін, себебі жақсы жобалы нұсқаларды жалғыз жүйеде әжептәуір жақсы болуы мүмкін. Бұл қауіпсіз жерде ғана болады және сәтсіздіктің шығындары өте биік сыни жүйелерді тапсырады, онда көп нұсқалық бағдарламалық жасақтамаға мүмкіндік береді. Мысалы, тіпті жүйенің ғарыш кемесіне жататын мұндай жағдайларында, шектелген функционалдықты және әзірше қалпына келтірілген негізгі жүйесі резервтік көшірмені беруі мүмкін.

13.4. Тәуелді бағдарламалау

Жалпы, мен бұл кітапта бағдарламалауды талқыламадым. Өйткені бағдарламалауды белгілі бір бағдарлама жасау тілінің егжей-тегжейін білмей тұрып талқылау мүмкін емес. Қазіргі таңда бағдарламалау тілдерінің соншалықты көп болғандығынан, мен де бұл кітаптағы мысалдарымда тек қана бір тілмен шектелмедім. Алайда, тәуелді бағдарламалауды қарастырғанда, барлық тәсілдерге ортақ жақсы бағдарламалау тәжірибелерінің қатары бар және де олар қойылған жүйелердегі қате ықтималдығын кішірейтуге көмектеседі.

Тәуелді бағдарламалаудың нұсқаулары

1. Бағдарламадағы ақпараттың қолжетімділігін шектеу керек
2. Барлық енгізілетін ақпараттардың дұрыстығын тексерген жөн
3. Бағдарламаны барлық ерекшеліктерге дайын болуын әзірлеңіз
4. Сәтсіздікке ықтималдылығы бар құрылымдарды азайтыңыз
5. Жаңадан бастау мүмкіншілігін әзірлеңіз
6. Деректер ауқымының шектерін тексеріңіз
7. Сырттай компоненттерді шақырғанда жүйенің күту мүмкіншілігін әзірлеңіз
8. Дүниедегі мөлшерлерді анықтайтын барлық тұрақты шамаларды атап шығыңыз

13.8-сурет. Тәуелді бағдарламалаудың жақсы тәжірибелері

Тәжірибелермен қатар жақсы кепілдемелердің тізімі *13.8-суретте* көрсетіледі. Олар бағдарламалау қандай тілде болса да қолданылады және жүйелерді дамыту үшін пайдаланатын болады. Бір жағынан, олар пайдаланатын тәсілге, белгілі тілдерге бағынышты және жүйенің дамуы үшін пайдаланатын ескертулерге тәуелді болады.

1-нұсқау: Бағдарламадағы ақпараттардың жетімділігін бақылаңыз.

Ұйымдармен және әскери адамдармен қабылданған қауіпсіздіктің қағидаты. Қағидат ‘білуге тиісті’ болып табылады. Әр адамға өзіне тиісті ақпарат қана қолжетімді болады. Оның жұмысына қажетті емес ақпарат ол үшін жабық болады.

Бағдарламалау барысында, пайдаланатын айнымалылардың қолжетімділігін бақылап отырғаныңыз жөн. Бағдарлама бөлшектері өзіне қажетті деректерге ғана

қол жеткізе алуын әзірлеңіз. Басқа бағдарламалардың деректері олардан жабық болуы керек. Егер сіз белгілі бір деректерді жасырсаңыз, оларды сырттай басқа бөлшектермен зақымдау ықтималдылығы болмау керек. Интерфейстің тұрақты жағдайында, деректер көрінісінің өзгеруі жүйенің басқа бөлшектеріне әсер етпеуі керек.

Сіз осы нәтижелерге бағдарламаңыздағы деректер құрылымдарын абстрактілі болып әзірленуін талап етіп, жете аласыз. Абстрактілі деректердің анықталуы жасырынды болады. Құрылымның белгілері сырттай жасырынды болады да, операция арқылы мәліметтердің барлығына қол жеткізу мүмкіншілігі жойылады. Мысалы, сұраныстардың кезегін суреттейтін деректерді абстрактілі деректерге жатқызуға болады. Операциялар жүйеге қосатын және жүйеден жоятын мүмкіншілітерін қамтамасыз етеді. Сіз кезекті жиындар арқылы әзірлеп отырып, аяқ астынан оны байланысқан тізбек арқылы әзірлеуге шешім қабылдауыңыз мүмкін. Бұл өзгеріс жалпы кодқа ешқандай әсерін тигізбейді.

Сонымен қатар абстрактілі деректерді берілген мөлшердің дұрыс аралықта екенін тексеру үшін қолдануға болады. Мысал үшін, химиялық үдерістің температурасын баяндау керек дерлік. Рұқсат етілген температура аралығы 20-200 градустың аралығы болсын. Мөлшердің рұқсат етілген аралықта екендігін тексеретін шарт енгізіледі. Осылай температураның рұқсат етілген аралықта болғанына кепілдік беріледі.

Кейбір нысанға бағытталған бағдарламалау тілдерінде, абстрактілі деректер интерфейстерді анықтау арқылы жүзеге асырылады. Мысалы, сіз Кезек интерфейсін анықтай аласыз. Бұл деректердің кезекке тұруын, жойылуын және кезектің көлемі жайлы ақпараттарды демейді. Осы әдістерді орындайтын кластардың интерфейстері мен атрибуттары өздік болады.

2-нұсқау: Барлық енгізілетін ақпараттарды дұрыстыққа тексеріңіз

Бағдарламаның барлығы өз ортасынан кіріс деректерді алады және оларды өңдейді. Спецификация осы деректердің нақты пайдалануын тұспалдайды. Мысалы, банк шотының санын әрқашан болжауға болады, ол сегіз бүтін сандардан құралады. Егер кіріс дұрыс емес болған жағдайда, көптеген бағдарламалар қандай шара қабылдануы керектігін анықтамайды. Сөзсіз, пайдаланушылар қателер жібереді және анда-санда дұрыс емес деректерді енгізеді. Кейде, мен *14-тарауда* талқылағандай, кейде теріс енгізілген мәліметтер жүйеге жаман оймен шабуыл жасау мақсатында әдейі енгізіледі. Тіпті кіретін мәліметтер көрсеткіші құрылғыдан немесе басқа жүйелерден алынғанда, бұл жүйелер теріс жұмыс істеп және теріс мәндер бере алады. Сол себептен бағдарлама операциялық ортадан салыстырылып оқылған бойда, кіретін деректердің дұрыстығын ылғи тексерулеріңіз жөн. Тексерулер кіретін мәліметтерге ғана тәуелді болады, кейбір пайдалануға келетін тексерулердің мысалдары:

1. *Ауқымның тексерісі.* Сіз енгізілген ақпараттың берілген аралықтың шектерінде екенін тексере аласыз. Мысалы, ықтималдық болатын енгізу, 0,0-

ден 0,1-ге дейінгі аралықта болуы керек; судың температурасын көрсететін енгізу, 100 градуспен 0 градус арасында болуы керек.

2. *Өлшемнің тексерісі*. Мысалы, сіз енгізілетін сандардың таңбалар санын болжай аласыз. Кейбір жағдайда таңбалар бекітілген бола алмайды, бірақ жоғарғы шекті анықтауға болады. Мысалы, адамның есімінде 40 белгіден артық болуы күмән.
3. *Ұсынудың тексерісі.* Сіз енгізілген ақпараттың пішімін болжай аласыз. Мысалы, адам аты цифрлық белгілерден тұрмайды, ал электрондық поштаның мекенжайлары @ белгісі арқылы екі бөлікке бөлінеді, және т.б.
4. *Анықтықтың тексерісі.* Егер енгізілген ақпарат тізбекті болса, ол тізбектегі басқа мүшелерге ұқсас болады деп күтіледі. Мысалы, егер кіретін мәлімет тұрмыстық электр энергиясының санауышынан алынатын болса, онда электр энергиясының мөлшері жуық шамамен өткен жылғы дәл осындай уақыттың көрсеткіштеріне тең болады. Әрине, ауып кетулер болады, бірақ осы ауып кетулердің арқасында пайда болған мәселені анықтауға болады.

Кіретін мәліметтердің тексеру кезінде қателер шыққан жағдайдағы орындалатын әрекеттер:

Сіз мәселе туралы пайдаланушыға хабарлайсыз және мәліметтің қайтадан енгізілуін талап етесіз.

Егер мәліметтер сенсор құрылғысынан түссе, ең соңғы дұрыс енгізілген мағынаны қолдануға болады.

Кейбір жүйелерде керекті мөлшерді бұрынғы енгізілген мәліметтердің тарихы арқылы анықтауға болады.

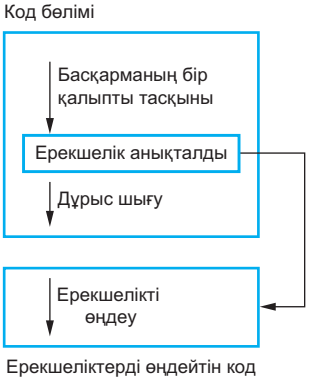
3-нұсқау: Барлық ерекшеліктерді өңдеумен қамтамасыз ету

Бағдарлама орындалу уақытында қате немесе күтпеген жайт оқиғалар сөзсіз болады. Мұндай жағдайлардың пайда болуы бағдарламаның қатесінің немесе болжамсыз сыртқы жағдайлардың нәтижесі болады. Қате немесе бағдарламаны орындау уақытындағы пайда болған күтпеген жағдай “ерекшелік” деп аталады. Ерекшеліктердің мысалы ретінде жүйе қуатының өшіп қалуын, жоқ деректерді талап етуін қарастыруға болады.

Ерекшеліктер бағдарламалық жасақтаманың және техникалық қамсыздандырудың шарттарынан пайда болады. Ерекшелік пайда болғанда, жүйе оларды бақылап және басқаруы керек. Бұл бағдарламаның өзінде жасалуы мүмкін немесе жүйелік ерекшеліктердің өңдеу тетігіне жіберілу керек. Әдеттегідей, жүйелік ерекшеліктерді өңдеу тетігі қате туралы хабарлайды және орындауды жабады.

Егер, мәлімдемені пайдалану керек болса, бағдарлама әзірлейтін C сияқты тілдерінде, ерекшеліктерді тауып, өңдеу кодексіне бақылауға беру керек. Бұл ерекшеліктер бағдарламаны қайта тексеру керек екендігін хабарлайды. Алайда бұл ерекшеліктерді өңдеу үдерісі өте шығынды болады.

Кейбір C++, Java, және Ada сияқты бағдарламалау тілдері, ерекшеліктерді өңдейтін құрылымдарды қосады, сондықтан сіз ерекшеліктерді тексеру үшін қосымша шартты операторларды қажет етпейсіз.



13.9-сурет. Сәтсіздіктің алдын алу

Ерекшеліктер пайда болған жағдайда жүйе дыбыс шығарады және олар ерекшеліктер басқару өңдеушісіне жіберіледі. Кодтың бұл бөлімі ерекшеліктердің атын бекітеді және әрбір ерекшелікті өңдеу үшін лайықты әсерлер әзірленеді (*13.9-сурет*). Ерекшеліктердің өңдеушісі басқарудың тұрақты ағынының сыртында болатынын есте сақтаңыз.

Ерекшеліктерді өңдеушілер әдетте бір немесе үш заттардан тұратын істерді істейді:

1. Жоғарғы деңгейдегі компонентке сигнал беріліп, ерекшелік болғанын айтады және сол ерекшеліктің түрі жөнінде мәлімет береді. Бір компонент арқылы басқа компонентті шақырған кезде, шамданған компонент шақырған компонентті табысты істелінгенін біледі. Егер жоқ болса, онда шамданған компонент мәселені жою үшін шаралар қабылдай алады.
2. Сол әлдеқандай талғаулы өңдеуді орындаңыз немесе бастапқы арналған өңдеуді орындаңыз. Сондықтан ерекшеліктердің өңдеушісі кейбір мәселеден кейін бұрынғы қалпына келтіретін шараларды қолданады.
3. Шығаруды жасайтын орындаудың қолдауын жүйе басқаруы тапсырады. Бұл бағдарламада кез келген қателердің пайда болуында жиі әсерін тигузеді.

Бағдарлама ерекшеліктерді өңдеу шеңберінде табуға рұқсат береді және кейбір кіретін қателерден кейін күтпеген жайтты сыртқы оқиғаларға жеткізеді. Бағдарлама тоқтамаушылықтың дәрежесін қамтамасыз ету үшін қателердің пайда болу тарихын жинау керек.

4-нұсқау: Құрылымдардың дұшар болған қателерінің қолдануын минимумға түйістіріңіз

Бағдарламалардағы қате және программалық бұзылуының салдары адами қателігінің әсері болып табылады. Бағдарламаушылар жиі қате жібереді, өйткені ол айнымалылардың арасындағы көп байланыстардың арақатынастығын жоғалтады. Олар жүйеге кенеттен жүріс-тұрысқа және өзгеріске алып келген бағдарламаларды жазады. Адамдар әрдайым қателіктер жасайды, бірақ 1960 жылдардың аяғында кейбір бағдарламалау тәсілдерінің басқаларға қарағанда, қате жасау ықтималдылығы жоғары екені дәлелденді.

Қателерді азайту мақсатында, кейбір бағдарламалау тілдері қателердің тарихын сақтап, олардың қайталану ықтималдылығын минимумға жеткізетін құрылымдарды әзірлейді. Осындай құрылымдардың үлгілері:

1. Шартсыз жүру мәлімдемесі Бұл 1968 жылы анықталған болатын (Дейкстра, 1968). Олар бағдарламалаудың қазіргі тілдерден ерекшеліктері еді.
2. Жүзитін үтірден кейінгі сандар. Бұл сандар табиғат заңы бойынша дәл емес. Бұл, әсіресе сандарды салыстыруда нақтылы мәселелерді тудырады. Мысалы, 2.99999999 сияқты сан кейде 3.00000000, ал кейде 3.00000001 сияқты болып көрінеді. Осы сандарды салыстыру күрделі болады.
3. Көрсеткіштер. Бағдарлама жасау тілдері, C және C++ (олар жад тұрған орынына көрсетеді) машина жадының облыстарына тікелей жататын мекен-жай ұстаған көрсеткіш аталған төменгі деңгейдің құрылымына сүйенеді. Егер олар теріс орнатылған болса және жадтың теріс облыстарына көрсеткіштерді пайдалануда қате қиратушы бола алады. Олар да тағы басқа құрылыспен байланған тексерісті ауырырақ жасайды
4. Жадты динамикалық түрде үлестіру. Бұл жағдайда жадтың бөлінуі программаның орындалу сәтінде басталады. Ең басты қауіп қатер, программалау жадының қолжетімсіз жерінде орындалған кезінде болады. Бұл ақауды байқау өте қиын, себебі жүйе бастапқыдай сәтті жұмыс істей береді, бірақ ақаулар кейін байқалады.
5. Үдерістердің параллельдігі. Үдерістердің параллель жұмыс істеу қиындығы, олардың мәліметтердің өзара алмасуынан болады. Бұл қиыншылықтарды жай бағдарлама кодына қарап анықтай алмайсыз; әдетте, сынақ кезінде жүйе уақытша параметрдің комбинациясын әрқашанда шығара бермейді. Үдерістердің параллельдігі болмаған жағдайда, олардың арасындағы өзара тәуелділікті азайтуға тура келеді.
6. Рекурсия. Рәсім немесе әдіс өзін-өзі немесе басқа процедурамен шақырыла алады және ол соңында бастапқы процедураны шақырады. Рекурсияның көмегімен логикасы қиын және жолы қысқа программалар құруға болады. Сондықтан программаның қателерін табу қиындыққа әкеледі. Рекурсияны қолданған кезінде коп қателікке әкеледі.
7. Үзілістер. Дәл қазір жасалып жатқан прогаммаға тәуелсіз, программаның нақты бір бөлігіне басқарудың ықтиярсыз ауысатын құрал-жабдық-

тарының бірі. Бірақ қателігінен қандай да бір маңызды операциялар тоқтап қалуы мүмкін.

8. Мұрагер. Объектіге-бағытталған программалау тілінде кодта нақты бөліктерді бірнеше рет қолдануға мүмкіндік береді және есептің декомпозициясын есептейді. Программалау коды бір объектіге тиісті, бірақ программаның әртүрлі тарауларында орналасады. Демек, бұл объект түсінігін қиындатады және қателер көп шығады.
9. Бірнеше аттарды біріктіру. Бұл әртүрлі аттар бір ғана программалау объектісін шақыру үшін пайдаланылған кезде болады. Программалау кодын оқыған кезде бірнеше атты шақыратын, өзгертілген объект жағдайын оңай өтіп кетуге болады.
10. Шексіз жиындар. С тілінде, массив – жадқа бару жолдарының тәсілі және сіз массивтің соңында тағайындаулар жасай аласыз. Жүйе, тағайындаулар массив элементтеріне тиісті немесе тиісті емес екендігін тексермейді.
11. Тексерусіз енгізілетін мәліметтер. Кейбір жүйелер мазмұнында тәуелсіз мәліметтерді енгізу процесін ұйымдастырады. Бұл жүйенің қорғаныштығындағы қатеге әкеп соғады, себебі жүйеге қабыл алмайтын мәліметтер енгізіледі.

Даму жүйелердің қауіпсіздігі үшін кейбір стандарттар сынауға қатысты құрылымдарды пайдалануға толық тыйым салады. Алайда, мұндай кесімділік әрқашан ақталмайды. Бұл құрылымдар мен әдістер пайдалы, бірақ оларды сақ пайдалану керек. Мүмкіндігінше, абстракт түріндегі мәліметтер немесе объектілер әлеуеті қауіпті әсерлерге тексерілу керек. Егер қате болса, олар 'брандмауэр' сияқты әрекет жасайды және зиянын тигізеді.

5-нұсқау: Қайта қосылуын қамтамасыз ету

Ақпараттық жүйенің көбісі қысқа транзакцияларды қолданады. Бұл жүйе қолданушының терген мәліметін тез өңдеуге көмек береді. Жүйе алдымен кезекте тұрған операцияларды жасағаннан кейін ғана, дерекқордағы өзгерістерді өңдейді. Өңдеу кезінде ақау пайда болса, дерекқор жаңармайды. Қазіргі электронды коммерциялық жүйенің көбісі осылай жұмыс істейді. Қолданушы мен Электронды коммерциялық жүйе әдетте бірнеше минут уақытқа созылады және өңдеу операциялары аз болады. Қысқа транзакциялар бірнеше секундқа ғана созылады. Алайда CAD және мәтін өңдейтін жүйелер ұзақ уақыт алады. Ұзын транзакциялық жүйеде операцияның басымен аяғының арасында созылған уақыт бірнеше сағатқа дейін созылу мүмкін. Егерде операцияның ішінде қателік болса, жұмысы қайта басталу керек болады.

Бұндай жағдайларда қайта қосылу жүйесімен қамтамасыз ету қажетсіз. Бұл жүйе арқылы өңдеу уақытында жаңадан жасаған немесе жинап алынған бүкіл деректерінізді сақтай аласыз. Қайта қосылу әдісі сіздің жүйеңізде, сақталған мәліметтерді қолданып, бітірген жұмысыңыздан ары қарай жалғастыруға көмектеседі. Мысалы:

1. Электронды коммерция жүйесінде сіз толтырған тармақтарды сақтап, келесі кірген сәтте сол терген мәліметтерді қайта қолдануыңыз болады.
2. Ұзын транзакцияларда және үлкен есептеу жүйелерде, сіз автоматты түрде бірнеше минут сайын мәліметтеріңізді сақтай аласыз, жүйелік ақау сәтінде сіз өз жұмысыңызды оңайлықпен қайта қосылу арқылы жалғастыра аласыз. Қолданушының ақаулықтарын есепке алуыңыз керек. Сондай жағдайда қолданушыға ақау басталған сәттен бастап жұмысын жалғастыруға мүмкіндік беруіңіз қажет.

Егер ерекше ақау немесе жұмысты жалғастыруға еш мүмкіндік болмаса, ерекше ақауды қайта өңдеуден ақауды қайта жөндеу жүйесі арқылы өткізу қажет. Ең соңғы тоқталған жерінен бастап жүйені қайта іске қосу дегенді білдіреді.

6-нұсқау: Массивтің шегін тексеріңіз

Барлық бағдарлама тілдерінде массивті анықтау рұқсат етілген. Мәліметтерді массивтің индекс саны арқылы қол жеткізуге болады. Бұл массивтер әдетте жадының іске асатын жеріне жақын орналасады. Әр массифтің өзіндік көлемі болады. Мысалы, сіз 10,000 адамның жасын анықтап оларды үлкен немесе кіші деп бөлуіңіз керек. Бұл жағдайда сізге көлемі 10,000 тең массиф құруыныз қажет.

Кей бағдарлама тілдері, JAVA тілі секілді, енгізілген мәліметтердің индексі құрылған массивтің көлемімен сәйкес болуын қадағалап отырады. Сол үшін массивіңіздің көлемі 0-ден 10,000-ға дейін болса, A[-5] немесе A[12345] элементтерін енгізе алмайсыз. Бағдарлама сізге ақау шығарады. Алайда C немесе C++ тілдері массивтін көлемін бақыламайды. Бұл тілдер элементті массивтің басына немесе артына қосады. Сол себепті A[12345] элементі өзінің индексіне сай мәлімет алады. Массивте бұндай элемент бар жоғына қарамастан.

Осы екі тілдің автоматты бақылау жүйесінің қолданбау себебі – бұндай типті процестер қосымша жад талап етуі мүмкін. Массив элементтерінің шақырылу индексі көп жағдайда дұрыс, сол себепті массивті бақылау жүйесінің болмауы орынды. Бұл процес қосымша уақыт алады. Алайда шектерді тексеру процесінің болмауы бүкіл бағдарламаның қауіпсіздігіне кері әсерін тигізу мүмкін. *14-тарауда* түсіндіргенімдей буфердің толып кетуіне алып келеді. Ал буфердің толып кетуі бағдарламаның жойылуына алып келу мүмкін. Егерде сіз шектерді автоматты түрде тексермейтін тілді таңдасаңыз, сіз шектерді ретке келтіретін қосымша код жазуыңыз тиіс. Бұл кодтарды 1-қосымшада түсіндіргенімдей абстрактты мәлімет түрімен жаза аласыз.

7-нұсқау: Сыртқы құрамдас бөліктер шақырғанда үзіліс қосыңыз

Таралған жүйелерде, жүйелер компоненттері әр түрлі компьютерлерде орындалып және шақырулар компонентке компоненттен желі арқылы істелінеді. Кейбір қызмет көрсетуді алу үшін, компонент А компонент Б-ні шақыра алады. Ал орындалудың

жалғасының алдында жауап беру үшін, А Б-ді тосады. Егер Б кейбір себептермен жауап бере алмай қалса, онда А өз ісін жалғастыра алмайды. А белгісіз ұзақ уақыт жауапты тосады.

Жүйеден жауап күткен адам, жүйеден тыныш жүйелік сәтсіздікті көреді. Оларда күту үдерісін жойып және жүйені қайта жүктеуден басқа таңдауы жоқ. Бұдан құтылу үшін, сіз сыртқы компоненттерді шақырар кезде, әрдайым үзілістерді қосуға тиістісіз. Үзіліс бұл – автоматты жорамал, ол компоненттің құрылыстан шыққандығын және жауапты өндіре алмайтындығын айтады. Шақырылатын компоненттен жауапты күту уақыт мерзімін өзіңіз таңдайсыз. Егер сіз жауапты берілген уақытта алмасаңыз, сіз оны қателік деп, шақырылған компоненттен бақылауды қайтарасыз.

8-нұсқау: Нақты мағынасы бар барлық тұрақты шамаларға ат қойыңыз

Барлық қарапайым емес бағдарламалар нақты тұрақты қатар мәндердің ұсынатын мәндер қатарына қосылады. Бұл мәндер бағдарламалар орындалу уақытында өзгермейді. Кейде, бұл абсолютті тұрақты ешқашан өзгермейді (мысалы, сәуленің жылдамдығы), бірақ олар ақырын ұзақ уақыттың ағымына өзгерген мәнмен тең келеді. Мысалы, табыс салығының есептеуі үшін бағдарлама бүгінгі салық мөлшер тұрақтысын қосады. Бұл өзгерістер жылдан жылға жалғасады, сондықтан бағдарлама жаңа тұрақты мәндермен жаңартылған болуы тиіс.

Тұрақтыны қолдануда, сіз оларға олардың құны бойынша емес, аты бойынша сұрауға тиістісіз. Беймаза сенімділік бойынша бұл екі артықшылықты беріп жатыр:

1. Сіз қатені жіберіп және бұрыс мән қолдануыңыз, кем дегенде ықтимал. Нөмірді енгізуде оңай қателесуіңіз және жүйе қателікті табуға шамасы жетпеуі мүмкін. Мысалы, салық мөлшерлемесі 34% құрайды. Транспозицияның жай ғана қателігі бұл жаңылысты 43% келтіре алады. Егер сіз есімді енгізуде қателессеңіз, (мысалы, стандарт-салық мөлшері), бұл әдетте жарияланбаған айнымалы ретінде компилятормен мәлім болып табылады.
2. Мәннің өзгерісінде, сіз бұл мән қолдану үшін барлық бағдарлама арқылы қарауға тиісті емес. Сізге тұрақты декларациядан сабақтас мәннің маңызды өзгеруін әзірлеу керек. Жаңа мән барлық қажетті жерде автоматты түрде қосылады.

НЕГІЗГІ ТҰЖЫРЫМДАР

- Бағдарламалардың функционалдық сенімділігі қателердің шығаруын және тоқтамаушылықтың құрал-жабдықтарын белгілі жолымен әзірлейді, әр

жүйенің қызмет ету жалғасы тіпті жүйеде қате келтіргеннен кейін қамтамасыз етіледі.

- Аппаратты қаржылар, программалық процестерде артықтық және әртүрліліктің қолдануы, жабдық және бағдарламалық қамтамасыз ету сенімді жүйелердің дамуы үшін маңызды мәнге ие болады.
- Егер жүйеге қате минимизациялауы керек болса, қайталанатын, айқын үдерісті пайдалану маңызды. Үдеріс талаптарды анықтамадан жүйелік енгізуге дейін барлық кезеңдерде тексеретін және бекітетін әрекетті қосуы керек.
- Сенімді жүйелік архитектура - тоқтамаушылық үшін жасалған жүйелік архитектура. Қорғаныс жүйесі өздігінен тексерілетін архитектура және көп нобайлы программалау сияқты тоқтамаушылыққа сүйенетін көптеген стильдер бар.
- Бағдарламалық жасақтаманың әртүрлілігіне жету қиын, өйткені бағдарламалық жасақтаманың әр нобайы шынымен тәуелсіз, әрі іс жүзінде кепілдік беруі мүмкін емес.
- Бағдарламаға тәуелді программалау артықтықшылықтың қосуын ойлау үшін кіре берістерді заңдылықты және бағдарламаның айнымалыларының мағынасын тексеру керек.
- Кейбір программалық құрылымдар және әдістер жүрген мәлімдемеде көрсеткіштер рекурсияны, мұрагер болуды және жүздеген үтір саны, ажырамас бейім қателері болып көрінеді. Сіз бұл құрылымдармен сенімді жүйелерді өндеп шығаруға жұмыс істеуіңіз керек.

ҚОСЫМША ӘДЕБИЕТТЕР

Software Fault Tolerance Techniques and Implementation. A comprehensive discussion of techniques to achieve software fault tolerance and fault-tolerant architectures. The book also covers general issues of software dependability. (L. L. Pullum, Artech House, 2001.)

'Software Reliability Engineering: A Roadmap'. This survey paper by a leading researcher in software reliability summarizes the state of the art in software reliability engineering as well as discussing future research challenges. (M. R. Lyu, *Proc. Future of Software Engineering*, IEEE Computer Society, 2007.) http://dx.doi.org/10.1109/FOSE.2007.24.

ЖАТТЫҒУЛАР

13.1. Бағдарламаның қатесіз орындалатынына көз жеткізу не үшін керек екенін талқылаңыз.

13.2. Сенімді үдерістерді пайдалану сенімді Бағдарламалық жасақтаманы жасауға әкелетінін түсіндіріңіз.

13.3. Қосылған сенімді үдерістерде бола алатын әрекеттердің екі мысалын беріңіз.

13.4. Бағдарламалық жасақтаманың тоқтамаушылығының қолдауына ыңғайлы болатын барлық архитектуралық стильдер үшін ортақ?

13.5. Бағдарламалық жасақтамаға негізделген басқару жүйесін жүзеге асыратынын болжаңыз. Жанында қателік жасамайтын сәулетке лайықты пайдаланған жағдайды ұсыныңыз және бұл тәсілді түсіндіріңіз.

13.6. Сіз 24/7 тұжырымы бойынша қолжетімді коммуникациялық ажыратқышты әзірлеу керексіз, бірақ қауіпсіздігі онша маңызды емес. Жауабыңыз үшін себеп келтіріңіз, жүйе ол үшін пайдалана алар ма дейтін сәулеттік стильді ұсыныңыз.

13.7. Қатерлі ісіктерді емдейтін құрылғының бағдарламасын N-нұсқалы тәсілмен жазу керек дерлік. Бұл тәсіл осы жағдайда қаншалықты тиімді екендігін талқылаңыз.

13.8. Бағдарламалық жасақтаманың әртүрлілігі жүйенің сәтсіздігіне қалай әкелетінін талқылаңыз.

13.9. Неліктен ерекшеліктердің алдын алу керектігін түсіндіріңіз.

ПАЙДАЛАНЫЛҒАН ӘДЕБИЕТТЕР

Avizienis, A. (1985). 'The N-Version Approach to Fault-Tolerant Software'. *IEEE Trans. on Software Eng.*,**SE-11** (12), 1491–501.

Avizienis, A. A. (1995). 'A Methodology of N-Version Programming'. In *Software Fault Tolerance*. Lyu, M. R. (ed.). Chichester: John Wiley & Sons. 23–46.

Boehm, B. (2002). 'Get Ready for Agile Methods, With Care'. *IEEE Computer,***35** (1), 64–9.

Brilliant, S. S., Knight, J. C. and Leveson, N. G. (1990). 'Analysis of Faults in an N-Version Software Experiment'. *IEEE Trans. On Software Engineering,***16** (2), 238–47.

Dijkstra, E. W. (1968). 'Goto statement considered harmful'. *Comm. ACM.,* **11** (3), 147–8.

Hatton, L. (1997). 'N-version design versus one good version'. *IEEE Software,***14** (6), 71–6.

Knight, J. C. and Leveson, N. G. (1986). 'An experimental evaluation of the assumption of independence in multi-version programming'. *IEEE Trans. on Software Engineering.*,**SE-12** (1), 96–109.

Leveson, N. G. (1995). Safeware: *System Safety and Computers. Reading,* Mass.: Addison-Wesley.

Lindvall, M., Muthig, D., Dagnino, A., Wallin, C., Stupperich, M., Kiefer, D., May, J. and Kahkonen, T. (2004). 'Agile Software Development in Large Organizations'. *IEEE Computer,***37** (12), 26–34.

Parnas, D. L., Van Schouwen, J. and Shu, P. K. (1990). 'Evaluation of Safety-Critical Software'. *Comm. ACM,***33** (6), 636–51.

Pullum, L. L. (2001). *Software Fault Tolerance Techniques and Implementation*. Norwood, Mass.: Artech House.

Storey, N. (1996). *Safety-Critical Computer Systems*. Harlow, UK: Addison-Wesley.

Torres-Pomales, W. (2000). 'Software Fault Tolerance: A Tutorial.' http://ntrs.nasa.gov/archive/nasa/casi./20000120144_2000175863.pdf.

14

Қауіпсіздікті қамтамасыз ету

Мақсаттары

Бұл тарау сіздерді қолданбалы жүйені әзірлеу үдерісінде шешу қажет сұрақтармен таныстыруға арналған. Осы тарауды оқығаннан кейін Сіз:

- қосымшаның қауіпсіздігі мен инфрақұрылымның қауіпсіздігі арасындағы айырмашылықты түсінесіз;
- өмір циклі тәуекелін бағалауды пайдалану әдістерін және жүйе құрылымына кері әсер ететін қауіпсіздік проблемаларын анықтауға пайдалану тәуекелін бағалау әдістерін білесіз;
- бағдарламалық қамтамасыз ету архитектурларымен танысасыз және қауіпсіз жүйелерді әзірлеуге қатысты нұсқаулар аласыз;
- жүйенің тіршілікке қабілеттік концептін, сонымен қатар, күрделі бағдарламалық жүйелер үшін тіршілікке қабілеттікті талдау неге маңызды екенін түсінесіз.

Мазмұны

1990 жылдары Интернетті кеңінен қолдану бағдарламалық қамтамасыз етуді әзірлеушілерге жаңа проблеманың – қауіпсіз жүйелерді жобалау және іске асыру туындауына алып келді. Интернетке барған сайын көптеген жүйелердің қосыла бастауына байланысты аталған жүйелер үшін қауіпті түрлі шабуылдар ойлап шығарылды. Функционалдық сенімді жүйелер құру проблемасы өзекті бола түсті. Жүйе әзірлеушілерге тек қана мол техникалық білімі бар бұзушылар тарапынан төнетін қауіппен ғана емес, сонымен қатар, әзірлеу үдерісінде кездейсоқ жол берілген қателіктер нәтижесінде туындаған проблемалармен де ұшырасуға тура келді.

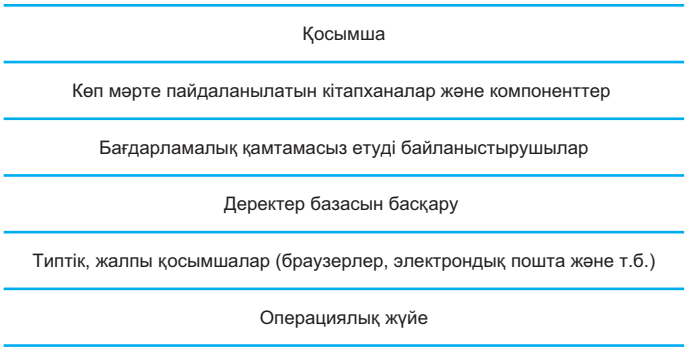
Қазіргі таңда сыртқы шабуылдарға төтеп беретін және мұндай шабуылдардан кейін қайта қалпына келуге қабілетті жүйелер әзірлеу маңызды болып табылады. Егер белгілі қауіпсіздік шаралары қабылданбаса, бұзушылар желі жүйесінің жұмысының міндетті түрде абыройын түсіреді. Олар жүйенің аппараттық құрал-жабдықтарын теріс пайдалануы, құпия деректерді ұрлауы немесе жүйе ұсынатын қызметтерді орындауды үзіп жіберуі мүмкін. Осылайша, жүйеде қауіпсіздікті қамтамасыз ету жүйе әзірлеудің негізгі аспектілерінің бірі болып табылады.

Қауіпсіздікті қамтамасыз ету – бұл мақсаты жүйеге немесе оның деректеріне зақым келтіру болып табылатын әдейі шабуылдарға төтеп беруге қабілетті жүйені әзірлеу және дамыту. Аталған сала көптеген жеке және заңды тұлғалар, сонымен қатар, желілік жүйелерді заңсыз мақсаттарға пайдалануға тырысатын қылмыскерлер үшін басымдыққа ие сала болып табылады. Бағдарламалық қамтамасыз етуді әзірлеушілерге жүйелерге төнетін қатерлер және мұндай қауіптердің алдын алу әдістері белгілі болуы тиіс.

Бұл тараудың мақсаты БҚ әзірлеушілерді қауіпсіздікті қамтамасыз ету әдістерімен таныстыру болып табылады, ал негізгі назар қосымшалардың қауіпсіздігіне әсер ететін жобалау аспектілеріне аударылады. Аталған тарау жалпы компьютерлік қауіпсіздікке арналмаған, сол себепті қолжетімділікті бақылау, авторлау механизмі, вирустар, трояндар және т.б. мәселелерді ашпайды. Аталған мәселелер компьютерлік қауіпсіздікке арналған еңбектерде кеңінен ашылады (Андерсон, 2008; Бишоп, 2005; Пфлегер және Пфлегер, 2007).

Бұл тарау осы оқулықта ашылатын қауіпсіздік мәселесін талқылауға қосымша болып табылады. Тараудың материалдарымен қатар Сізге оқып-білу қажет:

- 10.1-бөлім, мұнда қауіпсіздік пен функционалдық сенімділік неге тығыз байланысты екені түсіндіріледі;
- 10.4-бөлім, мұнда оқушы қауіпсіздік аспектісі терминологиясымен танысады;
- 12.1-бөлім, оқушы тәуекелдер негізінде жалпы ерекшелік түсінігімен танысады;
- 12.4-бөлім, қауіпсіздікке талаптарды анықтау мәселесі талқыланады;
- 15.3-бөлім, мұнда мен қауіпсіздікті тестілеуге бірқатар тәсілдерді талқылаймын.



14.1-сурет. Жүйенің қауіпсіздігі бұзылуы мүмкін қабаттары

Қауіпсіздік мәселелерін зерттегенде қолданбалы бағдарламалық қамтамасыз ету ретінде қарастыру қажет (басқару жүйесі, ақпараттық жүйе және т.б.), бұл сонымен қатар, аталған жүйе негізінде жатқан инфрақұрылымға да қатысты (*14.1-сурет*). Күрделі қосымшалардың инфрақұрылымдары төмендегідей болуы мүмкін:

- операциялық жүйе платформасы (мысалы, Linux немесе Windows);
- аталған жүйе шеңберінде жұмыс істейтін типтік қосымшалар (мысалы, Интернет-браузерлер және электрондық пошта клиенттері);
- деректер базасын басқару жүйесі;
- бөлінген есептеулерді және деректер базасына қолжетімділікті қолдайтын байланыстырушы бағдарламалық қамтамасыз ету;
- қолданбалы бағдарламалық қамтамасыз етуде пайдаланылатын көп мәрте пайдаланымдағы компонентер кітапханасы.

Сыртқы шабуылдардың көпшілігінің мақсаты жүйенің инфрақұрылымы болып табылады, себебі инфрақұрылымдар компоненттері жақсы таныс және қолжетімді (мысалға, Интернет-браузерлер. Бұзушылар аталған жүйелердің осал жерлерін тексеріп, өздері анықтаған кемшіліктер туралы бөлісуі мүмкін. Адамдардың көпшілігі бірдей бағдарламалық қамтамасыз етуді қолданатындықтан, бір бұзу әдісін көптеген пайдаланушыларға қатысты қолдануға болады. Инфрақұрылымда осал жерлердің, кемшіліктердің болуынан бұзушылар қолданбалы бағдарламалық қамтамасыз етуге және оның деректеріне рұқсатсыз енуі мүмкін.

Тәжірибеде қосымша қауіпсіздігі және инфрақұрылым қауіпсіздігі арасында белгілі айырмашылық бар:

1. Қосымшаның қауіпсіздігі – бұл бағдарламалық қамтамасыз етуді жобалаудың міндеті, яғни бағдарламалық қамтамасыз етуді әзірлеушілер жүйенің шабуылдарға қарсы тұру қабілетін қамтамасыз етуі тиіс дегенге негізделеді.
2. Инфрақұрылымның қауіпсіздігі – бұл басқару міндеті, яғни жүйе менеджерлері инфрақұрылымды ол шабуылдарға қарсы тұруға қабілетті бо-

латындай конфигурациялауы тиіс дегенге негізделеді. Жүйе менеджерлері инфрақұрылым қауіпсіздігі қасиеттерін неғұрлым тиімді пайдалануға қол жеткізуге болатындай етіп инфрақұрылымды конфигурациялауы қажет. Олар сонымен қатар, инфрақұрылымды пайдалану үдерісінде анықталатын кемшіліктерді де жоюлары тиіс.

Жүйе қауіпсіздігін басқару – бұл бір жолғы ғана тапсырма емес, шаралардың біртұтас қатары, мысалы: пайдаланушыларды және қолжетімділік құқығын басқару, жүйенің бағдарламалық қамтамасыз етуін пайдалануға беру және оған техникалық қызмет көрсету, бұзу әрекеттерінің мониторингі, қауіпсіздікті бұзуды анықтау және жою:

1. Пайдаланушыларды және қолжетімділік құқығын басқару дегеніміз пайдаланушыларды жүйеге қосу, алып тастау, пайдаланушыларды авторлау адекватты механизмі болуын кепілдендіру, жүйеге қол жетімділікті пайдаланушылар тек өздеріне қажетті ғана ресурстарға ене алатындай етіп баптауды білдіреді.
2. Жүйенің бағдарламалық қамтамасыз етілуін пайдалануға беру және оған техникалық қызмет көрсету дегеніміз бағдарламалық қамтамасыз етуді және бағдарламалық қамтамасыз етуді байланыстырушы жүйелерді орнатуды, қауіпсіздіктің бұзылуын болдырмайтын олардың конфигурациясын, сонымен қатар, нұсқаларды үнемі жаңартып отыруды, қауіпсіздік проблемасы анықталғаннан кейін жойып отыратын бағдарламалық қамтамасыз етуді түзетулерді білдіреді.
3. Шабуылдар мониторингі, шабуыл салдарын анықтау және жою шаралары жүйеге рұқсат етілмеген ену әрекеттерін қадағалау, оларды анықтау және шабуылдарға қарсы тұру стратегиясын енгізу, сонымен қатар, сыртқы шабуылдан кейін жүйе қалыпты жұмысына қайтып оралуы үшін қалпына келтіру операцияларын білдіреді.

Ішкі шабуылдар және психологиялық шабуылдар

Ішкі шабуылдар – бұл сенім білдірілген тұлғаның (штат қызметкері) сол сенімді теріс пайдалана отырып бұзу әрекеттері. Мысалы, ауруханада жұмыс істейтін медбике өзі күтетін сырқаттардың медициналық карталарына рұқсат алуы мүмкін. Ішкі шабуылдарды тоқтату өте қиын, себебі қосымша қауіпсіздік шараларын пайдалану жүйенің барлық пайдаланушыларының жұмысына, оның ішінде өз қызмет жағдайын теріс пайдаланбайтын қызметкерлердің жұмысына да кедергі келтіруі мүмкін.

Психологиялық шабуылдар – бұл өкілетті пайдаланушыларды тіркеу деректерін алу мақсатында алдау тәсілі болып табылады. Мұндай жағдайда бұзушы өзін штат қызметкері ретінде көрсетіп жүйеге рұқсат алуы мүмкін.

http://www.SoftwareEngineering-9.com/Web/SecurityEng/insiders.html

Қауіпсіздікті басқару өте маңызды аспект болып табылады, алайда әдетте оны қосымшаның қауіпсіздігін қамтамасыз ету бөлігі ретінде қарастырмайды. Өз кезегінде, қосымшаның қауіпсіздігін қамтамасыз ету – бұл қолайлылықтың және бюджет қаражатының шектеулерін ескере отырып әрі қауіпсіз, әрі функционалды жүйені жобалау. Аталған үдерістің бөлігі «тиімді басқаруды жобалау» болып табылады, соған сәйкес жүйелер жүйеге шабуылға алып келетін басқару қателігі ықтималдығы төмендетіліп жобаланады.

Критикалық жүйе және орнатылған жүйе жағдайында қолданбалы жүйені қолдайтын инфрақұрылымды таңдау стандарттық практика болып саналады. Мысалға, орнатылған жүйені әзірлеушілер әдетте, орнатылған қосымшаны өздеріне қажетті құралдармен қамтамасыз ететін шынайы уақыт операциялық жүйесін қолданады. Мұнымен бірге, белгілі әлсіз орындар және қауіпсіздікке талаптар ескерілуі мүмкін. Бұл қауіпсіздікті қамтамасыз ету үшін біртұтас тәсіл қажет екенін білдіреді. Қосымшаның қауіпсіздік талаптары инфрақұрылымның немесе сол қосымшаның өзінің көмегімен іске асырылуы мүмкін. Алайда, қолданбалы жүйелер әдетте қолданыстағы инфрақұрылымды (операциялық жүйелер, деректер базасы және т.б.) пайдаланатын ұйымдарда іске асырылады. Осылайша, жүйені әзірлеу үдерісінде аталған инфрақұрылымды пайдалану тәуекелін және оның қауіпсіздігінің ерекшеліктерін ескеру қажет.

14.1. Қауіпсіздікті бұзу тәуекелдерін басқару

Қауіпсіздікті бұзу тәуекелдерін басқару және бағалау қауіпсіздікті тиімді қамтамасыз ету үшін өте маңызды. Тәуекелдерді басқару – жүйе активіне шабуылдар нәтижесінде туындауы мүмкін жоғалтуларды бағалау, сонымен қатар мұндай жоғалтуларды қысқарта алатын қауіпсіздікті қамтамасыз ету шараларын жүргізу құнын аталған жоғалтумен салыстыру. Несиелік карталар шығаратын компания үнемі осымен айналысады. Несиелік картаға байланысты алаяқтыққа қарсы күресу мақсатында жаңа технологиялар енгізу қатысты түрде жеңіл. Алайда, көбінесе алаяқтыққа қарсы жаңа технологиялар сатып алып енгізуге қарағанда, пайдаланушыларға алаяқтардан келген шығындарын өтей салған арзанырақ түседі. Технология құнының төмендеуі, шабуылдар санының артуына қарай аталған теңгерім өзгеруі мүмкін. Мысалға, қазіргі уақытта несиелік карталар шығаратын компания ақпаратқа магниттік жолақ арқылы емес чиптің көмегімен код қояды. Бұл жалған карта жасауды әлдеқайда қиындатады.

Тәуекелдерді басқару көбіне техникалық емес іскерлік сипатта болады, сондықтан, бағдарламалық қамтамасыз ету әзірлеушілері жүйеге орнатылатын жүйеге бақылау құралдарына қатысты шешім шығаруға міндетті емес. Қауіпсіздікті қамтамасыз ету шығындарын бекітуді немесе қауіпсіздікті қамтамасыз ету бойынша әсерлі процедуралар жетіспеулігі нәтижесіне ұшырау мүмкіндігін жоғарғы басшылық шешеді. Бағдарламалық қамтамасыз ету әзірлеушілерінің рөлі қолдағы деректер негізінде әзірленген қауіпсіздік мәселесі жөніндегі ұсынымдар ғана берумен шектеледі. Осылайша, олар да тәуекелді басқару

үдерісінің маңызды қатысушылары болып табылады. *12-тарауда* айтылғандай, бағалау және тәуекелді басқару үдерісі үшін критикалық деректерді ұсыну ұйымның қауіпсіздікті қамтамасыз ету саясатының бөлігі болуы тиіс. Ұйымның қауіпсіздікті қамтамасыз ету саясаты барлық жүйелерге таралуы және неге рұқсат беріледі, неге берілмейтінін белгілеуі керек. Қауіпсіздікті қамтамасыз ету саясаты қауіпсіздік жүйесі әрқашан орындауы тиіс шарттарды белгілейді және туындауы мүмкін тәуекелдер мен қатерлерді тануға көмектеседі. Осылайша, қауіпсіздікті қамтамасыз ету саясаты неге рұқсат берілетінін, неге берілмейтінін белгілейді. Қауіпсіздікті қамтамасыз ету үдерісінде Сізге аталған саясатты жүзеге асырудың механизмдерін әзірлеу қажет.

Тәуекелдерді бағалау белгілі жүйені сатып алу туралы шешім қабылданбастан бұрын басталуы тиіс және жүйені әзірлеудің барлық үдерісінде, тіпті жүйе пайдалануға енгізілгеннен кейін де жалғасады (Альбертс и Дорофи, 2002). Сол сияқты, *12-тарауда* тәуекелдерді бағалау сатылы үдеріс болып табылатыны көрсетілген:

1. *Тәуекелдерді алдын ала бағалау.* Бұл сатыда толық жүйелік талаптарға, жүйе құрылымына, іске асыру технологиясына қатысты шешімдер әлі қабылданбаған. Аталған бағалау үдерісінің мақсаты үнемді шығындар деңгейінде қауіпсіздіктің адекватты деңгейіне қол жеткізу мүмкіндігін анықтау болып табылады. Егер бұл мүмкін болса, Сіз жүйенің қауіпсіздігіне нақты талаптарды анықтай аласыз. Сіз жүйенің немесе көп мәрте пайдаланылатын жүйе компоненттері немесе байланыстырушы бағдарламалық қамтамасыз ету ие болатын бақылау құралының ықтимал кемшіліктері туралы ақпаратты біле алмайсыз.
2. *Тіршілік циклі тәуекелдерін бағалау.* Мұндай тәуекелдерді бағалау жүйені әзірлеу циклі үдерісінде жүзеге асырылады, ал ол туралы ақпарат техникалық жүйені жобалау және әзірлеу шешімдері негізінде алынады. Бағалау нәтижелері жүйелік талаптардың өзгеруіне және жаңа талаптардың қосылуына алып келуі мүмкін. Мәлім және ықтимал кемшіліктер анықталады, бұл мәліметтер жүйенің функционалдығы, іске асырылуы, тестіленуі және пайдалануға берілуіне қатысты ақпараттық шешімдер қабылдауға пайдаланылады.
3. *Пайдалану тәуекелдерін бағалау.* Жүйе пайдалануға берілгеннен кейін де жүйенің қалай қолданылатынын назарға алу үшін тәуекелдерді бағалауды одан әрі жалғастыру қажет және өзгертілген талаптар туралы ұсыныс дайындау керек.

Жүйені жобалау үдерісінде жасалған, пайдаланушылық талаптар туралы жорамалдар қате болуы мүмкін. Ұйымдастырушылық өзгерістер енгізу жүйенің ең басында жоспарланғаннан өзгеше пайдаланылатынын білдіруі мүмкін. Осылайша, пайдаланушылық тәуекелдерді бағалау жүйенің эволюциясына қарай іске асыру қажетті қауіпсіздікке жаңа талаптардың туындауына алып келеді. Тәуекелдерді алдын ала бағалаудың негізгі міндеті қауіпсіздік талаптарын анықтау болып та-

былады. *12-тарауда* қауіпсіздікке талаптардың бастапқы жинағы тәуекелдерді алдын ала бағалау нәтижесінде алынуы мүмкін екені көрсетілген. Сондықтан берілген бөлімде тіршілік циклі тәуекелін және пайдаланушылық тәуекелдерді бағалауға ерекше назар аударылады. Бұл технологиялардың жүйені анықтауға және әзірлеуге, сонымен қатар, жүйенің қандай түрде қолданылатынына қандай әсер ететінін көрсетеді.

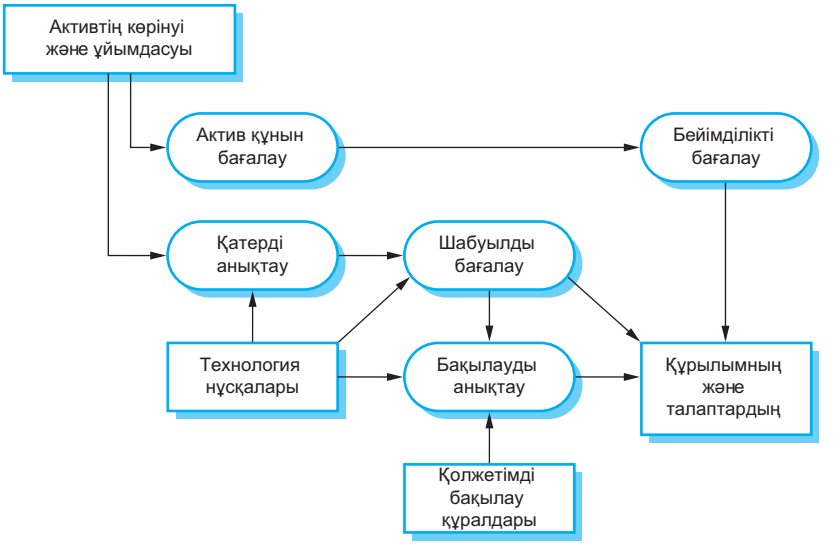
Тәуекелдерді бағалау жүргізу үшін Сізге жүйеге төнуі мүмкін қауіптерді анықтау қажет. Мұндай анықтаудың тәсілдерінің бірі «теріс пайдалану оқиғалары» кешенін әзірлеу болып табылады (Александр, 2003; Синдр және Опдаль, 2005). Біз бұрынырақ жүйемен типтік өзара әрекетті жүйе талаптарын анықтауға қалай пайдалануға болатынын талқылағанбыз. Теріс пайдалану оқиғалары – бұл жүйемен зиянды өзара әрекетті көрсететін сценарий.

Сіз оларды ықтимал қауіптерді анықтауға және талқылауға және тиісінше, қауіпсіздік жүйесіне талаптарды анықтауға қолдана аласыз. Олар жүйе талаптарын анықтау кезінде басқа оқиғалармен қатар пайдаланылуы мүмкін.

Пфлегер және Пфлегер (2007) қатерлерді төрт класқа бөледі, олар мүмкін болатын теріс пайдалану оқиғаларын анықтаудың жөнелту нүктесі ретінде қолданылуы мүмкін. Бұл кластар төменде көрсетілген:

1. Ұстап қалу қатері, ол бұзушыға активтерге рұқсат алуға мүмкіндік береді. Клиниканың деректер базасы жүйесін теріс пайдаланудың ықтимал оқиғасы бұзушы атақты адам болып табылатын емделушінің медициналық картасына қол жеткізуі болып табылады.
2. Қызметті бұзу қатері, бұзушыға жүйенің бөліктерінің бірін істен шығаруға мүмкіндік береді. Осылайша, теріс пайдаланудың ықтимал оқиғасы жүйенің деректер базасы серверіне «қызмет көрсетуден бас тарту» типтес шабуыл болуы мүмкін.
3. Модификация қатерлері, бұзушыға жүйе активін бұрмалауға мүмкіндік береді. Клиниканың деректер базасы мысалында бұзушы емделушінің медициналық картасындағы ақпаратты өзгертуі мүмкін.
4. Жалғандық қатері, бұзушыға жүйеге жалған ақпарат енгізуге мүмкіндік береді. Теріс пайдаланудың мұндай оқиғасы банк жүйелерімен оқиға кезінде іске асырылады, бұзушының шотына ақша қаражатын аудару үшін жүйеге жалған транзакциялар туралы ақпарат енгізілуі мүмкін.

Тәуекелдерді алдын ала бағалау сатысында пайдалану оқиғалары аса әсерлі емес, алайда, тіршілік циклі тәуекелін және пайдалану тәуекелдерін талдау кезінде пайдаланылуы мүмкін. Олар жүйеге гипотетикалық шабуылдың симуляциясы және жобалаудың белгілі шешімдері кезінде қауіпсіздіктің бұзылуы салдарын бағалау үшін пайдалы базаны қамтамасыз етеді.



14.2-сурет. Тіршілік циклі тәуекелін талдау

14.1.1 Тіршілік циклі тәуекелін талдау

Ұйымның қауіпсіздікті қамтамасыз ету саясатына сәйкес тәуекелдерді алдын ала бағалау жүйе қауіпсіздігіне ең маңызды талаптарды анықтауы тиіс. Олар қауіпсіздікті қамтамасыз ету саясатының берілген қосымшада қалай іске асырылуы қажеттігін, қорғанысты қамтамасыз етуге пайдаланылатын тәсілді анықтайды және қорғалатын активтерді көрсетеді. Алайда, тиісті түрдегі қауіпсіздікті қамтамасыз ету үшін бөлшектерге назар аудару қажет. Қауіпсіздікке ең басында белгіленген талаптар қауіпсіздікке әсер ететін барлық бөлшектерді ескере алмайды.

Тіршілік циклі тәуекелін талдау жүйенің қауіпсіздігіне ықпалын тигізетін жүйені әзірлеу мен іске асыру бөлшектерін анықтайды. Тіршілік циклі тәуекелін бағалау мен тәуекелдерді алдын ала бағалау арасындағы айырмашылық осыған негізделеді. Тіршілік циклі тәуекелін бағалау қауіпсіздікке қолданыстағы талаптардың түсіндірілуіне әсер етеді, жалпы жүйені әзірлеуге ықпал етеді.

Берілген сатыда тәуекелдерді бағалау кезінде Сізге нені қорғау қажеттілігі туралы көп мөлшерде ақпарат қажет. Сонымен қатар, Сізде жүйенің әлсіз орындары және кемшіліктері туралы нақты ақпарат болуы тиіс. Бұл кемшіліктердің кейбірі жобалауда қабылданған шешімдерге байланысты болады. Мысалға, құпия сөз енгізуге байланысты жүйенің кемшілігі авторланған пайдаланушының өз құпия сөзін авторланбаған пайдаланушы ретінде аша беретіні болып табылады. Егер ұйым саясатында бағдарламалық қамтамасыз етуді әзірлеуді C тілінде қаласа, онда бағдарламалық қамтамасыз етуде бірқатар кемшіліктер орын алады, себебі аталған тіл массивтер шекарасын тексеруді жүргізуге мүмкіндік бермейді. Қауіпсіздіктің бұзылуы тәуекелін бағалау талаптарды әзірлеуден жүйені пайдалануға берген-

ге дейінгі тіршілік циклінің барлық операцияларының бөлігі болуы тиіс. Үдеріс жүйені жобалау кемшіліктерін бағалау және анықтау бойынша шаралармен толықтырыла отырып, тәуекелдерді алдын ала бағалау үдерісіндегі сияқты схемамен жүргізіледі.

Тәуекелдерді бағалау нәтижесі жүйені жобалау және іске асыруға тікелей ықпал ететін немесе оны пайдалану әдістерін шектейтін жобалау бойынша кешенді шешімдер болып табылады.

12.9-суретте сипатталған тәуекелдерді алдын ала талдау үдерісіне негізделген тіршілік циклі тәуекелін талдау моделі *14.2-суретте* көрсетілген. Аталған үдерістер арасындағы негізгі айырмашылық: Сіз енді ақпаратты бөлу және көрсету туралы ақпаратқа, қорғалуы тиіс жоғарғы деңгейдегі активтер деректер базасының ұйымдасуы туралы ақпараттарға ие болдыңыз. Сізге сонымен қатар, жобалау мен әзірлеуге қатысты маңызды шешімдер (мысалы, көп мәрте пайдаланылатын бағдарламалық қамтамасыз ету, инфрақұрылымдарды қорғау және басқару құралдары және т.б.) де белгілі. Аталған ақпараттар негізінде Сіздің талдауыңыз қауіпсіздік талаптарына, жүйе құрылымына енгізу қажетті жүйенің маңызды активтерін қорғауға қажетті өзгерістерді белгілейді.

Бұдан әрі қауіпсіздікке талаптарға ақпаратты бөлу және көрсету әдістеріне қатысты шешімдердің қандай әсер ететіні туралы екі мысал келтірілген:

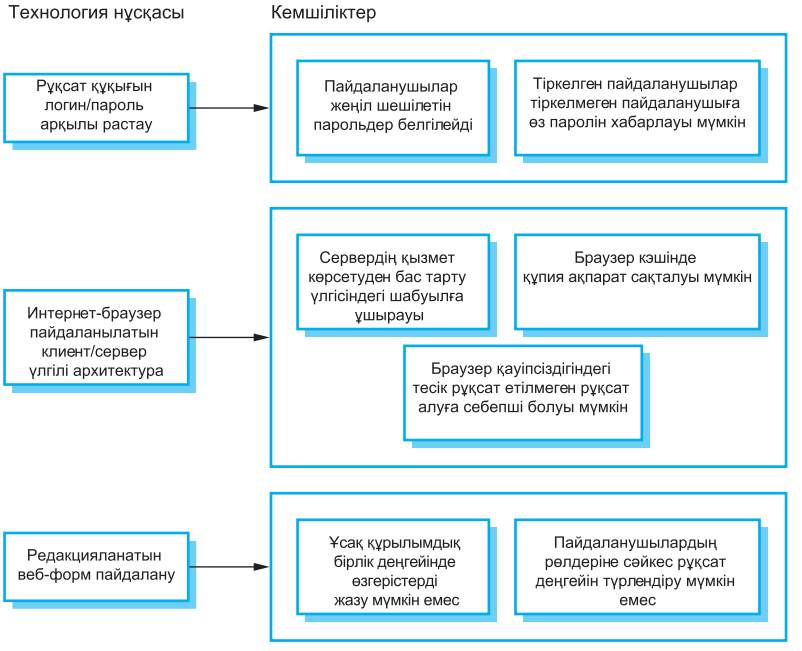
1. Сіз емделуші туралы жеке ақпаратты және қабылданатын дәрілер туралы ақпаратты бөлу туралы шешім қабылдай аласыз, бұл екі мұрағаттың арасында байланыстырушы буын ретінде кілт жүреді. Емделу туралы ақпарат емделуші туралы жеке ақпаратпен салыстырғанда құпиялығы төмен, сондықтан маңызды қорғауды талап етпейді. Егер кілт жақсы қорғалған болса, бұзушы тек пайдасыз ақпаратқа рұқсат алады, бірақ оны емделушінің тұлғасымен байланыстыра алмайды.
2. Сессия басында барлық емделушілердің медициналық карталары көшірмелерін дербес клиент-жүйеде сақтау туралы шешім қабылданды делік. Бұл егер сервер қолжетімсіз болса да жұмысты жалғастыра беруге мүмкіндік береді, яғни денсаулық сақтау қызметкерлері тіпті желілік байланыс болмаған күннің өзінде де ноутбуктердің көмегімен емделушілердің медициналық карталарына қол жеткізе алады. Алайда, енді Сізде медициналық карталардың екі кешені бар, олардың әрқайсысы қорғауды қажет етеді, оның үстіне клиенттік көшірмелер қосымша тәуекелдерге бейім (мысалы, егер ноутбук ұрланатын болса). Осылайша, Сізге қандай бақылау құралы тәуекелді қысқарта алатыны жайлы ойлану керек. Мысалы, ноутбуктегі медициналық карталар шифрленуі мүмкін.

Әзірлеу технологиясына қатысты шешім қауіпсіздікке қалай әсер ететінін көрсету үшін денсаулық сақтау саласы қызметкерлері еркін сатылымдағы ақпараттық жүйені қолдана отырып емделушілер туралы деректер базасын құруды ұйғарды делік. Аталған жүйе ол пайдаланылатын әрбір клиника үшін конфигура-

цияланууы тиіс. Аталған шешімнің қабылдану себебі, ол әзірлеуге төмен шығындар және пайдалануға жылдам енгізу кезінде жоғары функционалдықты ұсынады.

Сіз қолданыстағы жүйені екінші қайтара пайдалану жолымен қосымша әзірлегенде, Сізге аталған жүйені әзірлеушілер қабылдаған жобалау шешімдерімен келісу қажет. Олар төмендегідей жобалау шешімдерін қабылдады деп ойлайық:

1. Жүйені пайдаланушының рұқсат құқығы логин және пароль комбинациясы көмегімен тексеріледі. Рұқсат құқығын басқа тексеру әдістері қолданбайды.
2. Архитектура жүйесі үлгісі: клиент-сервер, клиенттер өз дербес компьютерлерінде орнатылған стандарттық Интернет-браузер көмегімен деректерге рұқсат алады.
3. Ақпарат пайдаланушыларға редакцияланатын веб-форма ретінде ұсынылады, Олар ақпараттың нысанын өзгерте алады және редакцияланған ақпаратты серверге жолдайды.



14.3-сурет. Таңдап алынған технология нұсқасына байланысты кемшіліктер

Жобалау туралы аталған шешімдер көп номенклатуралы жүйелер үшін толық тиімді, алайда тіршілік циклі тәуекелін талдау оларда бірталай кемшіліктер бар болуы мүмкін екенін көрсетеді. Мүмкін болатын кемшіліктер *14.3-суретте* көрсетілген.

Кемшіліктер анықталғаннан кейін Сізге бұл тәуекелдерді қысқарту үшін қандай қадам жасау қажеттігі туралы шешім қабылдау керек. Көбінесе бұл қауіпсіздікке қосымша талаптарға немесе жүйені қолданудың операциялық

үдерістеріне қатысты шешімдер қабылдауды білдіреді. Өкінішке орай, сипаттамалық кемшіліктерді жоюға ұсынылуы мүмкін талаптарды сипаттау үшін бұл кітап та жетпейді, алайда, біз мұндай талаптардың бірнеше мысалдарын келтіре аламыз:

1. Құпия сөздердің сенімділігін тексеретін бағдарламаның күнделікті жұмысын ұйымдастыру. Ол сөздіктерде бар пайдаланушы құпия сөздерін тануы тиіс, ал құпия сөзінің қорғалу деңгейі төмен пайдаланушылардың есімдерін жүйелік әкімгерге хабарлауы тиіс.
2. Жүйеге тек жүйелік әкімгерлер тіркеген және бекіткен компьютерлер ғана рұқсат алуы керек.
3. Барлық клиенттер жүйелік әкімгер бекіткен бір ғана браузерді пайдаланулары тиіс.

Сатылымда бар жүйе пайдаланылатындықтан, құпия сөздердің сенімділігін тексеретін бағдарламаны қолданбалы жүйенің өзіне орнатуға болмайды. Сондықтан, түрлі жүйелерді пайдалану қажет. Құпия сөздерді тексеру бағдарламалары белгіленетін құпия сөздердің сенімділік деңгейін талдайды және пайдаланушыларға сенімсіз құпия сөз таңдағаны туралы хабарлайды. Осылайша, сенімсіз құпия сөздер белгіленгеннен кейін бірден анықталады, бұл пайдаланушының құпия сөзін неғұрлым күрделіге ауыстыруына кепілдік беретін қажетті шараларды қабылдауға мүмкіндік береді.

Екінші және үшінші талаптар барлық пайдаланушылардың жүйеге кіруді тек қана бір браузердің көмегімен жүргізе алатынын білдіреді. Жүйені пайдалануға енгізгеннен кейін Сіз сенімді браузер таңдап, оны клиенттердің компьютеріне орната аласыз. Қауіпсіздіктің жаңартылу үдерісі жеңілдетіледі, себебі жүйе қауіпсіздігі кемшіліктері анықталып жойылғаннан кейін әр түрлі браузерлерді жаңарту қажеттілігі туындамайды.

14.1.2 Пайдаланушылық тәуекелдерді бағалау

Қауіпсіздік тәуекелдерін бағалау туындайтын тәуекелдерді және өзгерістерді анықтау мақсатында жүйенің барлық қызмет ету мерзімінде жүзеге асырылады. Аталған үдеріс пайдалану тәуекелдерін бағалау деп аталады. Жаңа тәуекелдер жүйе талаптарының өзгеруінен, жүйе инфрақұрылымының немесе жүйе қолданылатын орта өзгерістерінен туындауы мүмкін.

Пайдалану тәуекелін бағалау үдерісі жүйені қолдану ортасы туралы жаңа ақпаратты енгізе отырып, тіршілік тәуекелін бағалау үдерісімен ұқсас жүреді. Орта маңызды рөл атқарады, себебі орта ерекшеліктері жүйенің жаңа тәуекелдеріне алып келуі мүмкін. Жүйе пайдаланушыларды үнемі жұмысынан алаңдататын ортада қолданылады делік. Мұнда пайдаланушының өз жұмыс орнын қараусыз тастап кетуі тәуекелі бар деген сөз. Тіркелмеген тұлға мұндай жағдайда пайдаланушының жоқ екенін пайдаланып жүйе ақпаратына енуге мүмкіндік алады. Аталған жағдайға

жол бермес үшін компьютер экрандарын белгілі уақыт аралығында автоматты түрде бұғат салынып құпия сөзбен қорғау қажеттілігінің туындауына алып келеді.

14.2. Қауіпсіз жүйелерді жобалау

Жалпы алғанда, жүйе іске асырылғаннан кейін оның қауіпсіздігін көтеру қиын деген тұжырым дұрыс болып саналады. Сондықтан жүйені әзірлеу кезінде барлық қауіпсіздік мәселелерін ескеру қажет. Берілген бөлімде біз жүйені әзірлеу проблемасына баса назар аударамыз, себебі компьютерлік қауіпсіздік туралы кіаптарда бұл мәселеге тиісті көңіл бөлінбеген. Іске асыру қателіктері және проблемалары қауіпсіздікке елеулі әсер етеді, алайда олар көбінесе нақты пайдаланылатын технологияға байланысты. Біз Виег және МакГровтың (2002) бағдарламалау кезінде қауіпсіздікті қамтамасыз етудің негізгі принциптеріне арналған кітабымен танысуға кеңес береміз. Аталған кітапта қауіпсіз жүйелерді жобалау сұрақтарына негізгі назар аударылады:

1. Архитектура деңгейінде жобалау – жүйенің қауіпсіздігіне архитектура деңгейінде жобалау қандай әсер етеді?
2. Қабылданған нормалар – қауіпсіз жүйелерді жобалауға қандай нормалар қабылданған?
3. Пайдалануға енгізуді жоспарлау – жүйені пайдалануға енгізгеннен кейін кемшіліктердің туындауын болдырмас үшін жүйеге қандай қолдау құралдарын енгізу қажет?

Қызмет көрсетуден бас тарту үлгісіндегі шабуылдар

Қызмет көрсетуден бас тарту үлгісіндегі шабуылдар қызметтер көрсетуге өте көп мөлшерде сұраныстар көмегімен желілік жүйені зақымдауға тырысады. Бұл жүйеге өзінің шамасынан артық жүктеме түсіреді. Мұнымен бірге, олар жүйе қызметтерін ұсынуға заңды сұраныстардың болмауын ескереді. Нәтижесінде, жүйе орасан жүктемені көтере алмағандықтан немесе сұраныстар легін тоқтату үшін жүйе басқарушылары ажыратып тастағандықтан қолжетімсіз болады.

http://www.SoftwareEngineering-9.com/Web/Security/DoS.html

Әрине, бұлар қауіпсіздік үшін маңызды сұрақтардың барлығы емес. Барлық қосымшалар бір-бірінен ерекшеленеді, қауіпсіздікті қамтамасыз еткенде олардың мақсатын, маңызын, қызмет атқару ортасын ескеру қажет. Егер Сіз әскери жүйені әзірлеп жатсаңыз, Сізге олардың қауіпсіздік деңгейі біліктілік моделін қабылдау қажет (құпия, өте құпия және т.б.). Егер Сіз жеке ақпаратпен жұмыс істейтін жүйені әзірлеп жатсаңыз, Сізге жеке деректерді басқаруға шектеулер қоятын жеке деректерді қорғау туралы заңнамалық ережелерді назарға ұстаған дұрыс.

Функционалдық сенімділік және қауіпсіздік арасында өзара тығыз байланыс орнатылған. Функционалдық сенімділікті қамтамасыз етуге іргетастық маңызға ие артық және түрлі бейнедегі құралдарды пайдалану жүйенің әзірлеу мен іске асыру ерекшеліктеріне бағытталған шабуылдарға төтеп беріп, кейін қалпына келуге қабілеттігін білдіреді.

Қолжетімділіктің жоғарғы деңгейін қолдайтын механизмдер қызмет көрсетуден бас тарту сияқты жүйені зақымдауға және оның қалыпты жұмысын тоқтатуға бағытталған шабуылдардан кейін жүйеге қалпына келуге көмектесе алады.

Қауіпсіз жүйелер әзірлеу бірқатар бітімгершілікті көздейді. Қауіпсіздікті қамтамасыз етудің шабуылдардың табыстылығын төмендететін көптеген шараларын әзірлеу әрине мүмкін. Алайда, қауіпсіздікті қамтамасыз ету шаралары қосымша есептеулердің үлкен көлемін талап етеді, сондықтан жүйенің жұмысқа қабілеттілігіне кері әсерін тигізеді. Мысалы, Сіз құпия ақпараттың ашылу ықтималдығын төмендету үшін оған құпия код қойдыңыз, сонымен ақпаратты пайдаланушыларға оның ашылуын күтуге тура келеді де, жұмысының жылдамдығы төмендейді.

Сол сияқты, қауіпсіздік пен қолайлылық арасында сәйкессіздік бар. Қауіпсіздікті қамтамасыз ету шаралары пайдаланушыдан қосымша ақпараттарды еске сақтауды және ұсынуды талап етеді (мысалға, бірнеше құпия сөздер). Бірақ, кейде пайдаланушы аталған ақпаратты ұмытып қалады, бұл қосымша қауіпсіздік шаралары оған жүйені пайдалануға мүмкіндік бермейді. Осылайша, әзірлеушілерге қауіпсіздік, жұмысқа қабілеттілік, қолайлылық арасында белгілі теңгерім табу қажет. Аталған теңгерім жүйе үлгісіне және оның пайдаланылу ортасына байланысты болады. Мысалға, әскери жүйеде пайдаланушылар қауіпсіздік деңгейі жоғары жүйелермен таныс, сондықтан үнемі тексерулер талабына сәйкес үдерістерді оң қабылдап сақтауға бейім. Ал акцияларды сату жүйесінде жұмыста ешқандай үзіліске жол берілмейді.

14.2.1 Архитектура деңгейінде жобалау

11-тарауда айтылғандай, бағдарламалық қамтамасыз етудің архитектурасын таңдау жүйенің тәуелсіз қасиеттеріне кешенді әсер етуі мүмкін. Егер сәйкес келмейтін архитектура қолданылса, онда жүйелік ақпараттық тұтастығын, құпиялығын сақтау немесе жүйенің қолжетімділік қажетті деңгейіне кепілдік беру өте қиын міндетке айналуы мүмкін.

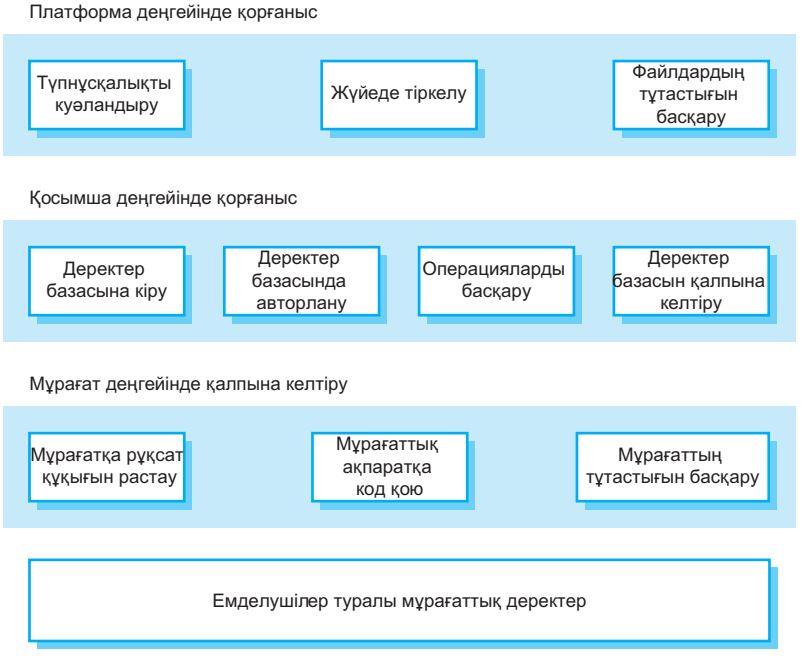
Жоғарғы қауіпсіздік деңгейін қолдайтын жүйе архитектурасын әзірлеу кезінде екі негізгі мәселені ескеру қажет:

1. Қорғаныс – критикалық активтері сыртқы шабуылдан қорғалуы үшін құрылым қалай ұйымдастырылуы тиіс?
2. Бөлу – табысты шабуылдың салдары төмендеуі үшін жүйе активтері қалай бөлінуі керек?

Аталған сұрақтар жанжалды болуы мүмкін. Егер Сіз активтеріңізді бір жерге қойсаңыз, оның айналасында бірнеше қабат қорғаныс құра аласыз. Алайда егер қорғаныс шабуылға төтеп бере алмаса, Сіздің барлық активтеріңіз зақым шегеді. Қорғаныстың бірнеше қабатын қосу жүйенің қолайлылығына кері әсер етеді, яғни мұндай жүйеге қолайлылық, жұмысқа қабілеттілік қсиеттерін қанағаттандыру қиын болады.

Екінші жағынан, егер Сіз активтеріңізді бөліп қойсаңыз, олардың қорғаныс құны артады, себебі әрбір көшірме үшін қорғаныс жүйесін іске асыру қажет болады. Әдетте бұл Сізге бірнеше қорғаныс қабатын құруға мүмкіндік бермейді. Бірақ шабуыл табысты болғанда Сіз деректерді толық жоғалтудан зардап шекпейсіз. Ақпараттық активтердің сенімді бөлінуі, дубликатар біреуі зақымданса да, екіншісін қолдануға мүмкіндік береді. Сонымен қатар, егер ақпарат құпия болса, бұзушының бұл ақпаратқа қол жеткізу тәуекелі ұлғаяды.

Емделушілер туралы деректерді сақтауда архитектурасы орталықтандырылған архитектурасы бар деректер базасын қолдану тиімді болып саналады. Қауіпсіздікті қамтамасыз ету үшін Сіз көп деңгейлі архитектураны қолданасыз, критикалық активтер төменгі деңгейде түрлі қорғаныс қабаттарымен қорғалған. *14.4-суретте* критикалық активтері жеке емделушілердің медициналық карталары болып табылатын деректер базасы жүйесі көрсетілген.



14.4-сурет. Көпдеңгейлі қорғаныс архитектурасы

Оларға рұқсат алу және түрлендіру үшін бұзушыға үш жүйе деңгейінен өту керек:

1. *Платформа деңгейіндегі қорғаныс.* Жоғарғы деңгей емделушілер туралы деректер базасы жұмыс істейтін платформаға қолжетімділікті бақылайды Әдетте ол пайдаланушы белгілі компьютерде тіркелуі қажеттігін білдіреді. Сонымен қатар, платформа жүйе файлдары, резервтік көшірмелер тұтастығын қолдайды.
2. *Қосымша деңгейінде қорғаныс.* Келесі қорғаныс деңгейі қосымшаның өзіне орнатылған. Ол пайдаланушы жүйеге рұқсат алу кезінде деректерді көру және өзгерту үшін тіркелуі тиіс екенін білдіреді. Қосымшаға қарай тұтастықты басқарудың қолдауы қолжетімді болуы мүмкін.
3. *Мұрағат деңгейінде қорғаныс.* Аталған деңгей белгілі мұрағаттарға рұқсат алу керек болғанда іске қосылады және пайдаланушының тиісті мұрағатта сұралатын операцияларды жүзеге асыруға құқығын тексеруді білдіреді.

Аталған деңгейдегі қорғаныс мұрағатқа файлдар диспетчері көмегімен енуді болдырмас үшін код тағайындауды да қарастырады. Криптографиялық бақылау сомасына негізделген тұтастықты тексеру мұрағатты жаңартудың стандартты механизмінен бөлек енгізілген өзгерістерді анықтай алады. Кез келген қосымшаның қажетті қорғаныс қабаты деректердің маңыздылығына байланысты. Барлық қосымшаларға мұрағат деңгейінде қорғаныс қажет емес. Қауіпсіздікті қамтамасыз ету үшін Сіз бір тіркеу дерегін екінші қайтара пайдалануға жол бермеуіңіз керек. Алайда пайдаланушыларға көптеген құпия сөздерді жаттау қиын. Осылайша Сізге жұмыстағы қолайлылықты қамтамасыз ету үшін қауіпсіздікті құрбан етуге тура келеді.

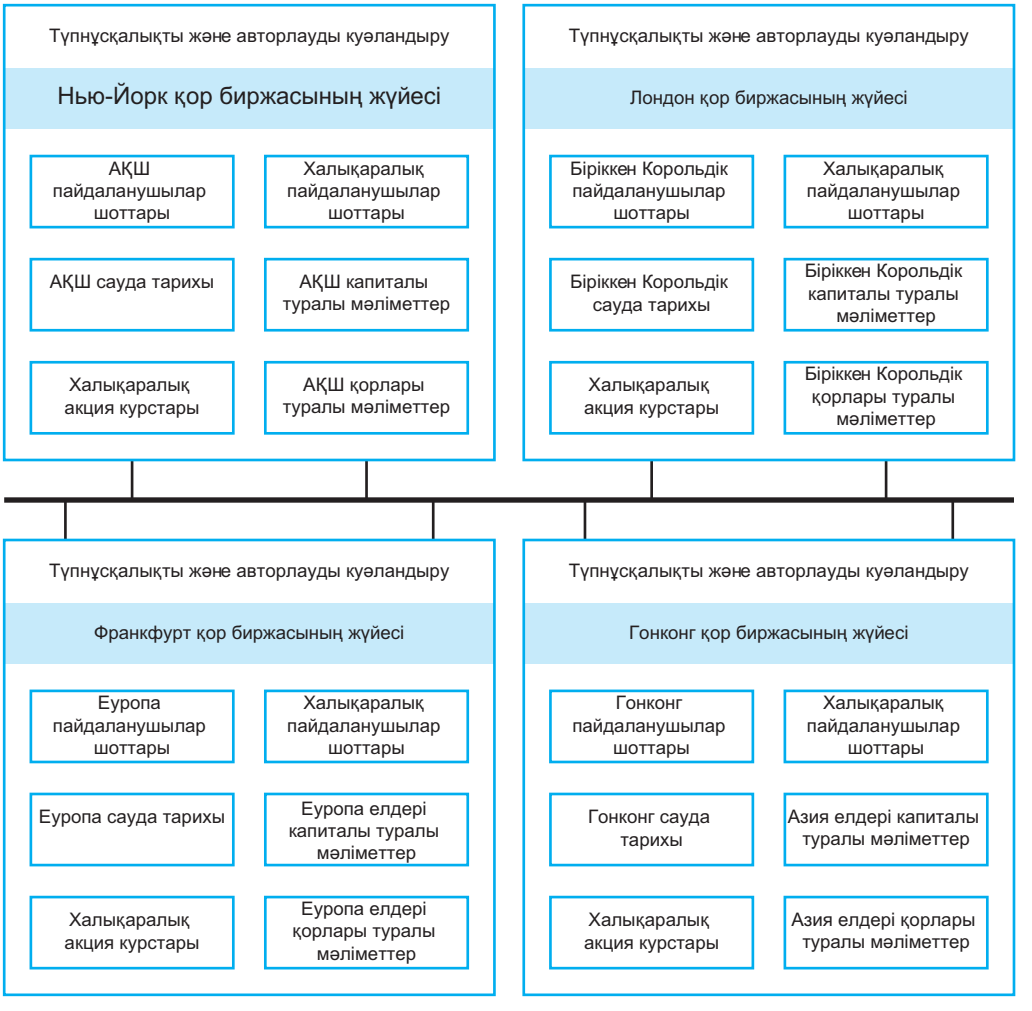
Егер деректерді қорғау критикалық талап болса, қорғаныс механизмі серверге орнатылуы тиіс, «клиент-сервер» архитектурасы пайдалануы тиіс. Алайда егер қорғаныс зақымданса, шабуылға байланысты шығындар, қалпына келтіру шығындары сияқты жоғары болуы мүмкін (мысалы, пайдаланушылардың барлық тіркеу деректері екінші рет берілуі тиіс). Мұндай жүйе қызмет көрсетуден бас тарту үлгісіндегі шабуылға осал, шабуыл нәтижесінде сервер жүктемесі көбеюі, жүйенің деректер базасына қолжетімділік барлығына бірдей жабылуы мүмкін.

Егер Сіз қызмет көрсетуден бас тарту үлгісіндегі шабуыл негізгі тәуекел деп санасаңыз, онда қосымшаның бөлінген архитектура пайдасына шешім қабылдауыңызға болады. 14.5-суретте көрсетілгендей жағдайда, жүйе активтері әрбірінің өзіндік қорғаныс механизмі бар түрлі платформалар арасында бөлінген. Бір торапқа жасалған шабуыл белгілі активтерге қолжетімділікті шектеуі мүмкін, алайда жүйе қызметін жалғастыра береді.

Деректердің көшірмелері жүйенің түрлі тораптары арасында тарауы мүмкін, бұл шабуылдан кейін қалпына келтіру үдерісін жеңілдетеді.

14.5-суретте Нью-Йорк, Лондон, Франкфурт және Гонконг нарықтарында акцияларды сату банкілік жүйесінің архитектурасы көрсетілген. Жүйеде әрбір нарық туралы ақпараттар жеке сақталады. Негізгі сауда қызметін қолдауға қажетті активтер (акция құны және пайдаланушылар шоттары) көшіріледі, ал оларға рұқсатты

кез келген тораптың көмегімен алуға болады. Егер белгілі бір торап шабуылға ұшырап, оған рұқсат жабылса, критикалық сауда қызметі басқа елге ауыстырылады және пайдаланушылардың оған рұқсаты болады.



14.5-сурет. Акцияларды сату жүйесінде активтердің бөлінуі

Біз қауіпсіздік пен жүйенің жұмысқа қабілеттілігі арасындағы теңгерім табу мәселесін жоғарыда талқыладық. Қауіпсіз жүйе әзірлеу проблемасы көптеген жағдайларда қауіпсіздік талаптарына лайық архитектура жұмысқа қабілеттілікке талаптарға жауап бере алмайтынына негізделеді. Қосымшаға ірі деректер базасының құпиялылығын сақтау және осы деректерге тез рұқсат алу талаптары қойылды делік. Жоғарғы қорғаныс дәрежесі бірнеше қорғаныс деңгейлерінің болатынын білдіреді, әрбір деңгей арасында байланыс орнатылады. Бұл жүйенің жұмысқа қабілеттілігін сөзсіз тежейді, деректерге қолжетімділікті баяулатады.

Егер құрылымның балама нұсқалары пайдаланылатын болса, онда қорғаныс талаптарын және құпиялылық талаптарын жүзеге асыру қиын және қымбат тұратын міндет болуы мүмкін. Мұндай жағдайда Сізге клиентпен сипаттамалық жанжалдарды талқылап, оны қалай шешуге болатыны туралы белгілі келісімге келу қажет.

14.2.2 Жобалау бойынша нұсқаулар

Жүйе қауіпсіздігін қамтамасыз етудің анық және тұрақты ережелері болмайды. Әр түрлі жүйе типтері жүйе иесінің талаптарын қанағаттандыратын қауіпсіздік деңгейіне қол жеткізуге қажетті түрлі техникалық құралдарды талап етеді. Көп жағдайда пайдаланушылардың түрлі топтарының пікірлері мен талаптары жүйенің қайсысы ұйғарымды, ал енді қайсысы қолайсыз екенін анықтайды. Мысалы, банкте, пайдаланушылар қауіпсіздіктің жоғары деңгейін күтеді, сондықтан да қауіпсіздікті қамтамасыз ету бойынша беймаза үдерістерге көнуге дайын болса, университетте мүлдем басқаша болады. Алайда, барлық жерлерде, жүйе қауіпсіздігі шешімдерін даярлау бойынша жалпы қолданылатын нұсқаулар да бар, олар қауіпсіз жүйелерді жобалаудың жалпы жұрт қабылдаған нормаларын қысқаша баяндайды. Бұдан әрі қарай суреттелетін жүйелерді даярлау кезінде қауіпсіздікті қамтамасыз ету бойынша жалпы нұсқаулардың екі түбегейлі мақсаты бар:

1. Олар бағдарламалық қамсыздандыруды жобалау бойынша командаға қауіпсіздік аспектілерін сезінуге көмектеседі. Бағдарламалық қамсыздандыруды әзірлеушілер көбінесе бағдарламалық қамсыздандырудың жұмыс қабілетін қамтамасыз етудің қысқа мерзімді мақсаттары мен оны сатып алушыларға жеткізуге назар аударады. Олар қауіпсіздік мәселесін оп-оңай ұмыт қалдыруы мүмкін. Аталған нұсқауларды білу – бағдарламалық қамсыздандыруды әзірлеу бойынша шешімдерді қабылдау кезінде қауіпсіздік мәселелері еске алынады дегенді білдіреді.
2. Олар жүйе валидациясы үдерісінде тексеріс параметрлерінің бақылаушы тізімі сапасында қолданылуы мүмкін. Жоғары деңгейдің осы жерде келтірілген нұсқаулары негізінде жүйе қауіпсіздігін қамтамасыз ету тәсілдеріне қатысты анағұрлым айрықша сұрақтар даярлануы ықтимал.

14.6-суретте қысқаша мазмұндалған жобалау бойынша 10 нұсқау түрлі деректер көзі негізінде әзірленген болатын (Шнайер, 2000; Виега мен МакГров, 2002; Вилер, 2003). Біздің кітабымызда бағдарламалық қамсыздандыру спецификациялары мен әзірлеу үдерістерінде қолданылатын нұсқауларға басты назар бөлінеді. «Жүйенің ең әлсіз буынының қауіпсіздігін қамсыздандыр», «Қисынсыз қиындатуды мойындама» және «Бүркеме есебінен қауіпсіздікті қамтамасыз етуден аулақ бол» тәрізді анағұрлым жалпы принциптер де маңызды, алайда олар жобалау бойынша шешім қабылдау үдерісімен тікелей байланысты емес.

Қауіпсіздікті қамсыздандыру бойынша нұсқаулар
1. Қауіпсіздік бойынша шешімдерді қауіпсіздікті қамсыздандырудың нақты саясатымен негіздеңіз.
2. Істен шығуы барлық жүйенің бас тартуына әкелетін құрамдастардан аулақ болыңыз.
3. Бас тарту қауіпсіздігін қамсыздандырыңыз.
4. Қауіпсіздік пен жинақтылықты теңдестіріңіз.
5. Пайдаланушының әрекеттерін тіркеңіз.
6. Қауіпті төмендету үшін көпсандылық пен әртүрлілікті пайдаланыңыз.
7. Барлық ендірмелердің дұрыстығын тексеріңіз.
8. Активтерді блоктар бойынша бөліңіз.
9. Пайдалануға беру қарапайымдылығын қамтамасыз ете отырып әзірлеңіз.
10. Қалпына келтірулікті қамтамасыз ете отырып әзірлеңіз.

14.6-сурет. Қауіпсіз жүйелерді жобалау бойынша нұсқаулар

1-нұсқау: Қауіпсіздік бойынша шешімдерді қауіпсіздікті қамсыздандырудың нақты саясатымен негіздеңіз.

Қауіпсіздікті қамсыздандыру саясаты бұл – ұйым үшін қауіпсіздікті қамтамасыз етудің іргелі шарттарын анықтайтын жоғары деңгейлі құжат. Ол «қалай?» деген сұрақтан гөрі, «не?» деген сұраққа жауап береді және сондықтан да қауіпсіздікті қолдау мен қамсыздандыру механизмдерін саясат анықтамауы керек. Теория жүзінде, қауіпсіздікті қамсыздандыру саясатының барлық аспектілері жүйелік талаптарда көрсетілуі тиіс. Іс жүзінде және ең алдымен, қосымшаларды жеделдетіп әзірлеу жағдайында, бұл жиі сақталмайды.

Осылайша, даярлаушылар қауіпсіздікті қамсыздандыру саясатына сүйенулері керек, себебі ол әзірлеу және оларды бағалау бойынша шешімдерді қабылдау үшін негіздерді ұсынады.

Елестетіп көріңіз, сіз аурухана емделушілерінің деректер базасына кіруге басқару жүйесін әзірлеудесіз. Ауруханаңың қауіпсіздік саясаты қызметкерлер құрамының тек уәкілетті мүшелері ғана емделушілердің электрондық медициналық карталарын өзгертуге құқылы деп айтуы мүмкін. Осылайша, Сіздің жүйеңізде осы жүйедегі деректерді өзгертуге тырысқан әрбір тұлғаның өкілеттігін тексеретін және тиісті өкілеттігі жоқ тұлғалармен енгізілген өзгертулерді қабылдамайтын механизмдері болуы қажет.

Сізге кездесуі ықтимал мәселенің мәні мынада, көптеген ұйымдарда қауіпсіздікті қамсыздандырудың айқын саясаты жоқ. Уақыт өте келе жүйеге анықталған мәселелерге қарсы әрекет ретінде саналатын өзгерістер енгізілуі мүмкін, алайда олардың негізінде жүйе эволюциясының бағытын шарттасатын, барлығын тегіс қамтитын саясат жатпайды. Бұндай жағдайларда, Сіз саясатты тәлімдер негізінде әзірлеп және құжатты түрде ресімдеуіңіз керек және компания басшыларының бекітуіне шығару қажет.

2-нұсқау: Істен шығуы барлық жүйенің бас тартуына әкелетін құрамдастардан аулақ болыңыз

Сыни жүйелерді жобалаудың жалпыға ортақ нормаларында істен шығуы барлық жүйенің бас тартуына әкелетін құрамдастардан аулақ болу керек деп айтылған. Қауіпсіздік терминдерімен айтсақ, бұл дегеніміз, Сіз қауіпсіздікті қамсыздандыратын жалғыз бір механизмге ғана толық сенім артпауыңыз керек дегенді білдіреді. Керісінше, Сізге түрлі әдістерді қолданған жөн. Бұны кейде «тереңдіктегі қорғаныс» деп атайды.

Мысалы, егер Сіз жүйеде пайдаланушының кіру құқығын растау үшін құпиясөзді пайдаланатын болсаңыз, онда Сізге бұнымен бірге «сұрақ-жауап» түріндегі түпнұсқалықты растау механизмін де енгізу қажет, бұған сәйкес, пайдаланушылар бірқатар сұрақтар мен оларға жауаптарды алдын ала жүйеде тіркеулері керек. Дұрыс құпиясөзді енгізген соң пайдаланушы рұқсат алу үшін осы сұрақтарға да дұрыс жауап беруі тиіс. Жүйе деректерінің бүтіндігін қорғау үшін Сіз, сонымен бірге, деректердің барлық өзгерісінің атқарылу журналын да жүргізе аласыз (5-кепілдікті қара). Жарамсыздық жағдайы пайда болғанда деректер жинағын қалпына келтіру үшін Сіз журналға бет бұра аласыз. Сондай-ақ Сіз белгілі бір өзгертулер енгізбес бұрын, өзгертілетін барлық деректерді көшіріп жаза аласыз.

3-нұсқау: Бас тарту қауіпсіздігін қамсыздандырыңыз

Бас тарту барлық жүйелер үшін бұлтартпас құбылыс болып табылады, алайда, қауіпсіздік сыни параметрі болып табылатын жүйелер «бас тартудан қауіпсіз» болуы тиіс. Жүйенің бас тартуы жағдайында Сіз, жүйенің өзіне қарағанда қауіпсіздігі шамалы болатын апаттық режімге көшу үдерісін қолданбауыңыз керек. Бұдан тыс, жүйенің бас тартуы әдетте қорғалған деректерге бұзушы кіре алады дегенді білдірмеу керек.

Мысалы, емделушілер туралы ақпарат жүйесі үшін біз мынадай талаптарды ұсынған едік, оған сәйкес емделушілер туралы мәліметтер клиенттің жүйесіне сессия басында жүктелуі керек. Бұл кіруді жылдамдатады және серверге қол жетпейтін кезде де деректерге кіруге болатынын көрсетеді. Әдетте сервер бұл деректерді сессияның соңында жояды. Алайда, егер сервердің бас тартуы орын алса, деректердің клиент жүйесінде қалу мүмкіндігі бар. Бұл шарттардағы қауіпсіздіктен бас тарту көзқарасының мәнісі клиент жүйесінде сақталған

емделушілер туралы барлық мәліметтерді кодтауда болады. Бұл тіркелмеген пайдаланушы осы мәліметтерді оқи алмайды дегенді білдіреді.

4-нұсқау: Қауіпсіздік пен жинақтылықты теңдестіріңіз

Қауіпсіздік пен жинақтылық талаптары көбіне ішінара қарама-қайшы болады. Жүйенің қауіпсіздігін қамтамасыз ету үшін Сізге қауіпсіздікті қамсыздандыру саясатына сәйкес пайдаланушы жүйесінің әрекет құқығын тексеруді енгізу керек. Бұның бәрі пайдаланушыларға еріксізден бірқатар талаптарды жүктейді – логиндер мен құпиясөздерді есте сақтау, жүйені белгілі бір компьютерде ғана пайдалану және т.б. Бұл пайдаланушының жүйені іске қосу мен тиімді пайдалануға біршама уақыты кетеді дегенді білдіреді. Жүйеге қауіпсіздікті қамсыздандырудың көптеген құралдарын қосқан кезде оның пайдалылығы еріксіз төмендейді. Қауіпсіздік пен жинақтылықтың жалпы сұрақтарының бірқатары ашылған Кренор мен Гарфункельдің кітабын (2005) оқуға кепілдеме береміз.

Қауіпсіздікті қамсыздандырудың жаңа құралдарын қосу жөнсіз болатын және жинақтылықтың төмендеуіне әкелетін белгілі бір кезең туындайды. Мысалы, егер Сіз пайдаланушылардан белгілі бір реттілікте көптеген құпиясөздерді енгізуді немесе олардың құпиясөздерін есте сақталуы қиын символдар комбинациясына ауыстыруын талап ететін болсаңыз, олар өз құпиясөздерін әшейін жазып ала бастайды. Бұзушы (бірінші кезекте штаттағы қызметкер) құпиясөзді тауып алады және жүйеге кіруге мүмкіндік алады.

5-нұсқау: Пайдаланушының әрекеттерін тіркеңіз

Егер іс жүзінде орындау мүмкін болса, онда Сіз әрқашан пайдаланушы әрекетінің журналын жүргізуіңіз керек. Бұл журналда түрлі операцияларды кімнің жүзеге асырғаны, жүйенің пайдаланылғаны активтері туралы ақпарат, сонымен қатар іс-әрекеттің уақыты мен күні жазылуы керек. 2-нұсқауда айтылғандай, егер Сіз осы ақпаратты атқарылатын командалар түрінде сақтаған болсаңыз, жарамсыздық жағдайынан кейін ақпараттарды қалпына келтіру мақсатымен журналға қарау мүмкіндігіңіз бар. Әлбетте, сонымен бірге Сізге журналдың мазмұнын талдауға және әлеуетті аномальды әркеттерді анықтауға көмектесетін аспаптар қажет болады. Бұл аспаптар журналды саралап және аномальды әрекеттерді табуы тиіс, және осылайша, шабуылды айқындауға көмектесіп және бұзушының жүйеге кіру рұқсатын қалай алғанын зерттеп тексеруі керек.

Пайдаланушының әрекет журналының жарамсыздықтан кейін қалпына келтіруде көмектен басқа да пайдасы бар, ол ішкі шабуылдардың алдын алу құралы ретінде де пайдалы. Егер адамдар өз әрекеттерінің журналға жазылатынын білсе, олардың рұқсат берілмеген операцияларды жасау ықтималы шұғыл төмендейді. Бұл қарапайым қиянатшылық (мысалы, медбикелердің бөтен емделушілердің медициналық карталарын қарауы) кезінде немесе психологиялық шабуыл көмегімен авторластырылған пайдаланушылардың тіркелу мәліметтерін ұрлау жолымен іске асатын шабуылдарды анықтауда аса тиімді. Әрине, аталған

әдіс толығымен қауіпсіз бола алмайды, өйткені шебер бұзушылар журналға да кіру рұқсатын алып, ондағы деректерді өзгерте алады.

6-нұсқау: Қауіпті төмендету үшін көпсандылық пен әртүрлілікті пайдаланыңыз

Көпсандылық жүйенің бағдарламалық қамсыздандырудың немесе деректер жиынтығының біреуден артық версиясымен жабдықталғанын білдіреді. Бағдарламалық қамсыздандыруға қатысты әртүрлілік түрлі версиялардың әртүрлі платформаларда негізделуі немесе әр түрлі технологиялар көмегімен жүзеге асырылуы керектігін білдіреді. Осылайша, платформаның немесе технологияның кемшіліктері БҚ-ның барлық версиясына кері әсерін тигізе алмайды және жалпы жарамсыздыққа әкеліп соқпайды. 13-тарауда көпсандылық пен әртүрлілік неліктен функционалды сенімді жүйелерді жобалаудың іргелі механизмдері болып табылатыны туралы талқыланған болатын. Біз алдында көпсандылықтың мысалдарын айтып кеттік – емделушілер туралы ақпаратты серверде және клиенттік базада сақтау, алдымен психикалық денсаулық сақтау жүйесі үлгісінде, соңынан 14.5-суретте көрсетілген банктік сауда жүйесі үлгісінде. Емделушілер туралы деректер базасының жүйесі жағдайында Сіз сервер және клиент компьютері үшін түрлі операциялық жүйелерді қолдана аласыз (мысалы, серверде Linuxe, клиент жүйесінде Windows). Бұл операциялық жүйенің кемшіліктері негізінде жасалған шабуылдың сервердегі де және клиент компьютеріндегі де деректердің беделін түсіре алмайтынына кепілдік береді. Әрине, бұл үстемдік үшін ұйымда пайдаланылатын түрлі операциялық жүйелерге қызмет көрсетуді көтеріңкі бағамен өтеуге тура келеді.

7-нұсқау: Барлық ендірмелердің дұрыстығын тексеріңіз

Жүйеге жасалатын қарапайым шабуыл жүйе күтпеген енгізу сандарына тап болғанда, жүйе тәртібінің күтпеген үлгіге әкелуін қарастырады. Бұл жүйенің толығымен бас тартуын туғызуы мүмкін, ол жағдайда қажетті қызметтер көрсетілмейтін болады. Сонымен қатар, енгізулерде зиянды кодтар болуы ықтимал, ол жүйемен атқарылады. Бұзушылар кеңінен пайдаланатын (Бергель, 2001) және Интернет құрттардың әрекеті негізделген буфердің толу қателері (Спаффорд, 1989) ұзын енгізулермен активтендірілуі мүмкін. Бұзушы құрылымданған сұрау салу тілінде фрагмент енгізген кезде сервер оны өзінше түсінетін, «Құрылымданған сұрау салуды жасырын көшіру» деп аталатын бұзушылық бұзудың әйгілі тәсілдерінің бірі.

13-тарауда айтылғандай, мұндай мәселелердің бәрінен жүйені енгізулерді бағдарламалық тексерумен жабдықтау жолымен құтылуға болады. Көп жағдайда, Сіз енгізіліп жатқан ақпаратты тиісінше тексермей ешқашан қабылдамауыңыз керек. Енгізулерді тексеруге қолданылатын тәсілдерді белгілеу талаптардың бір бөлігі болып табылады. Тексерудің қажетті тәсілдерін белгілеу үшін Сізде өзіндік енгізулер туралы мағлұмат болу керек. Мысалы, егер адамның тегін енгізу керек болса, Сіз енгізуде бос орынның болмауын тексере аласыз, соны-

мен бірге қолданылған жалғыз тыныс белгісі – дефис екеніне көз жеткізесіз. Сіз сондай-ақ енгізудің символдар санын тексере аласыз және шектен тыс ұзын болған енгізулерді қабылдамайсыз. Мысалы, адамның тегі 40 символдан артық құралмайды, ал мекен-жайы – 100 символдан артық болмайды. Енгізуді шектеуді көрсететін мәзірді пайдалану Сізге енгізуді тексерудегі бірқатар проблемалардан аулақ болуға көмектеседі.

8-нұсқау: Активтерді блоктар бойынша бөліңіз

Блоктар бойынша бөлу жүйе ақпаратына кірудің «барлығы немесе ештеңе» деген сызбасы болмауы керек дегенді білдіреді. Керісінше, Сізге жүйе ақпаратын блоктар бойынша ұйымдастыру қажет. Пайдаланушылардың жүйенің барлық ақпаратына емес, тек оның жұмысына қажетті ақпаратқа ғана рұқсаты болуы тиіс. Бұл жемісті шабуылдың салдарларын шектеуге мүмкіндік береді. Кейбір ақпарат жоғалуы немесе зақымдалуы мүмкін, бірақ шабуылдың жүйенің барлық ақпаратына әсерін тигізу ықтималы өте аз болады.

Емделушілер туралы ақпараттар жүйесі жағдайында, Сізге мысалы, мынадай жүйе даярлаған дұрыс: оған сәйкес бір клиниканың қызметкерлер құрамы тек осы клиникаға келетін емделушілер туралы ғана ақпаратқа кіру рұқсатын ала алады. Әдетте оларда жүйенің барлық медициналық карталарына кіру құқығы болмауы тиіс. Бұл ішкі шабуылдардан туындайтын әлеуетті шығындарды әжептәуір шектейді.

Жоғарыда айтылғандармен қоса, Сіздің әзірлеген жүйеңіз кездейсоқ ақпараттарға – мысалы, дәрігерге бару тағайындалмаған, бірақ жедел емдеуге зәру ауыр сырқатты пациент туралы ақпаратқа – кіру рұқсатын беретін механизмдермен жабдықталуы тиіс. Бұндай жағдайларда Сіз ақпараттарды блоктар бойынша бөлуді айналып өтуге мүмкіндік беретін басқалай қорғаныс механизмін пайдалана аласыз. Жүйенің қолжетімдігін көтеру мақсатында қауіпсіздік босаңтылған осы тәрізді жағдайларда, пайдалану жөніндегі операцияларды журналға жазып кіргізу механизмін пайдалану аса маңызды. Мұндай сәттерде, Сіз әрқашан журналдың көмегімен рұқсат етілмеген қолданыстар жағдайларын зерттеп тексере аласыз.

9-нұсқау: Пайдалануға енгізу қарапайымдылығын қамтамасыз ете отырып әзірлеңіз.

Қауіпсіздіктің көпшілік проблемалары жүйені пайдалануға беру кезіндегі дұрыс емес конфигурация нәтижесінде туындайды. Сол себепті, жүйені әзірлеу барысында оның сатып алушылар ортасына енуін жеңілдететін және пайдалануға енгізілетін жүйе конфигурациясының кемшіліктері мен әлеуетті қателерін зерттеп тексеретін құралдармен жабдықталғаны дұрыс. Бұл тақырып 14.2.3-тарауында толығырақ ашылған.

10-нұсқау: Қалпына келтірулікті қамтамасыз ете отырып әзірлеңіз

Жүйе қауіпсіздігін қамтамасыз етуге кеткен күш-жігерге қарамастан, Сіз әрқашан жүйе жарамсыздығының тууын еске ала отырып, жүйелерді әзірлеуіңіз керек. Осылайша, Сізге әр заман жүйені әлеуетті жарамсыздықтан кейін қайта қалпына келтіру және оны жұмыс жағдайына келтіру туралы ойлау қажет. Мысалы, Сіз құпиясөзді енгізу жүйесі беделін түсірген жағдайда, түпнұсқалықты растаудың қосымша жүйесін енгізе аласыз.

Былай ойлап қараңыз, бөтен бір тұлға емделушілердің медициналық карталар жүйесіне кіру рұқсатына ие болады, алайда Сізге ол тұлғаның логин мен құпиясөздің дұрыс комбинациясын қалай алғаны туралы мәлім. Бұл жағдайда Сізге бұзушы пайдаланған тіркелу деректерін өзгертіп қана қоймай, түпнұсқалықты растау жүйесін қайтадан бастамалау керек. Бұл өте маңызды, өйткені бұзушы, бәлкім басқа да пайдаланушылардың құпиясөздеріне кіру рұқсатын алды. Сол себепті, Сізге барлық тіркелген пайдаланушылардың өз құпиясөздерін ауыстырғаны жөн. Сонымен қатар, Сізге тіркелмеген пайдаланушының құпиясөзді ауыстыру механизміне кіру рұқсатын алмағанына көз жеткізу қажетті. Осы себептен, Сізге құпиясөздер ауыстырылмайынша, барлық пайдаланушыларға кіру рұқсатын бермейтін, ал пайдаланушылардың жаңа құпиясөзді алу құқы олардың таңдап алған құпиясөзінің қауіпсіз емес екендігіне жол бере отырып тексерілетін жүйені даярлау керек.

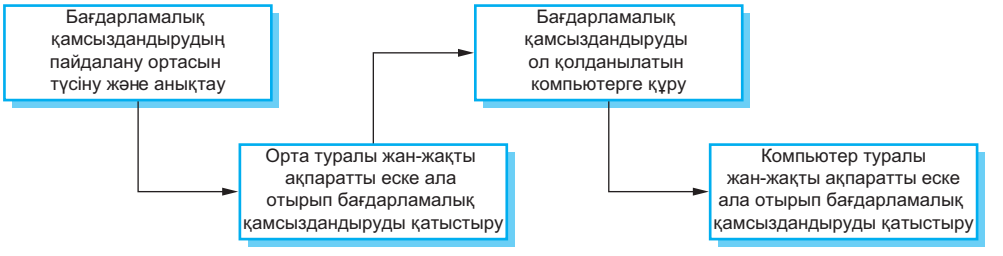
Аталмыш талапқа қол жеткізудің бір тәсілі болып «сұрақ/жауап» механизмін пайдалану табылады, яғни пайдаланушылар өздері алдын ала тіркеген жауабы бар сұраққа жауап беруі керек. Бұл механизм құпиясөзді ауыстыру жағдайында ғана активтендірілу керек, ол өз кезегінде жүйені шабуылдан қорғай отырып, пайдаланушылардың жұмыстан мойын бұруының аз деңгейін тудырады.

14.2.3 Пайдалануға берудің қарапайымдылығын қамтамасыз ететін әзірлеме

Жүйені пайдалануға беру белгілі бір ортада оның тиімді қызмет етуі мақсатымен бағдарламалық қамсыздандырудың конфигурациясын, осы орта компьютеріне жүйені құруды, ал содан кейін осы компьютерлердің ерекшеліктерін еске ала отырып, жүйені конфигурациялауды білдіреді (14.7-сурет). Конфигурация пайдаланушының ықыласын көрсету мақсатымен бағдарламалық қамсыздандырудың кейбір алдын ала белгіленген параметрлерін икемдеудің қарапайым процесі бола алады. Бірақ кейде конфигурация күрделі де болады және бағдарламалық қамсыздандыру операцияларының орындалуына әсерін тигізетін ережелер мен іскер үлгілердің нақты айқындалуын талап етеді.

Бағдарламалық қамсыздандыру конфигурациясының бұл кезеңінде көбінесе бағдарлама жұмысына қателікпен бірнеше кемшіліктер енгізіледі. Мысалы, құру барысында бағдарламалық қамсыздандыруға көбінесе тіркелген пайдаланушылардың тізімі енгізіледі. Бұл тізімде әдетте әкімшінің сипатты логині

(мысалы «admin) мен үндемеу бойынша құпиясөзі (мысалы, «password») болады. Бұл әкімшіге жүйені икемдеу тапсырмасын жеңілдетеді. Пайдаланушының сипатты атын жою мен жаңа логин мен құпиясөзді құру олардың алғашқы әрекеті болуы тиіс. Алайда бұл туралы ұмытып кету өте оңай. Логин мен үндемеу бойынша құпиясөзді білетін бұзушы жүйеге кіруге айрықша құқықты ала алады.



14.7-сурет. Бағдарламалық қамсыздандыруды пайдалануға енгізу

Конфигурация мен пайдалануға беру көбінесе жүйелік әкімшінің аспектісі ретінде қарастырылады, сондықтан да бағдарламалық қамсыздандыруды жобалау және әзірлеу үдерістерінен тыс шығарылады. Әрине, басқарудың тиісті практиктері конфигурация мен пайдалануға беру қателіктерінің нәтижесінде туындайтын қауіпсіздік мәселелерінің көпшілігін жоюға қабілетті. Бірақ бағдарламалық қамсыздандыруды әзірлеушілер «пайдалануға берудің қарапайымдылығын қамтамасыз ететін әзірлеме» үшін жауапкершілік алады. Сізге әрқашанда жүйелік әкімшілердің (немесе пайдаланушылардың) бағдарламалық қамсыздандыру конфигурациясы кезінде қателік жіберу ықтималдығын төмендететін, пайдалануға берудің кіріктірме қолдауын ұсыну қажет. Пайдалануға беруді қолдау жүйесіне енгізудің төрт тәсіліне кепілдік береміз:

1. *Қарап шығу және конфигурацияны талдау құралдарын қолдау.* Әкімшілер мен белгілі бір пайдаланушыларға жүйенің ағымдағы конфигурациясын тексеруге мүмкіндік беретін құралдарды жүйеге әрқашан қосып отыру қажет. Ең қызығы, бұндай құралдар көптеген бағдарламалық жүйелерде жоқ, ал пайдаланушылар конфигурацияны дұрыс икемдеуді іздестірумен туындаған қиындықтардан ашуға басады. Мысалы, тарауды жазу үшін қолданылған тексттік редактордың осы версиясында жүйенің барлық икемін жеке экранда қарап шығу немесе оларды басып шығару мүмкін емес. Алайда, егер әкімші конфигурацияның толық бейнесін ала алса, онда оның барлық қателіктер мен кемшіліктерді табу мүмкіндігі көп есе артады. Дұрысында, конфигурацияның бейнелену дисплейі әлеуетті қауіпті болып табылатын икемдерге де назар аудартуы керек, мысалы, құпиясөз құрылмаған жағдайда.
2. *Үндемеу бойынша берілетін артықшылықтар санын азайтыңыз.* Бағдарламалық қамсыздандыруды әзірлегенде үндемеу бойынша қолданылатын жүйенің конфигурациясын өте аз артықшылық беретін етіп

әзірлеу керек. Осылайша, кез келген шабуыл тигізген зиян шектеулі бола алады. Мысалы, әкімшіні тіркеудің үндемеу бойынша қолданылатын жүйесі бағдарламаға кіру рұқсатын жалғыз қолжетімді функцияға дейін шектеуі керек: әкімшінің жаңа тіркеу мәліметтерін құру. Ол басқа кез келген жүйелік құралдарға кіруге рұқсат бермеуі тиіс. Жаңа тіркеу мәліметтерін құрғаннан кейін үндемеу бойынша қолданылған логин мен құпиясөз автоматты түрде жойылуы керек.

3. *Конфигурацияны икемдеуді оқшаулandыру.* Жүйенің конфигурациясын қолдауды әзірлеу кезінде Сіз жүйенің бір-ақ бөліктеріне әсер ететін конфигурация параметрлерінің бір-ақ жерлерде құрылуына көз жеткізуіңіз қажет. Біз қолданып отырған тексттік редактордың версиясын үлгі ретінде қайтадан пайдалану үшін, біз ақпарат қауіпсіздігінің белгіленген икемдерін жүзеге асыра аламыз: құжатқа кіруді мәзір көмегімен құпиясөзбен қорғау -Ықылас/Қауіпсіздік. Қалған параметрлер мына мәзір көмегімен икемделеді Құралдар/Қорғау. Егер конфигурация туралы ақпарат оқшауландырылмаған болса, оны икемдеуді ұмыту өте оңай. Бұдан басқа, кейбір жағдайларда, пайдаланушылар жүйенің белгілі бір қорғаныс құралдары бар екенін тіпті білмейді.
4. *Қауіпсіздік кемшіліктерін жоюдың жеңіл тәсілдерін ұсыну.* Сіз анықталған қауіпсіздік кемшіліктерін жою мақсатымен жүйені жаңартудың түсінікті механизмдерін енгізуіңіз керек. Олар қауіпсіздіксіз жаңартуларды автоматты түрде тексерумен немесе бұл жаңартуларды пайда болған сәттен жүктеумен ұсынып ала алады. Пайдаланушылардың осы механизмдерді айналып өте алмаулары өте маңызды, өйткені, олар сөзсіз басқа жұмысты маңызды және шұғыл деп есептейді. Пайдаланушылардың жүйе талап еткен уақытта бағдарламалық қамсыздандыруды жаңартпауы нәтижесінде туындаған (мысалы, аурухана желісінің жаппай жарамсыздануы) ірі қауіпсіздік мәселелерінің бірнеше тіркелген үлгілері бар.

14.3 Жүйелердің өміршеңдігі

Дәл осы кезге дейін біз дайындалу үстіндегі қосымша тұрғысынан қауіпсіздікті қамтамасыз ету мәселесін талқылап келдік. Жүйені сатып алушылар мен әзірлеушілер жүйенің шабуыл жасалуы мүмкін барлық аспектілерін бақылап отырады. 14.1-суретте көрсетілгендей, тәжірибеде бөлінген заманауи жүйелер еркін сатылуда қолжетімді жүйелерді және түрлі ұйымдар әзірлеген көп рет пайдаланылатын компоненттерді қамтитын инфрақұрылымдарға сөзсіз негізделеді. Бұл жүйелердің қауіпсіздігі әзірлеу жөніндегі жергілікті шешімдерге ғана байланысты емес. Оған, сондай-ақ сыртқы қосымшалар, веб-қызметтер мен желілік инфрақұрылымдар әсер етеді.

Бұл қауіпсіздік проблемасына қаншалықты назар аударылатындығына қарамастан жүйенің кез келген сыртқы шабуылға төтеп бере алатындығына кепілдік берілмейді дегенді білдіреді. Сәйкесінше, күрделі желілік жүйелер жағдайында

сізге жүйеге кіруді мүмкін, ал жүйенің тұтастығына кепілдік берілмейтіндей жасау керек. Сонымен, сізге кез келген жағдайда жүйе пайдаланушыларға негізгі қызметтерді көрсете алатындай оны қалай өміршең жасау керектігі туралы ойлану қажет.

Өміршеңдік немесе сыртқы әсерлерге тұрақтылық (Уэстмарк, 2004) – бұл өз бетінше өміршеңдікке ие болмайтын жүйенің жекелеген компоненттерінің қасиеттері емес, жүйенің тұтас тәуелсіз қасиеті. Жүйенің өміршеңдігі оның шабуыл жағдайында немесе жүйенің бір бөлігі зақымданғанан кейін заңды пайдаланушыларына негізгі іскерлік немесе күрделі қызметтер көрсете алу қабілеті болып табылады. Жүйенің зақымдануына жүйенің жаңылысуы немесе ондағы шабуыл себеп болады.

Жүйенің өміршеңдігін зерттеуге біздің өміріміздің экономикалық және әлеуметтік саласының компьютерлік жүйелер бақылайтын сыни инфрақұрылымдарға байланыстылығы факті себепші болды. Бұл өзіне коммуналдық қызметтер көрсету инфрақұрылымын (электр, су, газ және т.б.), сондай-ақ едәуір маңызды болып табылатын ақпараттар беру және басқару инфрақұрылымын (телефон, Интернет, почта қызметі және т.б.) қамтиды. Алайда, өміршеңдік – бұл инфрақұрылымның жай ғана күрделі аспектісі емес. Жұмысы күрделі желілік компьютерлік жүйелерге негізделген кез келген ұйым қасақана шабуыл жасаудың немесе жүйенің апатты жаңылысуының нәтижесінде жүйе қызметінің тоқтауының олардың бизнесіне қалай әсер ететіндігіне қызығушылық танытуы тиіс. Осылайша, іскерлік күрделі жүйелер жағдайында өміршеңдікті талдау және қамтамасыз ету қауіпсіздікті қамтамасыз етудің бір бөлігі болуы тиіс.

Өміршеңдіктің мәні кез келген жағдайда күрделі қызметтерді көре алу қабілетін сақтауда. Бұл сіздің келесілерді білуіңіз тиіс дегенді білдіреді:

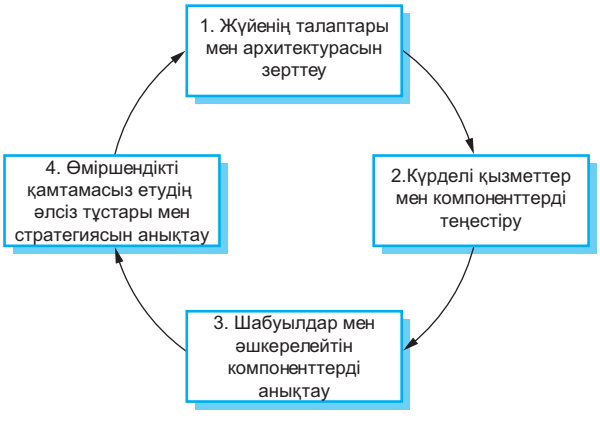
- жүйенің бизнес үшін сыни рөл атқаратын қызметтерін;
- орын алған жағдайға қарамастан көрсетілуі тиіс қызметтердің ең аз көлемін;
- бұл қызметтердің қалай әшкереленуі мүмкін екендігін;
- бұл қызметтерді қалай қорғауға болатындығын;
- егер жүйенің қызметтері қолжетімсіз болып қалған жағдайда оларды қалай тез арада қалпына келтіруге болатындығын.

Мысалы, қоңыраулар қабылдаумен және шапшаң жәрдем етуді қажет ететін пациенттерге жедел жәрдем машинасын жіберумен байланысты қызметтер жүйенің сәйкес шақыруларға жауап ретінде жедел жәрдем бригадасын жіберетін қиын қызметтері болып табылады. Өзге қызметтер (қоңырауларды журналға тіркеу, жедел жәрдем машинасының тұрған жерін анықтау) айтарлықтай қиын емес, себебі олар нақты уақытта өңдеуді қажет етпейді немесе олар баламалы механизмдерді қолдануға жол береді. Мысалы, жедел жәрдем машинасының тұрған жерін анықтау үшін сіз оның бригадасының мүшесіне қоңырау шалып, олардың қайда тұрғандығын сұрай аласыз.

Эллисон және оның ұжымдастары (1999a; 1999b; 2002) Жүйенің өміршеңдік деңгейін талдау деп аталатын әдісті әзірледі. Ол жүйенің кемшіліктерін бағалау

үшін және жүйенің өміршеңдігін қамтамасыз ететін құралдар мен архитектураларды әзірлеуге қолдау көрсету үшін пайдаланылады. Олар өміршеңдікке қол жеткізу бірін-бірі толықтыратын үш стратегиямен байланысты деп сендіреді:

1. *Қарсыласу.* Жүйеге шабуылдарға қарсыласу әдістерін енгізу арқылы проблемалардан құтылу. Мысалы, жүйе пайдаланушылардың кіру құқығын анықтау үшін сандық куәгерлерді пайдалануы мүмкін, ал бұл тіркелмеген пайдаланушылардың кіруге рұқсат алуын қиындатып жібереді.
2. *Тану.* Жүйені шабуылдар мен жаңылысуларды анықтау әдістерімен, сондай-ақ зақымдану нәтижесінде пайда болатын бағалау әдістерімен жабдықтау арқылы проблемаларды айқындау. Мысалы, бақылау сомалары сыни деректермен байланысты болуы мүмкін, сондықтан бұл деректердің зақымдалуы анықталуы ықтимал.
3. *Қалпына келтіру.* Проблемалар жүйеге шабуыл жағдайында қызметтер көрсету әдістерін, сондай-ақ шабуылдан кейін толыққанды қызмет көрсетуді қалпын келтіру әдістерін енгізу арқылы жүйемен тасымалданады. Мысалы, сол бір функцияны жүзеге асырудың түрлі әдістеріне негізделген тоқтап қалуға қарсы тұра алушылық механизмдері жүйенің бір бөлігінде қызметтер көрсету мүмкіндіктерін жоғалтып алумен күресу үшін жүйеге енгізілуі мүмкін.



14.8-сурет. Өміршеңдікті талдау кезеңдері

Жүйенің өміршеңдігін талдау олардың барысында пайдаланылатын немесе ұсынылып отырған жүйенің талаптары мен архитектурасы талданатын; қиын қызметтер, шабуылдардың ықтимал сценарийлері және жүйенің әлсіз жерлері теңдесетін; жүйенің өміршеңдігін арттыра алатын өзгерістер ұсынылатын 4 кезеңнен (14.8-сурет) тұратын процесс болып табылады. Ары қарай әрбір кезеңнің негізгі операциялары ұсынылған:

1. *Жүйені түсіну.* Пайдаланылатын немесе ұсынылып отырған жүйенің мақсаттарын (кейде олар қызмет атқарудың негізгі міндеттері деп аталады), сондай-ақ жүйенің талаптары мен архитектурасын зерттеңіздер.
2. *Қиын қызметтерді сәйкестендіру.* Көрсетілуі тоқтатылмауы тиіс қызметтер, сондай-ақ орын алған жағдайға қарамастан өзінің жұмысын жалғастыра беруі тиіс компоненттер анықталады.
3. *Шабуылдың сылтауратуы көлгірлігі.* Ықтимал шабуылдар сценарийі, сондай-ақ осы шабуылдар арқылы әшкереленетін компоненттері анықталады.
4. *Өміршеңдікті талдау.* Бір уақытта күрделі және әшкереленетін шабуыл болып табылатын компоненттер, сондай-ақ қарсыласуға, тануға және қалпына келтіруге негізделген өміршеңдікті арттыру стратегиясы анықталады.

Эллисон және оның ұжымдастары психикалық бұзылу клиникасын қолдау жүйесі мысалында (1999b) тәжірибеде қолданылған керемет әдісті ұсынды. Бұл жүйе осы кітапта мысал ретінде үнемі пайдаланатын емделушілер туралы деректер қорының жүйесімен ұқсас. Олардың талдауын қайталамас үшін біз өміршеңдікті талдаудың кейбір ерекшеліктерін көрсету мақсатында 14.5-суретте көрсетілген акциялар саудасы жүйесін қолдануды шештік.

14.5-суретте жүйенің өміршеңдіктің белгілі бір стратегияларын қарастыратындығы көрінеді. Пайдаланушылар шоттарының көшірмелері мен акцияның бағалары көшіріп алынып, серверлердің арасында жергілікті серверге кіру мүмкін болмаған жағдайдың өзінде тапсырыстарды орналастыруға болатындай етіп бөлінген. Бағалы қағаздарға тапсырыстар орналастыру жұмысқа қабілеттілігіне кез келген жағдайда қолдау көрсетілуі тиіс негізгі қызмет болып табылады деп болжап көрейік. Пайдаланушылар жүйеге сену үшін оның тұтастығын қамтамасыз ету қажет. Тапсырыстар ұқыпты, тиянақты болуы тиіс және жүйені пайдаланушылардың сатып алу немесе сату жөніндегі нақты операцияларын көрсетуі керек.

Тапсырыстарды орналастыру қызметінің жұмысын қолдау үшін жүйенің үш компоненті пайдаланылады:

1. *Пайдаланушының шын екендігін тексеру.* Тіркелген пайдаланушыларға жүйеге кіруге мүмкіндік береді.
2. *Баға белгілеу.* Акцияларды сатып алу немесе сату бағаларын тағайындауға мүмкіндік береді.
3. *Тапсырысты орналастыру.* Белгіленген бағамен сатып алуға немесе сатуға тапсырыстар жасауға мүмкіндік береді.

Аталмыш компоненттердің негізгі ақпараттық активтерді: пайдаланушы шотының деректер қорын, бағалардың деректер қорын және транзакциялардың деректер қорын пайдаланатындығы анық. Шабуыл болған жағдайда аталмыш активтердің жұмысына қолдау көрсетілуі тиіс.

Шабуыл	Қарсыласу	Тану	Қалпына келтіру
Жүйеге рұқсатсыз кірген пайдаланушы зиянды тапсырыстарды орналастырады	Тапсырысты орналастыру паролінен ерекшеленетін мәміле жасау паролі қажет болады	Тіркелген пайдаланушыға орналастырылған тапсырыстың көшірмесін байланыс телефонын көрсете отырып (оның зиянды тапсырысты анықтай алуы үшін) электронды почтамен жіберу Пайдаланушының тапсырыстар тарихын жүргізу және стандартты емес мінез-құлық үлгілерін анықтау	Сатып алу немесе сатуды автоматты түрде «жою» және пайдаланушы шотының деректерін қалпына келтіру механизмінің болуы. Зиянды іс-әрекеттердің нәтижесінде туындаған пайдаланушының шығынын өтеу. Жанама зияндардан сақтандыру.
Транзакцияның деректер қоры жұмысының бұзылуы	Жеке тұлғасын растайтын айтарлықтай сенімді механизмнің көмегімен артықшылықты пайдалану-шыларды тіркеу талап етіледі (мысалы, сандық куәгерлердің көмегімен)	Халықаралық сервердегі оқу үшін ғана қолжетімді транзакциялар көшірмесін сақтау. Зиянды іс-әрекеттерді анықтау мақсатында транзакцияларды оқтын-оқтын тексеру. Зиянды іс-әрекеттерді анықтау мақсатында транзакциялар туралы барлық деректер бар криптографиялық бақылау қағазын сақтау.	Резервтегі көшірмелердің көмегімен деректер қорын қалпына келтіру. Транзакцияның деректер қорын қалпына келтіру мақсатында уақыттың белгілі бір кезеңінен бастап жүзеге асырылған барлық сауда операцияларын қалпына келтіру механизмінің болуы.

14.9-сурет. Акциялар саудасы жүйесі мысалында өміршеңдікті талдау

Аталмыш жүйеде шабуылдың бірнеше түрі жүзеге асырылуы мүмкін. Ары қарай екі нұсқаны қарастырамыз:

1. Арам ниетті пайдаланушы жүйенің тіркелген пайдаланушысына шабуыл жасады. Арам ниетті пайдаланушы олардың тіркелген мәліметтерінің көмегімен жүйеге кіруге мүмкіндік алды. Тіркелген пайдаланушыға про-

блема тудыру мақсатында оның негізінде бағалы қағаздар сатып алынған және сатылған зиянды тапсырыстар орналастырылды.

2. Қуатталмаған пайдаланушы SQL командасын тікелей жүзеге асыруға рұқсат алу арқылы транзакциялардың деректер қорының жұмысын бұзады, ал бұл сатып алу және сату бойынша келісілген операцияларды мүмкін еместей етеді.

14.9-суретте осы шабуылдармен күрес жүргізу үшін пайдаланылуы мүмкін қарсыласу, тану және қалпына келтіру мысалдары берілген.

Әрине, жүйенің өміршеңдігін немесе сыртқы әсерлерге тұрақтылығын арттыру көп шығынды қажет етеді. Егер компаниялар бұрындары қауіпті шабуылдарға ұшырамаған және сәйкесінше шығын шекпеген болса, олар жүйелердің өміршеңдігіне лажысыздан қаражат салуы мүмкін. Алайда, сіздің үйіңізді тонап кеткеннен кейін емес, тонап кеткенге дейін жақсы құлып пен дабылдар жүйесін сатып алған абзал. Осыған ұқсас жағдайда, ең жақсысы жүйенің өміршеңдігіне сәтті шабуылдан кейін емес, оған дейін қаражат салыңыз. Алайда, өміршеңдікті талдау бұрынғысынша көптеген бағдарламамен қамтамасыз етуді жобалаудың бір бөлігі болып табылмайды. Дегенмен, жүйелердің көп мөлшері бизнес үшін қаншалықты сыни болатындығына байланысты, аталмыш талдау кеңінен танымал бола бастайды.

НЕГІЗГІ ТҰЖЫРЫМДАР

- Мақсаттары компьютерлік жүйелерді немесе олардың деректерін зақымдау болып табылатын арам ниетті шабуылдарды анықтай алатын бағдарламалық жүйелерді дайындау және оларға қолдау көрсету қауіпсіздікті қамтамасыз етудің негізгі міндеті болып табылады.
- Қауіпсіздікке төнген қатерлер жүйенің құпиялылығына, тұтастығы мен қолжетімділігіне және оның деректеріне төнген қатерлер түрінде болуы мүмкін.
- Қауіпсіздікті бұзу тәуекелдерін басқару мақсаты аталмыш шығындардың алдын алу немесе азайту болып табылатын қауіпсіздік талаптарын әзірлеу және жүйелерге шабуыл нәтижесінде туындауы ықтимал шығындарды бағалауды білдіреді.
- Қауіпсіз жүйелерді жобалау жүйенің қауіпсіз архитектурасын әзірлеуді, қауіпсіз жүйелерді әзірлеудің жалпы қабылданған нормаларын сақтауды, жүйені пайдалануға енгізген кезде кемшіліктердің пайда болу ықтималдығын азайту мақсатында функционалдықты қамтамасыз етуді білдіреді.
- Негізгі активтерді қорғау үшін жүйенің құрылымын ұйымдастыру және сәтті шабуыл нәтижесінде шығынды азайту үшін жүйенің активтерін бөлу жүйенің қауіпсіз архитектурасын әзірлеудің негізгі аспектілері болып табылады.

- Қауіпсіздікті қамтамасыз ету жөніндегі ұсыныстар әзірлеушілердің назарын олар ескермей кетуі мүмкін қауіпсіздік проблемаларына аудартады. Олар қауіпсіздікті тексерудің бақылау тізімі үшін негіз жасайды.
- Пайдалануға қауіпсіз енгізуді қолдау үшін сіз конфигурацияларды бейнелеу және талдау әдістерін қамтамасыз етуге, конфигурацияның маңызды параметрлері ұмытылып кетпес үшін конфигурацияның күйге келтірулерін шектеуге, жүйені пайдаланушыларға берілетін жасырын артықшылықтардың санын азайтуға және қауіпсіздіктің кемшіліктерін жою әдістерін жасауға тиіссіз.
- Жүйенің өміршеңдігі шабуыл жағдайында немесе жүйенің сол немесе басқа бөлігі зақымданғанан кейін жүйенің заңды пайдаланушыларға негізгі немесе қиын қызметтер көрсетуді жалғастыра бере алуын көрсетеді.

ҚОСЫМША ӘДЕБИЕТТЕР

'Survivable Network System Analysis: A Case Study.' An excellent paper that introduces the notion of system survivability and uses a case study of a mental health record treatment system to illustrate the application of a survivability method. (R. J. Ellison, R. C. Linger, T. Longstaff and N. R. Mead, *IEEE Software*, **16** (4), July/August 1999.)

Building Secure Software: How to Avoid Security Problems the Right Way. A good practical book covering security from a programming perspective. (J. Viega and G. McGraw, Addison-Wesley, 2002.)

Security Engineering: A Guide to Building Dependable Distributed Systems, 2nd edition. This is a thorough and comprehensive discussion of the problems of building secure systems. The focus is on systems rather than software engineering with extensive coverage of hardware and networking, with excellent examples drawn from real system failures. (R. Anderson, John Wiley & Sons, 2008.)

ЖАТТЫҒУЛАР

14.1. Қосымшаның қауіпсіздігін қамтамасыз ету мен инфрақұрылымның қауіпсіздігін қамтамасыз ету арасындағы түбегейлі айырмашылықтың неде екенін түсіндіріңіз.

14.2. Клиника емделушілері туралы мәліметтер жүйесінің мысалын пайдала отырып, активтер, кемшіліктер, шабуылдар, қауіп-қатер және бақылау құралдарының мысалдарын атаңыз.

14.3. Неге тәуекелдерді бағалауды талаптарды анықтаудың ерте кезеңінен бастап жүйелерді пайдалану кезеңіне дейін жүзеге асыру қажет екендігін түсіндіріңіз.

14.4. №2 сұрақтың жауабын пайдалана отырып, аталмыш жүйемен байланысты тәуекелдерді бағалаңыз және осы тәуекелдерді төмендететін екі жүйелік талапты ұсыныңыз.

14.5. Бағдарламамен қамтамасыз ету саласынан алынбаған аналогтарды пайдалана отырып, активтерді қорғау үшін көп деңгейлі әдістің неге қолданылу керектігін түсіндіріңіз.

14.6. Жүйеге кіру қиын болған кезде бөлінген жүйелерді қолдау үшін түрлі технологияларды пайдаланудың неге соншалықты маңызды екенін түсіндіріңіз.

14.7. Психологиялық шабуыл деген не? Ірі ұйымдарда мұндай шабуылдардан сақтану неге қиын?

14.8. Сіз пайдаланатын сатылымда қолжетімді кез келген жүйелердің ішінен конфигурация құралдарын талдаңыз (мысалы, Microsoft Word) және сіз анықтаған барлық проблемаларды сипаттаңыз.

14.9. Қарсыласу, тану және қалпына келтірудің бірін-бірі толықтырып тұратын стратегиялары жүйенің өміршеңдік деңгейін қалай көтере алатындығын түсіндіріңіз.

14.10. Архитектурасы 14.5-суретте көрсетілген, 14.2.1-тарауда сипатталған акциялар саудасы жүйесіне жасалған шабуылдың тағы екі ықтимал сценарийін және аталмыш шабуылдармен күрес жүргізу стратегиясын ұсыныңыз.

ПАЙДАЛАНЫЛҒАН ӘДЕБИЕТТЕР

Alberts, C. and Dorofee, A. (2002). *Managing Information Security Risks: The OCTAVE Approach*. Boston: Addison-Wesley.

Alexander, I. (2003). 'Misuse Cases: Use Cases with Hostile Intent'. *IEEE Software*, **20** (1), 58–66.

Anderson, R. (2008). *Security Engineering, 2nd edition*. Chichester: John Wiley & Sons.

Berghel, H. (2001). 'The Code Red Worm'. *Comm. ACM*, **44** (12), 15–19.

Bishop, M. (2005). *Introduction to Computer Security*. Boston: Addison-Wesley.

Cranor, L. and Garfinkel, S. (2005). *Security and Usability: Designing secure systems that people can use*. Sebastopol, Calif.: O'Reilly Media Inc.

Ellison, R., Linger, R., Lipson, H., Mead, N. and Moore, A. (2002). 'Foundations of Survivable Systems Engineering'. *Crosstalk: The Journal of Defense Software Engineering*, **12**, 10–15.

Ellison, R. J., Fisher, D. A., Linger, R. C., Lipson, H. F., Longstaff, T. A. and Mead, N. R. (1999a). 'Survivability: Protecting Your Critical Systems'. *IEEE Internet Computing*, **3** (6), 55–63.

Ellison, R. J., Linger, R. C., Longstaff, T. and Mead, N. R. (1999b). 'Survivable Network System Analysis: A Case Study'. *IEEE Software*, 16 (4), 70–7.

Pfleeger, C. P. and Pfleeger, S. L. (2007). *Security in Computing, 4th edition*. Boston: Addison-Wesley.

Schneier, B. (2000). Secrets and Lies: *Digital Security in a Networked World*. New York: John Wiley & Sons.

Sindre, G. and Opdahl, A. L. (2005). 'Eliciting Security Requirements through Misuse Cases'. *Requirements Engineering*, **10** (1), 34–44.

Spafford, E. (1989). 'The Internet Worm: Crisis and Aftermath'. *Comm ACM*, **32** (6), 678–87.

Viega, J. and McGraw, G. (2002). *Building Secure Software*. Boston: Addison-Wesley.

Westmark, V. R. (2004). 'A Definition for Information System Survivability'. 37th Hawaii Int. Conf. on System Sciences, Hawaii: 903–1003.

Wheeler, D. A. (2003). *Secure Programming for Linux and UNix HOWTO*. Web published:http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/index.html.

15

Функционалдық сенімділік пен қауіпсіздікті қамтамасыз ету

Мақсаттары

Осы тараудың басты мақсаты кризистік жүйелерді әзірлеуде қолданылатын верификация және валидация техникаларын сипаттау. Осы тарауды оқығаннан кейін Сіз:

- статикалық талдауға әр түрлі тәсілдер Бағдарламалық қамтамасыз етудің кризистік жүйелерін верификациялауда қалай қолдана алатындарыңызды түсінесіз;
- сенімділігі мен қауіпсіздігін тестілеудің және кризистік жүйелерді тестілеудің маңызды мәселелері мен негізгі қағидаларын түсінесіз;
- неліктен қамтамасыз ету үдерісі, әсіресе бақылау органдарымен сертификацияланатын Бағдарламалық қамтамасыз ету жағдайында соншалықты маңызды екендігін білесіз;
- дәлелдерде жүйесіне негізделген қауіпсіздік пен функционалдық сенімділікті талдау мысалдарымен танысасыз.

Мазмұны

Қауіпсіздік пен функционалдық сенімділікті қамтамасыз ету кризистік жүйе функционалдық сенімділіктер талаптарына сай екендігін тексеру болып табылады. Бұл үшін верификация және валидация үдерістері қажетті, олардың барысында қауіпсіздікке және сенімділікке теріс ықпал тигізе алатын бағдарламалар құрылымының ақаулары анықталады.

Верификациялар мен валидация үдерістері басқа бағдарламалардың кризистік жүйелерін валидациялау үдерістеріне ұқсас болып келеді. B және B үдерістері жүйенің өз мінездемелеріне сәйкес келетінін, сонымен қатар, қызметтер пен жүйенің жұмысы клиенттің талаптарға сәйкес келуін көрсетеді. Олардың өткізілуі нәтижесінде жөнделуі қажетті құрылымдар мен бағдарламалық ақаулар анықталады. Дегенмен, кризистік жүйелерге келгенде, ерекше қатал тестілеу өткізу қажет болып отыр, мұның екі себебі бар:

1. *Ақау кезіндегі шығындар.* Кризистік жүйелердің кризистік емес жүйелерге қарағанда ақауларының салдарынан болатын шығындардың сомасы және зардаптарының маңыздылығы әдетте әлдеқайда жоғары. Верификацияға және валидацияға көбірек қаражат бөлінген жағдайда, осы жүйелер ақауларының пайда болу тәуекелі төмендейді. Клиентке берілгенге дейін жүйенің кемшіліктерін жою жүйенің салдарынан ақаулар немесе апаттар пайда болуда өтемдер төлеуге қарағанда әлдеқайда арзанға түседі.
2. *Функционалдық сенімділіктің валидациялық белгілері.* Кейде Сізге клиенттерге және бақылау органдарына жүйе сенімділікке қойылған талаптарға (ашықтыққа, сенімділікке және қауіпсіздікке) жауап беретіндігін ресми дәлелдеулеріңіз қажет болуы мүмкін. Кейбір жағдайларда Бақылауға бөтен бақылау органдары (мысалы, ұлттық авиация басқармалары) жүйені қолданысқа енгізгенге дейін оны сертификациялаулары қажет. Осындай сертификаттарды алу үшін валидация жүйесі қалайша іске асатындығын көрсету керек болады. Бұл үшін, жүйенің функционалдық сенімділігі туралы мәліметтер алуға болатындай B және B-ның арнайы үдерістерін жасап шығару қажет болуы мүмкін.

Осы себептерге бола кризистік жүйелердің верификациясы мен валидация құны басқа таптық жүйелерге қарағанда, әдетте әлдеқайда жоғары болады. Әдетте кризистік жүйелерге әзірлеулер бөлінетін қаржының жартысы осыған жұмсалады.

B және B-ның құны жоғары болғанына қарамастан, осы шығындар өзін ақтап жатыр, себебі олар ақаулар мен апаттар салдарынан пайда болатын шығындарға қарағанда едәуір төмен. Мысалы, 1996 жылы Ариан-5 зымыран-сақтаушының кризистік жүйенің бағдарламалық ақауының нәтижесінде бірнешесі серіктің жойылуына әкелді. Апат салдарынан ешкім зардап шеккен жоқ, бірақ шығындар сомасы жүздеген миллион долларды құрады. Тергеу барысында B және B жүйесінің кемішіліктері анықталған еді. Құны төмен тиімді жүргізген тексерулер осы апаттың алдын алар еді.

Қауіпсіздік және функционалдық сенімділікті қамтамасыз ету үдерісінде негізгі ықылас жүйе валидациясына берілгенмен, жүйеге әзірлеуге нақтылы та-

лаптар сақталғанын тексеру қажет. 13-бөлімде көрсетілгендей, жүйе сапасы оның әзірлеуде қолданылатын үдерістердің сапасына әсер етеді. Қысқаша айтқанда, жақсы үдерістердің нәтижесінде жақсы жүйелер өңделеді.

Қауіпсіздік және функционалдық сенімділіктер қамтамасыз ету үдерістерінің нәтижесі (мысалы, сараптамалық есептер мен тестілеу нәтижелері және т.б). жүйелер функционалдық сенімділігінің нақты дәлелдерінің кешені болып табылады. Бұл дәлелдер осы жүйені қолдану үшін жеткілікті сенімді және қауіпсіз болуы қажет. Кейде жүйелердің қауіпсіздігі және функционалдық сенімділігі дәлелдері мен нәтижесі функционалдық сенімділік бойынша есеп жасау үшін жинақталады. Ол клиентті немесе бақылаудың бөтен органын жүйелердің сенімділігі немесе қауіпсіздігін дәлелдеу үшін қолданылады.

15.1 Статикалық талдау

Статикалық талдаудың әдістері – бағдарламаларды орындауды ескермеген жүйелерді верификациялау әдістері. Олардың негізінде бағдарламалық қамтамасыз ету көзін ұсыну жатыр: мінездемелер мен немесе құрылым үлгісі немесе бағдарламалардың бастапқы коды. Статикалық талдау әдістерін ақауды анықтау үшін, жүйе мінездемелері және құрылымының үлгілерін тексеру үшін қолдануға болады. Олардың артықшылығы ақаулардың жүйелерді тексеруге кедергі келтірмейтіндігі. Бағдарламаларды тестілеу кезінде ақаулар өз-өздерін немесе басқа ақауларды бүркеуі мүмкін, сол себепті Сізге анықталған ақауларды жойып, қайта тестілеу өткізу қажет.

8-бөлімде айтылғандай, статикалық талдаудың өте әйгілі әдісі тәуелсіз сарапшылармен іске асатын баға және тексеру болып табылады, оның барысында сарапшылардан құралған топ құрылым немесе бағдарламаның өзіне мінездемелер беріп, тексеру өткізеді. Олар құрылымды немесе кодтарды ықыласпен тексеріп, барлық ықтимал қателер және ақауларды табуға тырысады. Тағы бір әдіс құрылымды үлгілеу аспаптарын үлгілеудің унификацияланған тілін тексеруде қолдану болып табылады (мысалы, әр түрлі объектілер үшін бірдей атаулардың қолданылуы). Дегенмен, кризистік жүйелерге статикалық талдаудың қосымша әдістерін қолдану қажет болуы мүмкін:

1. Формалды верификация, оның барысында бағдарлама өз мінездемелерге сәйкес келетінін көрсететін математикалық дәл дәлелдер жиналады.
2. Үлгіні тексеру, оның барысында бағдарламаны формалды сипаттаудың мақсатымен теоремалардың дәлелденуі қолданылады.
3. Бағдарламаларды автоматты талдау, оның барысында бағдарламалардың бастапқы коды потенциалдық қате келген комбинациялардың бар болуына тексеріледі.

Барлық әдістер өзара байланысты. Үлгілерді тексеру жүйесінің формалды үлгісіне негізделеді, ол арнайы спецификациялардың көмегімен құрастырылады.

Статикалық анализаторларды тексеру үшін лайықты код осы бағдарламаға кірістірілген формалды қайшы келмейтіндігін тексеруде қолданылады.

15.1.1 Формалды әдістер мен верификация

12-бөлімде айтылғандай, бағдарламалық қамтамасыз етуді әзірлеу әдістері жүйенің формалды үлгілеріне негізделеді, ол жүйенің спецификациялар жиынтығы ретінде қолданылады. Осы формалды әдістер спецификацияларды математикалық талдауға; спецификацияларды семантикалық баламалы тапсырмаға түрлендіруге; жүйенің семантикалық эквиваленттілігін верификацияға түрлендіруге негізделген.

Cleanroom әдісі

Cleanroom бағдарламалық қамтамасыз етулер әзірлеу әдісі Бағдарламалық қамтамасыз етуді формалды верификациялау мен статистикалық тестілеуге негізделген. Cleanroom үдерісінің мақсаты жүйелердің бағдарламалық қамтамасыз етудің ақауларының толықтай жоқтығы мен жүйе сенімділігінің жоғары дәрежесіне кепілдік беруге қолжеткізу болып табылады. Cleanroom үдерісі барысында бағдарламалық қамтамасыз етулердің барлық кеңейтпелері сипатталып, бұл сипаттамалар орындалады. Бағдарламалық қамтамасыз етудің ақаусыздығы формалды тәсілдің көмегімен көрсетіледі. Осы үдеріс барысында ақаулардың блоктық тестілеуі іске асады, ал тестілеу кезінде негізгі ықылас жүйелер жүйенің сенімділілігіне баға беруге бөлінеді.

http://www.SoftwareEngineering-9.com/Web/Cleanroom/

Формалды әдістер B және B үдерісінің әр түрлі кезеңдерінде қолданылады:

1. Реттілікті тексеру үшін жүйенің формалды спецификациясы өңделіп, математикалық талдаудан өтеді. Осы әдіс спецификацияларда қателер мен ақауларды анықтауда аса тиімді. Келесі бөлімде суреттеп айтылған үлгілерді тексеру, бір спецификацияларға талдау тәсілдердің бірі болып келеді.
2. Математикалық дәлелдерді қолданып, Сіз формалды бағдарламалық қамтамасыз ету жүйелерінің кодтарының өз спецификацияларына қайшы келмейтіндігін тексере аласыз. Бұл үшін формалды спецификация қажет. Осы әдіс бағдарламалық және құрылымдық ақаулар анықталған кезде тиімді.

Жүйенің формалды спецификациясы мен бағдарламалық код арасындағы семантикалық қарама-қайшылық салдарынан, жеке бағдарлама өз спецификацияларға қайшы келмейтіндігін дәлелдеуге болады. Сондықтан бағдарламалардың верификациясы трансформациялық әзірлеуге негізделген.

Трансформациялық әзірлеу үдерісінде формалды спецификация бағдарламалық кодқа айналады. Бағдарламалық қамтамасыз ету аспаптары өзгерулерді жүзеге асыруға мүмкіндік туғызып, лайықты тапсырмалардың ретті түрде жүзеге асатынын дәлелдейді. Бұл әдіс өте кең тараған формалды трансформациялау әдісі болып табылады (Эбриал 2005; Уорсуорс 1996). Ол басқару және авиациялық бағдарламалық қамтамасыз етулер мен темір жол жүйелерін әзірлеу үшін кең қолданылып жатыр.

Формалды әдістерді жақтаушылар осы әдістердің қолданылуы өте сенімді және қауіпсіз жүйелердің әзірленуіне алып келетінін айтады. Формалды верификация әзірленген бағдарлама өз мінездемелеріне жауап беріп, ақаулар жүйелердің функционалдық сенімділігіне нұқсан келтірмейтінін дәлелдейді. Егер Сіз CSP сияқты сондай тілде жазылған спецификациялар көмегімен параллель жүйелердің формалды үлгі жасасаңыз (Шнайдер, 1999), сіз бағдарламаның өзара блокировкасына алып келетін шарттарды анықтап, оларды жоя аласыз. Тек қана тестілеу көмегімен бұған қол жеткізу күрделі.

Бірақ, формалды спецификация сол бағдарламалық қамтамасыз ету оның практикалық қолдануында сенімді болатынының кепілі бола алмайды, мұның келесі себептері бар:

1. Спецификациялар жүйені қолданушылардың нақты талаптары қамтып көрсете алмайды. 12-бөлімде көрсетілгендей, жүйелер қолданушылары формалды белгілерді өте сирек түсініп, ал қателер мен ақауларды табу үшін олар формалды спецификацияларды тікелей оқи алмайды. Бұл спецификацияның құрамында қате бар екендігін және жүйелердің талаптарына дәл келмейтіндігін білдіреді.
2. Дәлелдің да құрамында қателер бола алады. Бағдарлама дұрыстығының дәлелдері үлкен және кешенді болып келеді, ал үлкен және кешенді бағдарламалардың құрамында әдетте ақаулар да болып жатады.
3. Дәлел бағдарлама қандай әдіспен қолданылатынына қатысты бұрыс жорамалдарға негізделуі мүмкін. Егер жүйе болжанғандай қолданылса, дәлел жарамсыз болып танылады.

Бағдарламалық қамтамасыз етуді қарапайым верификация едәуір уақыт алып, білімдердің және мамандандырылған аспаптардың (мысалы, теоремаларды дәлелдеуге арналған бағдарламалар) болуын талап етеді. Осылайша, бұл қаражатты талап ететіп үдеріс және жүйе үлкейген сайын формалды верификация құны да өседі. Сондықтан Бағдарламалық қамтамасыз етулер көптеген өндеушілердің көбісі формалды верификация тиімді екендігіне сенеді. Олар валидацияның арзанырақ әдістерін қолдана отырып (мысалы, тексерулер және тестілеу жүйелер көмегімен), бағдарламада сенімділік деңгейін көтеруге болатындығына сенеді.

Біздің көзқарасымыз бойынша, барлық кемшіліктерге қарамай, формалды әдістер мен формалды верификацияда бағдарламалық қамтамасыз етудің кризистік жүйелерін әзірлеуде маңызды рөл атқарады. Формалды спецификациялар жүйенің ең кең тараған ақауларын анықтауда аса тиімді. Формалды верифика-

ция ірі жүйелер үшін құнтсыз болса да, ол қауіпсіздіктің кризистік компоненттерін верификациялау үшін қолданыла алады.

15.1.2 Үлгіні тексеру

Дедукциялық тәсіл көмегімен бағдарламалардың формалды верификациясы өте күрделі болады, дегенмен формалды талдау үшін альтернативті тәсілдер құрастырылған болатын.

Осы тәсілдердің арасында ең нәтижелісі үлгілерді тексеру (Байер және Катен 2008). Ол аппаратты қамтамасыз ету жүйелерінің құрылымын тексеру үшін кең қолданылған еді және Марстың бетін зерттеу үшін аппараттардың бағдарламаларының бағдарлаушылары сияқты, НАСА бағдарламалық қамтамасыз ету сондай кризистік жүйелерін тексеру үшін (Риган және Хамильтон, 2004) және телефон байланысын қамтамасыз ету үшін бағдарламалық қамтамасыз ету жиі қолданылып жатыр (Чандра және др. 2002).

Үлгілерді тексеру бағдарламалық жүйе үлгісі мен осы үлгіні тексеру қамтамасыз етуден мамандандырылған аспаптарының көмегімен жүзеге асады. Үлгілердің тексерулердің көптеген алуан түрлі аспаптары құрастырылған еді, бірақ ең танымалысы SPIN болып табылады (Хольцманн 2003). Үлгілер тексерулері кезеңдері 15.1-суретте көрсетілген.

Үлгілер тексеру үдерісінде жүйенің формалды үлгісін құрастыруға негізделеді, бұл әдетте ұлғаймалы түпкі автомат формасында болады. Үлгілер олар тексерілетін тілде өрнектеледі. Үлгілер тексеру үшін бағдарлама SPIN, мысалы, Promela тілін қолданады. Жүйелер қасиеттерінің жиынтығы формалды белгілерде белгіленіп, жазылады, оның негізінде уақытша логика қағидасы жатыр. Осындай қасиеттің мысалы «жазу» режимінен «беру» режиміне ауыса алатын метеожүйелер.

15.1-сурет. Үлгіні тексеру

Содан соң тексеру бағдарламасы үлгінің аралық жолдарын тексереді (демек, жағдаяттар арасындағы барлық ауысу жолдарын), сонымен қатар, берілген қасиет осы үлгіде сақталатындығы тексеріледі. Егер ол сақталса, біресе үлгілерді тексеру бағдарламасы оны растап, үлгі осы қасиеттерге қатынаста дұрыс келетіндігі көрсетіледі. Егер белгілі бір жолда қасиет сақталмаса, үлгілерді тек-

серу бағдарламасы осы қасиет сақталмайтын жердің контрмысалын құрады. Үлгілерді тексеру, әсіресе параллель жүйелерді валидациялауда пайдалы, оларды уақытқа деген сезгіштігі салдарынан тестілеу өте күрделі болып табылады. Тексеру бағдарламасы өткінші, параллель ауысуларды және потенциалдық мәселелерді анықтай алады.

Үлгілерді тексерудің ең маңызды мақсаты жүйенің үлгілерін жасау болып табылады. Егер үлгіні қолдан жасау қажет болса (талаптар бойынша немесе жобалық құжаттамаға сай), онда бұл үдеріс арзан емес болады және көп уақытты қажет етеді. Оған қоса, жасалған үлгі талаптардың дәл үлгіге немесе құрылымға сай болмайтындығының ықтималдығы бар. Үлгіні автоматты түрде бағдарламалардың бастапқы кодынан жасау жақсы болар еді. Жүйе Java Pathfinder (Виссер және т.б., 2003), Java тілде код үлгімен тікелей жұмыс істеген үлгілердің тексеру жүйелерінің мысалы болып табылады.

Үлгіні есептеу арқылы тексеру өте қымбат, себебі ол үшін жүйе көмегімен өте күрделі тексеру тәсілі қолданылады. Жүйенің көлемі үлкейген кезде, жағдаяттардың саны да өседі, осыған орай тексерілуі қажетті жолдардың саны да өседі. Ірі жүйелер үшін үлгілер тексеру қажетті барлық тексерулерді жүзеге асыру үшін ЭЕМ жұмысы салдарынан уақыт бойынша құнтсыз болуы мүмкін. Нақтылы қасиет тексеру үшін қажетті сол компоненттердің идентификация алгоритмдері жақсаратындықтан, уақыт өте келе жағдаяттарының теңестірулерін кризистік жүйелерді әзірлеуде үйреншікті қолданылуы әлдеқайда жоғары бола бастайды. Бұл әдіс ақпараттық-бағдарланған ұйымдастыру жүйелері үшін онша қолданылмайды, дегенмен оны бағдарламалық етудің түпкі автоматтары сияқты кірістірілген жүйелердің үлгілерін верификациялау үшін қолдануға болады.

15.1.3 Автоматты статикалық талдау

8-бөлімде көрсетілгендей, бағдарламаларды тексеру әдетте қателердің бақылау тізімінің жиі негізде ғана эвристикалық процедураларға сүйене жасалады. Олар бағдармалаудағы әр түрлі тілдерде ортақ қателерді анықтайды. Осы тізімдер негізінде тексеру үдерісінің кейбір қателермен және эвристикалық процедуралары жағдайда автоматтандыруға болатын болса, кодтың бұрыс бөліктерін табуға қабілетті автоматты статикалық анализаторлар әзірленеді.

Статикалық талдаудың аспаптары жүйенің бастапқы кодымен жұмыс істеп, талдаудан кейбір түрлерімен, ақпараттың келесі енгізілуін талап етпейді. Бұл бағдарламашыларға бағдарламалар спецификацияларын жазу үшін арнайы белгілер үйрену керек емес екендігін білдіреді, ал мұндай талдаудың артықшылықтары анық болады. Осының арқасында автоматты статикалық талдауды үдеріске енгізу формалды верификация немесе үлгілерді тексеруден әлдеқайда жеңіл. Осылайша, ол статикалық талдаудың өте таралған әдісі болып табылады.

Ақаулық класы	Статистикалық талдауды тексеру
Ақпарат ақаулығы	Инициализацияның алдында қолданылатын ауыспалылар Ешқашан қолданылмайтын суреттелген ауыспалылар Иелендіру арасында қолданылмайтын екі рет иелендірілетін ауыспалылар Массивтің шекараларын бұзудың ықтималдылықтары Суреттелмеген ауыспалылар
Басқару ақаулығы	Қолжетпес код Нобайлардың тарамдалуы
Енгізу/шығару ақаулығы	Аралық иелендірусіз ауыспалылардың екі рет енгізілуі
Интерфейс ақаулығы	Параметрлік түрдің келісілмегендігі Параметрлердің саны келісілмегендігі Қызметтер нәтижелерінің қолданылуы Ретсіз қызметтер мен процедуралар
Сақтау ақаулығы	Белгіленбеген нұсқағыштар Нұсқағыштардың арифметикасы Жадыны қармауды және оны кейін босату

15.2-сурет. Автоматты статикалық талдауды тексеру

Автоматты статикалық анализаторлар – бағдарламалық қамтамасыз етудің аспаптары, олар ықтимал ақаулықтардан және қалыпсыздықтардан табу мақсатымен бағдарламалардың бастапқы мәтінін тексеріп отырады. Олар бағдарламалар мәтінінің синтетикалық талдауын жүзеге асырып, осылайша бағдарламада ұйғарымдардың әр түрлі түрлерін анықтап отырады. Содан соң олар, дұрыс қорытынды жасау үшін бағдарламаларға логикаға бағдарлаушы ұйғарымдар құрастырып, көптеген жағдайларда, осы бағдарламалардың барлығын ықтимал мәндерінің жиынын есептеп шығарады. Олар тілдің компиляторымен ұсынылатын қателердің анықтаулар құралдарын толықтырып, тексеру үдерісі үшін қолдана алады немесе В және В үдерісіне жеке шара ретінде оларды енгізе алады. Кодтардың толық талдаулары сияқты емес автоматты статикалық талдау уақыттың және құралдардың азшылығын талап етеді. Бірақ ол бағдарламалар бірлескен инспекция кезінде қателердің кластарын анықтай алады.

Автоматты статикалық талдаудың мақсаты есеп бағдарламаларының қалыпсыздықтар кодын оқырманның ықыласын тарту, мысалы, инициализациясыз қолданылған айнымалы мәндер мүмкін диапазон шектердің артына шығатын. 15.2-суретте статикалық талдаудың көмегімен анықтауға болатын мәселелердің мысалдары берілген. Арнайы тексерулер бағдарламалау тілімен бейнеленіп, сол ретсіз тілге немесе басқа мүмкін тілге тікелей тәуелді болады. Қалыпсыздықтар көбінесе бағдарламалаудың қателерінің нәтижесінде пайда болады, ал олар бағдарламаларды орындаудағы ақауларды келтіреді. Бірақ Сіз осы қалыпсыздықтар бағдарламалар ақаулықтарымен міндетті түрде келмейтінін түсінуге тиістісіз, олар

қасақана енгізілген конструктивтік элементтермен немесе ешқандай теріс зардаптары жоқ қалыпсыздықтармен енгізілуі де мүмкін.

Статикалық анализаторлар тексерудің үш деңгейін жүзеге асыра алады:

1. *Мінездемелік қателерді тексеру.* Статикалық анализаторлар осы деңгейде Java немесе C сияқты тілдерде бағдарламашылармен жіберетін өте таралған қателер туралы біледі. Аспап осы мәселе үшін тән болатын үлгілерді табу мақсатымен кодтарды талдап, оларды бағдарламашыға көрсетеді. Бұл құрал өзінің салыстырмалы оңайлығына қарамастан, еі көп тараған ақауларды талдауда тиімді болып келеді. Жень және оның қызметкерлері (2006) C және C++ тілдерінде жазылған код негізінде статикалық талдауды қолданудың тиімділігін зерттеп, бағдарламаларда 90% қателердің ішінен мінездемелік қателердің 10 түрлері бар деген нәтижеге келді.
2. *Қолданушымен анықталатын қателерді тексеру.* Осы тәсіл статикалық анализаторды қолданушыларға қателердің үлгілерін дербес анықтауға мүмкіндік береді. Бұл әсіресе реттілікті сақтау керек кезде тиімді (мысалы, B әдісін қарастырмас бұрын әрдайым A әдісін қолдану қажет). Уақыт өте келе ұйым олардың бағдарламаларында пайда болатын таралған қателер туралы мәліметті жинай алады және статикалық талдаудың аспаптары ақауларды өз бетінше анықтай алатындай жетілдіре алады.
3. *Шарттарды тексеру.* Статикалық талдаудың осы тәсілі өте таралған және тиімді. Өңдеушілер формалды шарттарды өз бағдарламаларына қосып, нақтылы уақытқа сақталуға тиісті компоненттердің арақатынастары анықтап түсініктемелер түрінде көрсетеді. Мысалы, мәні x...y диапазонында болуға тиісті кейбір айнымалы шарт енгізілген. Анализатор кодтарды бейнелі түрде орындап, сақтай алатын сол шарттар көрсетіледі Splint (Эванс және Ларошелль, 2002) және SPARK Examiner (Кроксфорд және Саттон 2006) сияқты анализаторларда осы тәсіл қолданылады.

Статикалық талдау бағдарламаларда қателерді табуда тиімді, дегенмен ол да жиі «ақау жібереді». Басқаша айтқанда, ол қателері болмаған кодтың бөліктерін ерекшелейді, статикалық анализатордың ережелері потенциалдық қателердің пайда болу мүмкіндігін табады. Жалған реакциялардың саны шартты формадағы бағдарламаны қосымша мәлімет беру жолымен қысқартуға болады, дегенмен бұл қосымша өңдеушіден қосымша жұмысты талап етеді. Осы жалған реакцияларды табу үшін кодты тескерместен бұрын скрининг өткізу қажет.

Статикалық талдау қауіпсіздікті тексеру үшін, әсіресе, бағалы (Эванс және Ларошелль, 2002). Буферлер асыра толтыру немесе енгізу қаталері сияқты, статикалық анализаторларда сондай таралған мәселелерді тексеруге қолдануға болады. Хакерлер бұза алатын әлсіз орындарды қорғау стратегиясы жақсарту үшін бұл әдіспен тиімді.

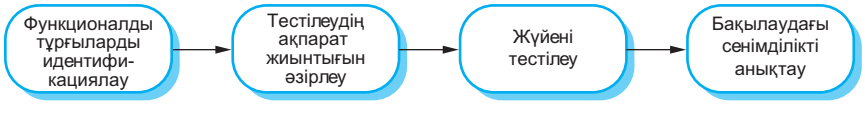
Кейін біз, қауіпсіздікті тестілеу күрделі болатындығы және бұзушылар тестілеуді жүргізетін орындаушылар бағдарламаларды қорғаудағы күтпейтін кенет тәсілдерді қолданулары мүмкіндігі туралы сөз етеміз. Статикалық анализа-

торларда тестілеу орындайтын қауіпсіздік туралы көлемді білімдер болады,және бұлар бағдарламаларды тестілеу алдында қолданылуы шарт. Статикалық талдауды қолдануда Сіз бағдарламаның барлық ықтимал орындаулары үшін нақты болатын сауалдар жасай аласыз, олар Сіз құрастырған тестке сәйкес келуі қажет.

Статикалық талдау бүгінгі күнге бағдарламалық қамтамасыз етуді әзірлеу үдерісінде көптеген ұйымдармен күн сайын қолданылып жатыр. Құрылымдардағы драйверлерді әзірлеуде бағадарламаның ақаулары кері әсерін тигізген кезде статикалық талдауды бұрын Microsoft қолданатын (Ларус және т.б.). Бүгін олар бағдарламалар сенімділігіне әсер ететін қауіпсіздік қателері мәселелерін табу мақсатымен көптеген әзірлеулер үшін осы тәсілді қолданып жатыр (Болл және басқалар, 2006). Көптеген кризистік жүйелер үшін (мысалы, авиацияда және ядролық өнеркәсіптегі жүйелер үшін) статикалық талдау үдерістің үйреншікті бөлігіне айналды (Нгюен және Оргханлиан 2003).

15.2 Сенімділікті тестілеу

Сенімділікті тестілеу – бұл басты мақсаты жүйенің сенімділігін анықтау болып табылатын тестілеу үдерісі. 10-бөлімде айтылғандай, сенімділікті анықтаудың бірнеше жүйесі бар (мысалы, POFOD), ақауларда пайда болғанда, жиілікте және қарқында ақаудың ықтималдықтары. Олар бағдарламалық қамтамасыз ету сенімділілігін сандық анықтау үшін қолданыла алады. Сенімділікті тестілеу үдерісінде Сіз сенімділік қажетті деңгейге сәйкес келетінін тексере аласыз.



15.3-сурет. Сенімділікті анықтау

Жүйелер сенімділігін анықтау үдерісі 15.3-суретте көрсетілген. Осы үдеріс төрт кезеңнен тұрады:

1. Сіз бірдей типтегі жүйелерді зеттеуде оларды практикада қалай түсінуге болатындығын тексеруге болатындығын білесіз. Сіз жүйелер сенімділігін қолданушының өзі сияқты өлшеуіңіз қажет. Функционалдық кескіндердің анықталуы сіздің мақсатыңыз. Функционалдық кескін сол жүйелер және ықтималдық енгізулердің кластарын белгілеп, не осы енгізулерді жүйеде жұмыста қолданады.
2. Содан соң Сіз функционалдық тілді қамтып көрсететін тестілеуді құрастырасыз. Бұл Сізге осы сол ықтимал бөлумен тестілеу жасау қажет екендігін және осы зерттелген жүйелер үшін тестілеу құрастыруыңыз қажет екендігін көрсетеді. Осы үдеріс үшін әдетте тестілеу генераторы қолданылады.

3. Сіз жүйені тестілейсіз де, осы ақпаратты қолдана отырып, пайда болатын ақауларды саны және түрі жағынын есептеп шығарасыз. Сонымен бірге ақаулардың пайда болу уақыты да анықталады. 10-бөлімде айтылғандай, уақыттың өлшемдері таңдаулы сенімділіктер өлшемдері бірліктеріне сәйкес келуі тиісті.
4. Кейіннен, Сіз ақаулардың статистикалық мағыналы санын аласыз, Сіз бағдарламалық қамтамасыз етулердің сенімділігін есептей аласыз және сенімділіктер өлшемдер жүйелерінің лайықты мәні белгілеуге мүмкіндік аласыз.

Осы төрт сатылы тәсіл «статистикалық тестілеу» деп те аталады. Статистикалық тестілеудің мақсаты – жүйелер сенімділігіне баға беру. Оның 8-бөлімде суреттелген, тестілеу міндеттерінен айырмашылығы бар, оның мақсаты – жүйелер ақаулықтарын табу. Проуелл және т.б. (1999) Cleanroom статистикалық әдіс көмегімен бағдарламалық қамтамасыз етулерге арнаулы әзірленетін тестілеуге жақсы сипаттама келтірген болатын. Бұл концептуалды тартымды тәсілді тәжірибеде жүзеге асыру қиын. Келесі маңызды қиындықтардың қатары пайда болуы мүмкін:

1. *Функционалдық кескіннің дәлсіздігі.* Тәжірибеде басқа жүйелердің жұмыстарына негізделген функционалдық кескіндер, жасалатын жүйелердің нақты қолданылуын бейнелей алады.
2. *Тестілеу ақпараттарын генерациялаудың жоғары құны.* Ақпараттың үлкен көлемін жасау осы функционалдық профиль үшін қажетті, жүйе толық автоматтандырылған жағдайлардан басқа бұл аса қымбат үдеріс.
3. *Сенімділіктің биік дәрежесіндегі статистикалық дәлсіздік.* Сенімділіктің дәл өлшемін іске асыру үшін, ақаулардың статистикалық мағыналы санын құру керек. Егер бағдарламалық қамтамасыз етудің сенімділік дәрежесі биік болса, ақаулардың саны төмен болады және жаңа ақаулардың генерациялану мүмкіндігі де төмендейді.
4. *Ақаулардың айқындалуы*. Жүйелердегі ақаулар әрдайым анық бола бермейді. Егер Сіз формалды спецификацияны қолданып жатқан болсаңыз, Сіз тап болған мінездемелерден ауытқулар анықтай аласыз, бірақ егер спецификациялар табиғи тілде болса, бірқатар жалған дәлелдер пайда бола алады, осыған орай ақаулар пайда болу туралы бақылаушылардың пікірлері де өзгеше болады.

Қазіргі кезде ақпараттардың ірі жиынтықтарын жасауда ұтымды әдіс – осы жүйенің сенімділігін анықтау үшін функционалдық тілге дәл келетін енгізулерді автоматты жасауға болатындай тестілеу генераторын қолдану. Дегенмен, интерактивті жүйелерге ақпаратты тестілеуді енгізулер шығарылымдарға деген реакцияға сай келгендіктен, жүйені тестілеу үшін өндіріс үдерістерін әрдайым автоматтандыруға келе бермейді. Осы жүйелер үшін жиынтықтарды қолдан құруға

болады, бірақ бұған көп шығын шығады. Егер толық автоматтандыру жасауға болса, тестілеу ақпаратын генераторға жазу көп уақыт алады.

Статистикалық тестілеу ақаулықтарды енгізу туралы жинақтау үшін қолданылады. Ақаулықтардың енгізілуі (Boac 1997) – бағдарламаға қателерді біле тұра енгізу. Бағдарламаларды орындауда олар бағдарламалар ақаулықтарының және қателерінің себебі болады. Содан соң, анықтау үшін, Сіз ақаулардың талдауын жүзеге асырасыз, осы ақаулардың салдарынан түбірлік қателер пайда болады. Егер Сіз, X% енгізілген қателер ақауларды алып келетінін анықтасаңыз, қателердің енгізулері әдісін жақтаушылар, тестілеу үдерісінде, тек қана жүйелердің X% анық ақаулықтардың пайда болуы байқалады.

Осының барлығы, әрине, не енгізілген ақаулықтардың бөлінуі және түрі практикалық қолдану үдерісінде пайда болатын ақаулықтармен дәл келіп жатады. Осы бекіту ақаулықтарды бағдарламалайды, бірақ қателерді енгізуі қателердің нәтижесінде пайда болатын ақаулықтардың санын анықтау үшін тиімді әдіс.

Статистикалық тестілеудің нәтижесінде B және B басқа үдерістерінің барысында бағдарламалық қамтамасыз етулерде қателер жиі пайда болады. Бұл қателер жүйелердің сенімділігі қойылған талаптарға сәйкес келмегендігін куәландыра алады және қосымша жұмысты талап етеді. Оның сенімділікті қайтадан бағалау үшін, жүйеден істеп бітіруден кейін қайта тестілеуге болады. Осыдан соң енгізу үдерісі бірнеше рет істелінген кезде тестілеу нәтижелері бойынша экстраполяция жасалады және сенімділіктер қажетті деңгейі болжанады. Экстраполяция жасау жүйелер сенімділігін уақытпен ұлғайтып, сенімділікті жоғарылату үлгілерін осы үшін даярлайды. Бұл жоспарлауды ұйымдастыруға көмектеседі. Сенімділікті жоғарылату кейде үлгіні түсінуге көмектесіп, сенімділікті тиісті деңгейге жеткізіп, ал ұсынылатын талаптарды қайтадан талқылауға әкеледі.

15.2.1 Функционалдық кескіндер

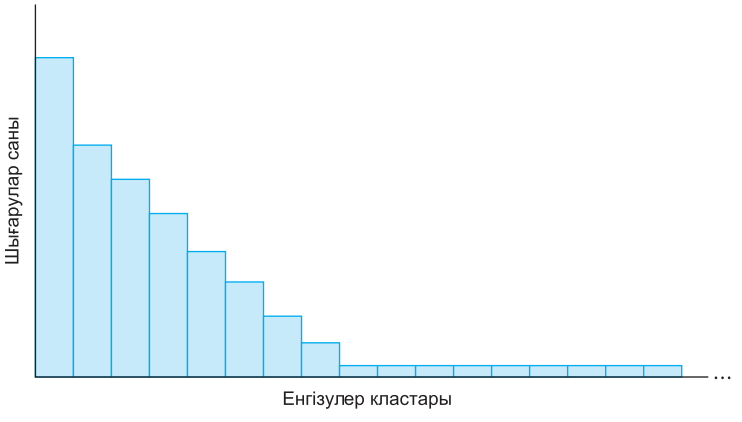
Жүйенің функционалдық кескіні тәжірибеде қолданылатын бағдарламалық қамтамасыз ету жүйелері функционалдық тілін бейнелейді. Ол спецификациялық енгізулер мен олардың пайда болу ықтималдықтарынан тұрады. Егер жаңа бағдарламалық қамтамасыз ету қазіргі автоматты жүйені алмастырса, жаңа бағдарламалық қамтамасыз етуді қолдануда потенциалдық әдісті анықтау жеткілікті болады. Ол жаңа бағдарламалық қамтамасыз етуге енгізген (болжамды) жаңа қызметтер туралы есепке енгізілетін жорамалдардың қолданылуы алдыңғы тәжірибеге сәйкес келуі тиіс. Мысалы, функционалдық кескін телефон сигналдарының коммутация жүйелері үшін тиісті телекоммуникациялық серіктестіктер қоңырау үлгілерімен таныс болулары қажет.

Әдетте функционалдық кескін биік ықтималдыққа ие болатын енгізулер көптеген класстарға бөлінеді, бұл 15.4-суретте көрсетілген. Кластардың саны көп болады, олардың енгізулерінің ықтималдылығы мүлдем жоқ емес, бірақ өте төмен. Олар 15.4-суреттің оң бөлігінде көрсетілген. Көп нүкте (…) ұқсас ерекше енгізулердің анағұрлым үлкен санын көрсетеді.

Сенімділікті жоғарылатуды үлгілеу

Сенімділікті жоғарылатудың үлгісі – сенімділік жүйелерін тестілеу үдерісінде өзгерістерді көрсететін үлгі. Осы ақаулықтар анықталғаннан кейін, осы ақаулардың негізіне жататын қателер түзеліп, сол себепті жүйенің сенімділігі уақыт өте келе тиісті тестілеуді өткізіп, қателерді жоюға көмектеседі. Сенімділікті болжау үшін қажетті өзгертулерді жоғарылату концептуалды математикалық үлгіге сүйенеді.

http://www.SoftwareEngineering-9.com/Web/DepSecAssur/RGM.html



15.4-сурет. Функционалдық кескін

Телекоммуникациялық жүйелердің функционалдық тілдерін дамытуды Муса (1998) суреттеді. Себебі осы сала үшін мәліметтер жинау ұзаққа созылып, функционалдық профильдің әзірлену үдерісі жеткілікті анық болды. Ол қолдану туралы тарихи ақпаратты бейнелейді. Жүйе үшін функционалдық профиль шамамен 15 адам-жылдарын 1 адам-айларын талап етеді. Функционалдық тілді жасауға басқа жағдайларда (2-3 адам-жыл) үлкен уақытты талап етеді, бірақ оның құны бірнеше жүйелер шығарылымдарға жіктеліп жатады. Мусаның ойынша, серіктестік функционалдық тілді әзірлеуге, ең кемінде 10-еселі қаржының құйылуын сталап етеді.

Егер бағдарламалық қамтамасыз ету жүйесі инновациялық болса, оның қолдануының әдістерін болжау күрделі. Сәйкесінше, дәл функционалдық кескінде жасау мүмкін емес болады. Жаңа жүйе әр түрлі қолданушылармен тәжірибеде қолданылады. Осы қолдану туралы тарихи базасы жоқ. Осы қолданушылар сондай әдістердің көмегімен қолдана алады.

Дәл функционалдық кескінді әзірлеу әбден ықтимал жүйелердің кейбір түрлері үшін (мысалы, телекоммуникациялық жүйелері) үйреншікті үлгісі болып табылады. Жүйелердің басқа түрлері әр түрлі қолданушылардың әрбір жүйені

қолдануында көрінеді. 10-бөлімде көрсетілгендей, әр түрлі қолданушылар бір сенімділік туралы тікелей қарама-қарсы әсерлер алады және оның әр түрлі түрін қолданады.

Мәселе функционалдық кескіндер статикалық болмайтындығымен ушығады, ол жүйенің қолдануымен бірге өзгеріп отырады. Қолданушылар жүйе туралы көбірек мәліметтер алғанда, олар оны едәуір күрделі міндеттер үшін қолдана бастайды. Сондықтан дәл функционалдық кескінді әзірлеу, көбінесе мүмкін емес. Демек, Сіз жүйені қолдану әдістері туралы бұрыс жорамалдарда негізделген сенімділікке кез келген өлшемдердің дәлдігіне дәйекті бола алмайсыз.

15.3 Қауіпсіздікті тестілеу

Кризистік жүйелердің қауіпсіздігін бағалау Ғаламтор арқылы жүзеге асып отыр, демек, кез келген қолданушы оған Ғаламтор жүйесі арқылы тікелей қосыла алады. Ғаламтор-жүйелерінің бұзылуы туралы жаңа әңгімелер күн сайын пайда болуда, ал вирустар болса Ғаламтор-хаттамалары көмегімен күнделікті таралып отыр.

Ғаламтор-жүйелердің верификация және валидация үдерістеріне негізгі ықылас қауіпсіздіктер баға беруге бөлінуі қажет екендігі күмәнсіз. Жүйелердің әр түрлі бұзуларға қарсы тұру қабілеттілігін тестілеу қажет. Дегенмен, осындай қауіпсіздікті бағалауды іске асыру Андерсонның пайымдауынша, (2001) өте қиын. Демек, қолданылатын жүйелердің қауіпсіздігінде кетіктер бар. Хакерлер осыларды қолданып, жүйеге кіруге рұқсат алады.

Үлкен есеп бойынша, қауіпсіздікті тестілеу күрделілігінің екі себебі бар:

1. Қауіпсіздікке талаптар «тиісті емес» түрдің талаптармен келіп жатыр. Басқа сөздермен, олар жүйеден функционалдықпен және жүйенің жұмысын анықтап, жүйемен тексерілетін қарапайым шектеулер формада ұқсас тәртіптерді анықтау керек.
   Қажетті ресурстар бар болғанда, ең кемінде теория жағынан Сіз жүйе өз функционалдық талаптарға жауап беріп жатқанын көрсете аласыз. Бірақ жүйе әлдеқандай бөтен қызметтерді орындамағанын дәлелдеу мүмкін емес. Тәуелсіз тестілеу қауіпсіздікте жұмсаған уақыт пен жүйеге жасалған енгізулерден кейін қауіпсіздік кемшіліктерге ие бола бастайды. Әрине, Сіз бұзудың кейбір белгілі түрлерден жүйе қорғау үшін арналған функционалдық талаптар жасай аласыз. Бірақ, Сіз шабуылдардың ескерілмеген түрлер үшін талаптарды жасай алмайсыз. Ағымға көптеген жыл қолданған жағдайда жүйелермен, ойлап тапқыш бұзушы бұзудың жаңа формасын ойлап тауып, қауіпсіз болып есептелген жүйеге кіре алады.
2. Жүйелерді бұзушы адамдар ақылды және кемшіліктерді әрдайым табанды іздейді. Олар жүйемен тәжірибе жасап, тіпті алыс жүйеден үйреншіктіден алыс қызметтен және қолдануларды іске асырғылары келеді. Олар, мысалы, аты-жөні жазылатын жерге тыныс белгілер, әріптер, цифрлар және 1000 таңбалардан тұратын комбинацияны енгізе алады. Олар жүйеде кетік

тапқан соң, ол туралы мәліметпен сыр қылып айта алады, үлкейтіп сол кетікті жүйеге потенциалдық бұзып кірудің көзі ретінде қарастырады.

Бұзушылар өңдеушілердің жорамалдарын тауып, кейін осы жорамалдардан қарсы барады. Олар уақыттың ағымға ұзақ мерзімдік жүйені қолдана және бағдарламалық аспаптар қолданып талдау жасай, кемшіліктерді анықтау үшін олар қолдана алады. Іс жүзінде, олар кетіктерді іздестіруге, мамандар тестілеуге жұмсағандай емес, көбірек уақыт жұмсай алады.

Бұл себеп бойынша статикалық талдау әсіресе қауіпсіздікте тестілеу аспабы ретінде пайдалы. Бағдарламалардың статикалық талдауы қателер және білместіктер бола алатын бағдарламалар бөлімшелер сынау бойынша мамандардың командасы тез көрсете алады.

Статикалық талдау нәтижесінде байқалған қалыпсыздықтар бірден дұрысталады немесе түсіну тесттер анықтау үшін қолданылады.

Жүйе қауіпсіздік тексеру үшін Сіз тестілеу, құралдық талдау және формалды верификация комбинациясын қолдана аласыз:

1. *Тәжірибе негізіндегі тестілеу.* Осы жағдайда жүйе валидациясы белгілі командаға бұзулардың сол түрлерден қорғауларға тиімділігін талдайды. Бақылау үшін жүйелердің бастапқы кодын зерттеу және әзірлеу талап етіледі. Мысалы, SQL жүйе кэшінде жалған жазулар жолымен бұзуға тексеру үшін, Сізге жүйені тестілеп, SQL командалары болатын енгізулерді қолдану қажет. Буферлерді асыра толтыру қателері пайда болмайтынына көз жеткізу үшін, Сізге барлық буфердің енгізуін зерттеу қажет, не олар тап болған шекті бұзбау үшін бағдарлама мәндерін буфер элементтеріне тексеру қажет. Валидацияның осы түрі әдетте құралдық валидациямен бірге қолданып жатыр, тестілеу жүйелері негізінде есептер анықтау Сізге мәліметті алуға көмектеседі. Қауіпсіздіктің белгілі мәселелерін бақылау тізім жасауға да болады. 15.5-суретте кейбір алдыңғы тәжірибеге негізде тестілеу өткізу үшін қолдана алатын сұрақтардың мысалдары көрсетілген. Қауіпсіздікте мәселелерді бақылау тізімге, сонымен бірге тексерулерді қосу сол құрылым және бағдарламалауды әзірлеу үшін кепілдемелер береді (14-бөлім).
2. *Шабуыл әдісімен қауіпсіздікті анықтау.* Бұл қолданбалы жүйені тестілеуге оны құрастыруға қатыспаған мамандарды тартуға мүмкіндік беретін тәжірибе негізіндегі тестілеу нысандарының бірі. Сізге жүйе қауіпсіздігін сақтау талабы қойылатын «шабуыл командасын» құру қажет. Команда жүйеге жасалған шабуылдарды сылтау етіп, жүйе қауіпсіздігін әшкерелеудің жаңа әдістерін анықтау үшін өз шеберлігін пайдаланады. Шабуыл командасының қатысушылары қауіпсіздікті тестілеу мен жүйе кемшіліктерін анықтаудың алдыңғы тәжірибесін иемденуі тиіс.
3. *Аспапты тестілеу.* Бұл әдіс жүйені талдау мақсатында қауіпсіздікті қамтамасыз етудің түрлі құралдарын пайдаланумен түсіндіріледі (мысалы, парольдерді тексеру құрылғысы). Парольдерді тексеру құрылғысы кең

таралған есімдермен қатар әріптердің жиынтығынан тұратын қауіпсіз емес парольдерді анықтайды. Бұл тәсіл тәжірибе негізіндегі кеңейтілген валидация нұсқасы болады, мұндағы алдыңғы кемшіліктер тәжірибесі пайдаланылатын аспаптарға енгізіледі. Сондай-ақ, статикалық талдау да аспаптық тестілеудің бір түрі болады.

Қауіпсіздік параметрлерінің бақылау тізімі
1. Қосымша арқылы жасалатын барлық файлдарда тиісті рұқсат бар ма? Тиісті рұқсаттың болмауы аталған файлдарға санкцияланбаған қолданушының қол жеткізуіне әкеліп соғады.
2. Жүйе белгілі бір әрекетсіздік мерзімінен кейін қолданушылық сеансты автоматты түрді тоқтата ма? Тоқтатылмаған сеанстар санкцияланбаған қолданушыға операторсыз жұмыс істейтін компьютер арқылы рұқсат алуына жол берді.
3. Егер жүйе бағдарламау тілінде ауқым шегін тексерусіз жазылса, буфердің толығумен пайдалану жағдайы тууы мүмкін бе? Буфердің толығуы қарақшыларға код бірізділігі жүйесін жолдап, содан соң мұны орындауға мүмкіндік береді.
4. Егер пароль қойылса, жүйе сол парольдің сенімділік деңгейін тексере ме? Сенімді парольдер әріп, сан және тыныс белгілерінен тұратын жиынтықты құрайды – мұндай парольдерді анықтау қиынға соғады. Сөздіктердегі қарапайым сөздерді қолданудан сақтану қажет.
5. Жүйенің енгізу ортасының енгізу спецификацияларына сәйкестігі тексеріле ме? Дұрыс жасалмаған енгізулерді қате ұқсату қауіпсіздік салдарының пайда болуының көп тараған себептері болады.

15.5-сурет. Қауіпсіздік параметрлерін бақылау парағы тармақтарының мысалдары

4. *Формалды верификация*. Жүйе қауіпсіздіктің формалды спецификациясы бойынша сәйкестік мәніне тексерілуі мүмкін. Алайда, басқа салалардағы сияқты, қауіпсіздіктің формалды спецификациясы кең қолданылмайды.

Қауіпсіздікті тестілеу үнемі уақытқа және команданың қолжетімді ресурстарына шектеулі болады. Бұл әдетте қауіпсіздікті тестілеу үшін тәуекел негізіндегі тәсілдерді қолданып, Сіздің пікіріңіз бойынша жүйе үшін мәнді тәуекелдер ұсынылатынына назардың аудару қажеттігін білдіреді. Егер Сізде жүйе тәуекелдерін талдау нәтижелері болса, олар тестілеу үдерісінің негізі ретінде қолданылуы мүмкін. Тестілеу командасы аталған тәуекелдермен ескертілген қауіпсіздік талаптарына жүйенің сәйкестігін тестілеуі тиіс және баламалы тәсілдер қолдана отырып, жүйеге қауіп төндіретін жүйені бұзуға тырысуы тиіс. Жүйенің соңғы қолданушыларына оның қауіпсіздігін тексеру қиынға түседі. Сондықтан Солтүстік Америка мен Еуропа елдерінің үкіметтік органдары мамандандырылған бағалаушылар тексере

алатын қауіпсіздікті бағалаудың бірқатар критерийлерін белгіледі (Пфлеегер және Пфлеегер, 2007). Бағдарламалық қамтамасыз ету жабдықтаушылары өз өнімдерін бағалап, аталған критерийлер негізінде сертификаттауды жүзеге асыра алады. Осылайша, егер Сіз қауіпсіздіктің белгілі бір деңгейіне талап қойсаңыз, тиісті қауіпсіздік деңгейі ресми валидациядан өткен өнімді таңдай аласыз. Алайда, іс жүзінде бұл критерийлер әскери жүйелерде қолданылған, сондықтан бүгінгі күнге дейін кең көлемде коммерциялық ретінде танылмаған.

15.4 Функционалдық сенімділікті қамтамасыз ету үдерісі

13-тарауда аталғандай және тәжірибе көрсеткендей сенімді үдерістер нәтижесінде сенімді жүйелер береді. Басқа сөзбен айтқанда, егер үдеріс негізінде бағдарламалық қамтамасыз етуді ұйымдастыру мен бақылаудың тиісті ережелері болса, нәтижесінде алынған өнім де сенімділігімен ерекшеленеді.

Бағдарламалық қамтамасыз етуді бақылау

Бақылау органдарын жеке өнеркәсіп қауіпсіздік, сенімділік және т.б. ұлттық стандарттарын сақтамау жолымен пайда көрмейтініне кепілдік беру үшін үкімет құрады. Өнеркәсіптің әр түрлі салаларында арнайы бақылау органдары бар: атом энергетикасы, авиация, банк салалары. Бағдарламалық қамтамасыз ету елдер инфроқұрылымдарында маңызды орын алуына байланысты, бұл бақылау органдары бағдарламалық қамтамасыз ету жүйелерінің қауіпсіздігі мен функционалды сенімділігіне үлкен мән беруде.

http://www.SoftwareEngineering-9.com/Web/DepSecAssur/Regulation.html

Әрине, сапалы үдеріс өз бетімен сенімділік кепілі бола алмайды. Алайда, сенімді үдерісті пайдалану фактісі жүйенің сенімді болуын арттырады. Функционалдық сенімділікті қамтамасыз ету үдерісінің негізгі міндеті – жүйені құрастыру барысында қолданылған үдерістер және осы үдерістердің нәтижелері туралы ақпарат жинау. Аталған ақпарат бағдарламалық қамтамасыз етуді құрастыру барысында жүргізілген талдау, тексеру және сынау фактілерінен тұрады.

Функционалдық сенімділікті қамтамасыз ету үдерісінің негізіне екі мәселе жатады:

1. Қолданылатын үдеріс дұрыс па? Ұйым қолданатын жүйені құрастыру үдерістеріне құрастырылатын жүйе типіне жарамды бақылау субүдерістері мен В және В жата ма?

2. Біз бұл үдерісті дұрыс орындаймыз ба? Ұйым бағдарламалық қамтамасыз етуді құру үдерісінің сипаттамасына сәйкес құрастыру бойынша жұмысты ұйымдастырды ма? Бағдарламалық қамтамасыз етуді құрастыру үдерісінің белгілі бір нәтижелері алынды ма?

Критикалық жүйелерді құрастыру саласында тәжірибесі бар компаниялар өз үдерістерін, олар верификация мен валидация тәжірибелерінің ұтымды тәжірибелерін ұсынатындай етіп жасайды. Кейбір жағдайларда бұл сыртқы бақылау органдарымен қандай үдерістерді қолдану тиімді болатыны туралы келліссөз жүргізуді білдіреді. Әр түрлі компаниялар қолданатын үдерістер бірбірінен өзгеше болғанымен, критикалық жүйені құрастыру үдерістерінің сатылары жалпы бірдей болады: талаптарды анықтау, өзгерістерді және конфигурацияларды бақылау, жүйені үлгілеу, тексерулер мен қадағалаулар, сынақтарды жоспарлау, тестілеу тиімділігін талдау. Құрастыру үдерістеріне тиісті тәжірибе енгізуді білдіретін үдерісті жетілдіру ұғымы 26-тарауда сипатталған.

Функционалдық сенімділікті қамтамасыз ету үдерісінің басқа аспектісі – бұл барлық үдерістер тиісті түрде орындалғанын тексеру. Әдетте бұл үдерістің тиісті құжаттамасын қамтамасыз ету мен сол құжаттаманы тексеруді білдіреді. Мысалы, бағдарламаларды ресми тексеру сенімді үдерістің бір бөлігі болады. Әр тексерудің құжаттар жиынтығына тексеру жүргізуде қолданылатын бақылау тізімдері, тексеруге қатысатын тұлғалар тізімі, тексеру кезінде анықталған мәселелер және қажетті шаралар енеді.

Осылайша, сенімді жүйенің қолданылуын дәлелдеу үшін үдеріс пен құрастырылатын қамтамасыз етудің сенімділігін растайтын бірқатар құжаттар жасалуы тиіс.

Мұндай ауқымды құжаттамаларды қажетсіну қауіпсіздік пен функционалдық сенімділік спецификациясының талаптары қойылған жүйелерде икемді үдерістердің сирек қолданылатынын куәландырады. Икемді үдерістер бағдарламалық қамтамасыз етудің өзіне бағытталып, үдеріс құжаттамасының көпшілік бөлігі ресімделгеннен кейін қолданылмайтыны туралы пайымдалған (бұл әділ пайым). Алайда Сізге үдеріс туралы ақпарат жүйенің қауіпсіздігі мен функционалдық сенімділігі туралы есептің бір бөлігі ретінде қолданылғанда, дәлелдер алып, үдеріс сатыларын құажаттамалық тіркеу қажет.

15.4.1 Қауіпсіздікті қамтамасыз ету үдерістері

Қауіпсіздікті қамтамасыз ету бойынша жұмыстың басым бөлігі критикалық жүйелерді құрастыру кезінде жүзеге асырылады. Қауіпсіз критикалық жүйелерді құрастыру үдерісіне екі себеп бойынша талдау мен қауіпсіздікті қамтамасыз етумен байланысты В және В үдерістері енуі маңызды:

1. Критикалық жүйелер жағдайында апаттар сирек болады және жүйені тестілеу кезінде оны дамыту мүмкін болмайды. Егер сіз апатқа әкелуі мүмкін

шарттарды жандандырғыңыз келсе, сынақтың қолданыстағы рәсімдеріне сене алмайсыз.

2. 12-тарауда аталғандай, қауіпсіздікке қойылатын талаптар әдетте жүйенің қауіпсіз емес қызметін болдырмайтын «болмауы тиіс» талаптарына сәйкес келеді. Тестілеу мен валидацияның басқа шаралары арқылы бұл талаптардың сақталғаны туралы шынайы дәлелдеу мүмкін емес.

Бағдарламалық қамтамасыз ету құрастырушыларын лицензиялау

Өнеркәсіптің кейбір салаларында қауіпсіздікке жауапты инженерлердің тиісті рұқсаттарының болуы талап етіледі. Тәжірибесіз, төмен білікті инженерлер қауіпсіздікті қамтамасыз етуге әдетте тартылмайды. АҚШ кейбір штаттарында бағдарламалық қамтамасыз етуді құрастырушыларды лицензиялау мәселесі кең талқыға алынғанына қарамастан, бұл талап қазіргі уақытта бағдарламалық қамтамасыз етуді құрастырушыларына таралмайды (Найт және Левесон, 2002). Алайда, болашақта қауіпсіз критикалық жүйелерді құрастыру стандарттарына құрастырушыларда біліктілік пен тәжірибенің минималды деңгейін бекітетін рұқсаттың болуы туралы тармақ енгізілуі мүмкін.

http://www.SoftwareEngineering-9.com/Web/DepSecAssur/Licensing.html

Бағдарламалық қамтамасыз етуді құрастырудың барлық сатыларында қауіпсіздікті қамтамасыз ету бойынша ерекше шаралар енгізілуі тиіс. Мұндай қауіпсіздікті қамтамасыз ету бойынша шаралар жүргізілген талдауларды тіркейді және талдауды жүргізуге жауапты тұлғаларды атайды. Бағдарламалық қамтамасыз етуді құрастырудың бір бөлігі болатын қауіпсіздікті қамтамасыз ету бойынша шаралар келесідей болады:

1. Мониторинг пен қауіп-қатерлерді тіркеу, олар арқылы қауіп-қатерді алдын ала талдаудан бастап, жүйені тестілеу мен валидацияға дейінгі қауіп-қатерлер бақыланады.
2. Құрастыру үдерісі барысында қолданылатын қауіпсіздікті тексерулер.
3. Қауіпсіздікті сертификаттау, оның барысында критикалық компоненттердің қауіпсіздігі ресми сертификатталады.

Ол үшін жүйені құрастыру үдерісіне қатыспаған мамандар тобы тартылады, бұл топ барлық дәлелдерді зерделеп, белгілі бір компонентті қолданысқа енгізер алдында қауіпсіз деп есептеуге болатын-болмайтынын анықтайды.

Аталған қауіпсіздікті қамтамасыз ету үдерістерінің мәліметтерін қолдау үшін жүйенің қауіпсіздік аспектілеріне жауапты құрастырушылар тағайындалуы керек. Бұл осы құрастырушылардың қауіпсіздіктің жеткіліксіздігімен байланысты жүйенің әр ақауы үшін жауапты болатынын білдіреді. Олар қауіпсіздік шараларының тиісті түрде орындалғанын дәлелдеуі тиіс. Қауіпсіздікке жауапты

Қауіп-қатерді есепке алу журналы					**4-бет: 20.02.2009 басылып шығарылды**
Жүйе: Инсулинді сорғы жүйесі *Қауіпсіздікке жауапты құрастырушы: Джеймс Браун*			*Файл: Инсулинды сорғы/Қауіпсіздік/ Қауіп-қатерді есепке алу журналы* *Журнал нұсқасы: 1/3*		
Анықталған қауіп-қатер	Емделушіге инсулиннің шектен тыс мөлшері енгізілген				
Анықталғаны	Джейн Уильямс				
Маңыздылық санаты	1				
Сәйкестендірілген тәуекел	Жоғары				
Қателіктер тізбегінің сәйкестендірілуі	ИӘ	Күні	24.01.07	Орны	Қауіп-қатерді есепке алу журналы, 5 бет
Қателіктер тізбегін құрастырғандар	Джейн Уильямс пен Билл Смит				
Қателіктер тізбегін тексеру	ИӘ	Күні	28.01.07	Тексеруші	Джеймс Браун

Қауіпсіздік жүйесіне жобалық талаптар

1. Жүйенің сенсорлық жүйені, сағат пен инсулин жіберу жүйесін тексеретін өзін-өзі бақылайтын бағдарламалық қамтамасыз етуі болуы тиіс.
2. Өзін-өзі бақылайтын бағдарламалық қамтамасыз ету өз міндетін әр минут сайын орындауы тиіс.
3. Өзін-өзі бақылайтын бағдарламалық қамтамасыз ету жүйенің кез келген компонентінің ақауын анықтаған жағдайда, ол дыбыс сигналын беруі тиіс, сорғы экранында ақау анықталған коспоненттің атауы шығуы тиіс. Инсулиннің жіберілуі тоқтатылуы тиіс.
4. Жүйенің қолданушыға жіберілуге тиіс есептелген инсулин мөлшерін өзгертуге мүмкіндік беретін апаттық басқару жүйесі болуы тиіс.
5. Түзетілген мән жүйені конфигурациялау кезінде медициналық қызметкер бекіткен мәннен жоғары болмауы тиіс (максималды түзетілген мәні).

15.6-сурет. Қауіп-қатерді есепке алу журналындағы жеңілдетілген жазба

құрастырушылар қауіпсіздікті қамтамасыз ету бойынша барлық құжаттарды қадағалау үшін конфигурациялық басқарудың тиісті жүйесінің қолданылуын, сондай-ақ, тиісті техникалық құжаттамалардың талаптарын сақтау үшін сапаны бақылау қызмет менеджерлерімен бірлесе жұмыс істейді. Бұл барлық сенімді үдерістер үшін маңызды. Клиентке конфигурациялық басқару жүйесінің қателіктері себебінен дұрыс емес жүйе ұсынылған жағдайда валидацияның қатаң процедураларын сақтаудың мәні жоқ. Конфигурация мен сапаны басқару мәселесі 24 және 25-тарауларда толық сипатталған. Қауіпсіз критикалық жүйені құрастыру үдерістерінің негізгі бөлігі болып табылатын қауіп-қатер талдауының үдерісі қауіпсіздікті қамтамасыз ету үдерісінің мысалы болады. Қауіп-қатерді талдау қауіп-қатерді сәйкестендіруден, олардың пайда болу мүмкіндігінің дәрежесінен, әр бір белгілі қауіп-қатер апатқа әкелуінен тұрады. Егер әр қауіп-қатерді тексеріп, жоя-

тын бағдарламалық код болса, сіз бұл қауіп-қатерлер апатқа әкелмейтінін дәлелдей аласыз. Мұндай дәлелдер осы тарауда сөз болатын қауіпсіздік дәлелдерімен толыға алады. Егер пайдалану алдында жүйе сертификация рәсімінен өтуі қажет болса (мысалы, авиациялық жүйе), сертификацияның стандартты шарттарының бірі трассалану болып табылады.

Міндетті түрде басталуы тиіс негізгі қаіпсіздік құжаты – бұл қауіп-қатерді есепке алу журналы. Бұл құжат бағдарламалық қамтамасыз етуді құрастыру үдерісінде анықталған қауіп-қатерлердің есепке алынуы туралы ақпарат ұсынады. Қауіп-қатерлерді есепке алу журналы анықталған қауіп-қатерді тіркеу үшін бағдарламалық қамтамасыз ету үдерісінің әр сатысында қолданылуы тиіс. Инсулин жіберу жүйесінің қауіп-қатерлерін есепке алу журналындағы жазбасының жеңілдетілген мысалы 15.6-суретте берілген. Бұл нысан қауіп-қатерлерді талдау үдерісін құжаттамалық бекітіп, бұл үдерістің барысында жасалған құрылым талаптарын көрсетеді. Бұл құрылым талаптарының басты мақсаты – бақылау жүйесі инсулин сорғысын қолданушыға инсулиннің шектен асқан мөлшерін енгізбеу кепілі.

15.6-суретте көрсетілгендей, жүйе қауіпсіздігіне жауапты тұлғалар нақты анықталуы тиіс. Бұл екі себеп бойынша маңызды:

1. Егер бұл тұлғалар анықталса, олар өз әрекеттері үшін жауапқа тартылуы мүмкін. Бұл құрастыру кезінде олардың мұқият болатынын білдіреді, себебі кез келген мәселе анықталып, олардың кінәсі дәлелденеді.
2. Апат жағдайында сот қарауы немесе тергеу жүргізілуі мүмкін. Қауіпсіздікті қамтамасыз етуге жауаптыны анықтаудың маңызы зор, себебі олар әрекеттері мен қателіктері үшін жауапқа тартылады.

15.5 Функционалдық сенімділік пен қауіпсіздік бойынша есептер

Қауіпсіздік пен функционалдық сенімділікті қамтамасыз ету үдерісінде көп мөлшерде ақпарат өндіріледі: сынақ нәтижелері, құрастырудың қолданылған үдерістері туралы ақпарат, шолу кеңестерінің хаттамалары және т.б. бұл ақпарат жүйенің қауіпсіздігі мен функционалдық сенімділігін қамтамасыз ету дәлелдерін қамтамасыз етеді және қолдану үшін құрастырылған жүйенің жеткілікті түрде сенімділігі туралы шешім қабылдау үшін қолданылады.

Қауіпсіздік пен функционалдық сенімділікті қамтамасыз ету бойынша есептер жүйенің қауіпсіз болуы немесе қауіпсіздік пен фунционалдық сенімділіктің талап етілген талаптарын қамтамасыз етуі туралы дәлелдер келтіретін құрамдастырылған құжаттардан тұрады. Кейде олар қамтамасыз ету есептері деп аталады. Іс жүзінде, қауіпсіздік пен функционалдық сенімділікті қамтамасыз ету бойынша есеп жүйенің сенімділігін дәлелдейтін барлық қолжетімді ақпаратты біріктіреді. Критикалық жүйелердің көп түрі үшін қауіпсіздік есебін ұсыну заңды талап болып табылады.

Мұндай есеп жүйе қолданысқа енгізілмес бұрын, бақылау немесе сертификаттау органымен құпталуы тиіс.

Бақылау органы қауіпсіздік пен функционалдық сенімділік параметрлерінің іс жүзінде мүмкін болатын параметрлерге сәйкестігін тексеруге жауапты, сондықтан олар өз міндеттерін жоба құрастырылуы аяқталған соң бастайды. Алайда құрылымдар мен бақылау органдарының бір-бірінен тәуелсіз жұмыс істеуі сирек болады; олар әдетте қауіпсіздік есебіне енгізілетін ақпаратты анықтау үшін құрастырушылар командасымен келіссөздер жүргізеді. Бақылау органы мен құрастырушылар үдерістер мен рәсімдерді, олардың қанағаттанарлықтай құжатталғанына көз жеткізу үшін бірлесіп зерделйді.

Функционалдық сенімділік есептері әдетте жүйені құрастыру үдерісінде және ол аяқталған соң жасалады. Кейде құрастыру үдерісінде жүйенің функционалдық сенімділігі жөніндегі дәлелдер алынбаса, бұл мәселе туғызуы мүмкін. Грейдон және басқалардың (2007) пікірінше қауіпсіздік пен функционалдық сенімділікті қамтамасыз ету бойынша есепті әзірлеу жүйе жобасын жасау және жүзеге асырумен тығыз байланыста болуы тиіс. Бұл жүйе жобасы бойынша шешімдерге функционалдық сенімділік есептерінің талаптары әсер етуі мүмкін. Есеп дайындауда қосымша қиындықтар мен шығындарға әкелетін жобалық шешімдерді қабылдаудан бас тартуға болады. Функционалдық сенімділік есептері жүйе қауіпсіздігі есептерінің жалпыламасы болып табылады.

Қауіпсіздік есебі – бұл сертификатталатын жүйенің сипаттамасы, жүйені құрастыруда пайдаланылған үдерістер, алдымен жүйенің ықтимал қауіпсіздігін дәлелдейтін дәлелдерден тұратын құжаттар жиынтығы. Бишоп пен Блумфилд (1998) қауіпсіздік есебінің қысқаша анықтамасын ұсынады:

> *Белгілі бір ортада қолдану әдісін анықтау үшін жүйенің адекватты қауіпсіздігіне шынайы және сенімді дәлелдер келтіретін құжатталған дәлелдер жиынтығы.*

Қауіпсіздік немесе функционалдық сенімділік есебінің құрылымы мен мазмұны сертификатталатын жүйе типі мен қызмет мазмұнының типіне байланысты. 15.7-суретте қауіпсіздік есебі құрылымының бір мысалы келтірілген, алайда қауіпсіздік есебінің құрылымы үшін таралған өндірістік стандарттар жоқ. Қауіпсіздік есептерінің құрылымы өнеркәсіп саласы мен белгілі бір саласының даму деңгейіне байланысты өзгереді. Мысалы, атом энергетикасы кәсіпорын жүйелері үшін қауіпсіздік есептері көп жылдар бойы талап етілген. Мұндай есептер атом энергетикасы саласында жұмыс істейтін инженерлердің артықшылықтарына негізделген күрделі нысанға ие. Алайда, медициналық құрылғылар жүйелерінің қауіпсіздік есептерін әзірлеу қажеттілігі кейінірек пайда болды. Олардың құрылымы икемді болып келеді, ал есептердің өздері атом өнеркәсібі жүйелерінің есептерімен салыстырғанда дәлдік дәрежесі аз.

Қауіп тек оны ірі компьютерлік немесе әлеуметтік-технологиялық жүйеге енгізуден кейін пайда болады, себебі оның бұзылуы құрылғының шағылуына немесе жарақат пен өлімге соқтыруы мүмкін. Жүйе қауіпсіздігі туралы есеп

әзірлегенде Сізге бағдарламалық қамтамасыз ету ақауларын бүкіл жүйенің ірі көлемді ақауларымен салыстырып, олардың бір-біріне тәуелсіздігін дәлелдеу қажет.

Бөлім	Сипаттама
Жүйе сипаттамасы	Кризистік компоненттердің жүйелер сипаттамасына қысқаша шолу.
Қауіпсіздік талаптары	Жүйелік талаптардан спецификациядан алған қауіпсіздік бойынша талаптар. Сонымен бірге басқа маңызды жүйелік талаптарды қосуға болады.
Қауіп-қатер мен тәуекелдер талдауы	Анықталған қауіп-қатерлер мен тәуекелдерді, сондай-ақ, қауіп-қатерді азайтуға қабылданған шараларды сипаттайтын құжаттар. Қауіп-қатерді талдау нәтижелері және қауіп-қатерді есепке алу журналдары.
Құрылымды талдау	Құрылымның қауіпсіздігін дәлелдейтін құрамдастырылған дәлелдер жиынтығы (15.5.1 бөлімді қара).
Верификация мен валидация	B және B қолданылған рәсімдер сипаттамасы, қажет болған жағдайда жүйені тестілеу жоспарлары. Анықталған және жойылған ақауларды көрсететін тестілеу нәтижелерін қысқаша шолу. Егер формалды әдістер қолданса, жүйенің формалды спецификациясы мен сол спецификацияның талдаулары кез келген болады. Бастапқы код статикалық талдауының хаттамалары.
Тексеру есептері	Құрылым мен қауіпсіздікті тексерудің барлық хаттамалары.
Команданың құзыреттілігі	Жүйені құрастыру мен валидация үдерісіне қатысатын барлық команда мүшелерінің құзыреттілігін дәлелдеу.
Сапаны қамтамасыз ету үдерістері	Жүйені құрастыру үдерісінде орындалған сапаны қамтамасыз ету үдерістерінің хаттамалары (24 тарауды қара).
Өзгерістерді бақылау үдерістері	Барлық ұсынылған өзгерістер, қабылданған шаралар және қажет болған жағдайда сол өзгерістердің қауіпсіздігін дәлелдеу хаттамалары.
Қауіпсіздік туралы қосымша есептер	Аталған қауіпсіздік есебіне қатысты қауіпсіздіктің басқа есептеріне сілтемелер.

15.7-сурет. Бағдарламалық қамтамасыз ету қауіпсіздігі туралы есептің мазмұны

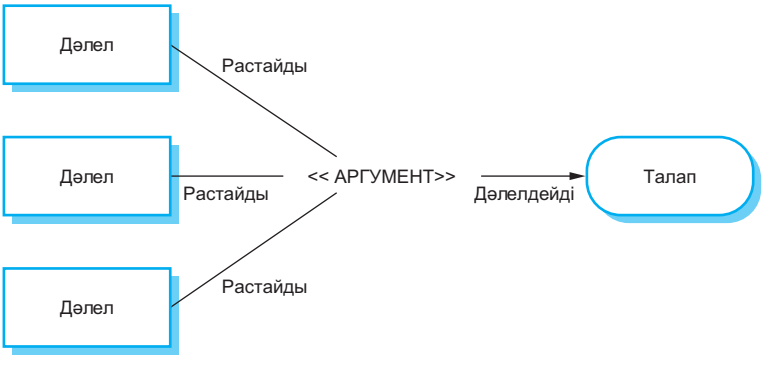
15.5.1 Құрамдастырылған дәлелдер

Жүйенің қолданысқа енгізілуі үшін жеткілікті түрде сенімді болатыны туралы шешім логикалық дәлелдерге негізделуі тиіс. Олар ұсынылған дәлелдердің жүйенің қауіпсіздік талаптары мен функционалдық сенімділікке сәйкестігін растайтынын көрсетуі қажет. Бұл талаптар абсолютті (X оқиғасы болады немесе болмайды) немесе ықтималды (Y оқиғасының болу мүмкіндігі 0,n тең).

Аргумент талаптар мен дәлелдер арасындағы байланыстырушы түйін болып табылады. 15.8-суретте көрсетілгендей аргумент талаптар мен жиналған дәлелдер жиынтығын байланыстырады. Аргумент неліктен қолжетімді дәлелдер негізінде талаптың жүйе қауіпсіздігі немесе функционалдық сенімділігі шарттары болатынын түсіндіреді.

Мысалы, инсулин сорғысы қауіпсіздік критикалық параметр болатын құрылғы болып табылады, оның бұзылуы қолданушы денсаулығына зиян келтіреді. Бұл көп елдерде бақылау органының (мысалы, Біріккен Корольдіктегі Медициналық құралдар мәселесі бойынша басқарма) құрылғы сатылып, қолданылмас бұрын жүйенің қауіпсіздігінде толық сенімді болу қажеттілігін білдіреді. Бұл шешімді қабылдау үшін бақылау органы стандартты қолдану кезінде жүйе қолданушы денсаулығына зиян келтірмейтінін дәлелдейтін құрамдастырылған аргументтерден тұратын жүйенің қауіпсіздігі туралы есепті бағалайды.

Қауіпсіздік есептері әдетте талаптарға сәйкестігін растайтын құрамдастырылған аргументтерге негізделеді. Мысалы, әрі қарай ұсынылған аргумент арқылы бағдарламалық қамтамасыз етудің инсулиннің шектен асқан мөлшерінің енгізілмейтініне кепіл беретін есептермен қамтамасыз ету талаптарының сақталғанын дәлелдеуге болады. Бұл аргумент әрине жеңілдетілген. Шынайы қауіпсіздік есебінде нақты дәлелдер келтірілер еді.



15.8-сурет. Құрылымданған аргументтер

*Талап: Б*елгілі бір емделушіге қауіпсіз деп белгіленген максималды мөлшер болған жағдайда, инсулин сорғысы есептеген максималды бірлік мөлшері сол максималды мөлшерден аспайды.

Дәлел: Инсулин сорғысы бағдарламалық қамтамасыз етудің бақылау бағдарламасы үшін қауіпсіздік аргументі (қауіпсіздік аргументтері әрі қарай осы тарауда сипатталатын болады).
Дәлел: Инсулин сорғысын тестілеу мәліметтерінің кешені. 400 сынақ өткізу нәтижесінде енгізілген мөлшер дұрыс есептеліп, бір де бір рет максималды мөлшерден асқан жоқ.
Дәлел: Инсулин сорғысы бақылау бағдарламасының статикалық талдау есебі. Бақылау бағдарламалық қамтамасыз етудің статикалық талдауы енгізілетін мөлшерге әсер ететін ешқандай аномалиялар, инсулин мөлшерін енгізу мәніне жауапты ауыспалы бағдарламаны анықтаған жоқ.
Аргумент: Ұсынылған дәлелдер инсулиннің есептелген максималды мөлшері максималды мөлшер мәніне тең болатынын көрсетеді.
Осылайша, біз сенімділікпен дәлелдердің талапты сақтау фактісін дәлелдейтінін болжай аламыз: инсулин сорғысымен есептелген емделушіге енгізілетін инсулин мөлшері максималды бірлік мөлшер мәнінен аспайды.

Ұсынылған дәлелдердің көп санды және әр түрлі болатынына назар аударыңыз. Бағдарламалық қамтамасыз ету бір-бірімен сәйкес келетін бірнеше әр түрлі механизммен тексеріледі. 13-тарауда аталғандай, көп санды және әр түрлі үдерістерді қолдану сенімділік дәрежесін арттырады. Егер валидацияның бір үдерісі нәтижесінде қателіктер мен кемшіліктер анықталмаса, олардың валидацияның басқа үдерістерінде анықталу ықтималдығы бар.

15.9-сурет. Инсулинді сорғы үшін қауіпсіздік талаптарының иерархиясы

Жүйенің функционалдық сенімділігі мен қауіпсіздігіне әдетте талаптардың көп саны қойылады, мұнда бір талаптың сақталуы жасқа талаптардың сақталуына

байланысты болады. Осылайша, талаптар иерархия түрінде ұйымдастырылуы мүмкін. 15.9-суретте инсулин сорғысына арналған ұқсас иерархия бөлігі ұсынылған. Жоғары деңгейдегі талаптың сақталуын көрсету үшін Сізге алдымен төменгі деңгейдегі талаптар аргументтерімен жұмыс істеу қажет. Егер Сіз төменгі деңгейдегі талаптардың сақталғанын көрсетсеңіз, жоғары деңгей талаптарының орындалғаны туралы қорытындыға келуге болады.

15.5.2 Қауіпсіздіктің құрамдастырылған аргументтері

Қауіпсіздіктің құрамдастырылған аргументтері – бұл бағдарламаның қауіпсіздік талаптарына жауап беретінін дәлелдейтін құрамдастырылған аргументтердің бір түрі. Қауіпсіздік аргументтері жағдайында бағдарламаның тиісті түрде істейтінін дәлелдеу қажет емес. Тек бағдарламаны орындау қауіп жағдайына әкелмейтінін көрсету қажет. Бұл қауіпсіздік аргументтерін анықтау дұрыстық аргументтерін анықтаудан арзан болатынын білдіреді. Сізге бағдарлама жағдайын зерделеу қажет емес, тек апатқа соқтыратын жағдайларға мән берген жеткілікті.

Жүйе қауіпсіздігінің негізіндегі жалпы долбар критикалық қауіпті қауіп-қатерлерден туатын жүйенің ақаулары саны жүйедегі ақаулардың жалпы санынан аз болатындығына негізделген. Қауіпсіздікті қамтамасыз ету барысында басты назар критикалық қауіп-қатерге соқтыратын ақауларға ғана аударылады. Егер мұндай ақаулардың болмайтынын немесе олар пайда болғанда апатқа соқтыратын қауіп-қатер төнбейтінін дәлелдей алсаңыз, жүйе қауіпсіз болады. Бұл қауіпсіздіктің құрамдастырылған аргументтерінің негізі.

Қауіпсіздіктің құрамдастырылған аргументтерінің мақсаты қалыпты шарттарда бағдарлама қауіпсіз болатынын көрсету. Әдетте олар қайшылықтарға негізделген. Әрі қарай қауіпсіздік аргументтерін құру сатылары ұсынылған:

1. Алдымен бағдарламаны орындау кезінде жүйе қауіп-қатерлерін талдау арқылы сәйкестендіруге болатын қауіпті жағдайдың туу мүмкіндігін түсінесіз.
2. Сіз осы қауіпті жағдайды анықтайтын предикатты (логикалық шарт) жазып аласыз.
3. Содан соң Сіз жүйе үлгісі мен бағдарламаны жүйелі талдап, соңғы жағдай қауіпті жағдай предикатына қайшы келетінін дәлелдейсіз. Бұл жағдайда бастапқы қауіптілік жағдайы түсінігі бұрыс болады.
4. Барлық сәйкестендірілген қауіп-қатерлер үшін осы талдауды қайталап, Сіз жүйе қауіпсіздігінің шынайы дәлелдерін аласыз.

Қауіпсіздіктің құрамдастырылған аргументтері талаптардан бастап, құрылым үлгілері мен кодқа дейінгі әр түрлі деңгейлерде қолданылады. Талаптар деңгейінде Сіз қауіпсіздіктің талаптары жіберілмегенін және талаптар жүйеге қатысты жарамсыз долбарлар жасамайтынын дәлелдейсіз.

Сіз құрылым деңгейінде қауіпті жағдайларды анықтау мақсатында жүйенің жағдай үлгісін талдай аласыз. Код деңгейінде барлық тораптарда қайшылықтар болатынын көрсету үшін сол тораптарды зерделейсіз.

Мысал ретінде, 15.10-суретте келтірілген кодпен танысыңыз, бұл инсулинді жіберу жүйесін жүзеге асырудың бір бөлігі болады. Код енгізілген инсулин мөлшері есептейді, содан соң инсулиннің шектен тыс мөлшері енгізілу мүмкіндігін азайтатын қауіпсіздікті тексеру жүзеге асырылады. Осы кодқа қауіпсіздік аргументін құру үшін инсулиннің енгізілетін мөлшері бірлік мөлшердегі максималды қауіпсіздік деңгейінен аспайтынын дәлелдеу қажет. Ол қант сусамыры бар әр қолданушы үшін емдеуші дәрігердің ұсынымдары негізінде бекітіледі.

Қауіпсіздікті дәлелдеу үшін жүйенің «дұрыс» мөлшерді енгізетінін дәлелдеу қажет емес, емделушіге артық мөлшерде енгізілмейтінін дәлелдеу жеткілікті. Жүйені қолданушы үшін максималды мөлшер қауіпсіз болатыны туралы пікірге сүйенесіз.

— Инсулиннің енгізілетін мөлшері алдыңғы енгізілген мөлшер мен уақытының қандағы қант деңгейінің функциясы болады

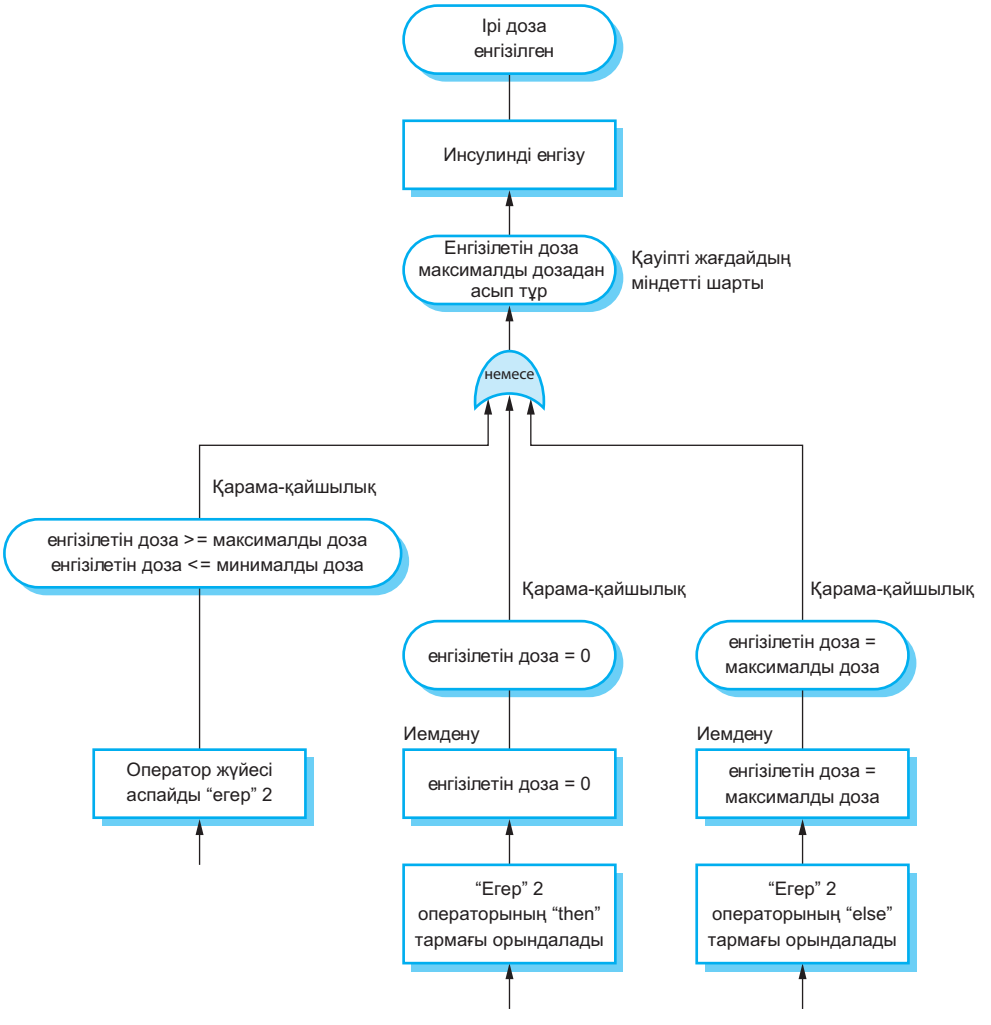
```
// Safety check–adjust currentDose if necessary.
// if statement 1
if (previousDose == 0)
{
        if (currentDose > maxDose/2)
        currentDose = maxDose/2 ;
}
else
        if (currentDose > (previousDose * 2) )
           currentDose = previousDose * 2 ;
// if statement 2
if ( currentDose < minimumDose )
    currentDose = 0 ;
else if ( currentDose > maxDose )
     currentDose = maxDose ;
administerInsulin (currentDose) ;
```

15.10-сурет. Инсулин мөлшері мен қауіпсіздікті тексеру есебі

Қауіпсіздік аргументін жасау үшін Сіз қауіпті жағдайды анықтайтын предикатты сәйкестендіресіз, яғни енгізілетін мөлшер максималды мөлшерден асады (currentDose > maxDose). Содан соң Сіз бағдарламаның барлық тораптары осы

қауіпті жағдай шартына қайшы келетінін дәлелдейсіз. Бұл жағдайда қауіпті жағдай шарты шынайы болуы мүмкін емес.

Егер мұны істей алсаңыз, бағдарлама қауіпті инсулин мөлшерін есептемейтініне сенімді болыңыз. Сіз қауіпсіздік аргументтерін 15.11-суретте көрсетілгендей сұлбалар түрінде көрсете аласыз.



15.11-сурет. Қайшылықтарды көрсетуге негізделген қауіпсіздіктің формалды емес аргументі

15.11-суретте көрсетілген қауіпсіздік аргументі жағдайында «инсулинді енгізу» командасын орындауға әкелетін үш жол бар. Сізге енгізілетін инсулин мөлшерінің максималды мөлшерден аспайтынын дәлелдеу қажет.

Бағдарлама қауіпті есептер жасамайтынын дәлелдейтін қауіпсіздік аргументін құру үшін ықтимал қауіпті жағдайға соқтыратын барлық жолдарды алдымен

сәйкестендіру қажет. Сіз сол қауіпті жағдайдан кері қарай қозғалуыңыз қажет, қауіпті жағдайды тудыратын жолдардағы ауыспалы жағдайлар үшін соңғысын есепке алуыңыз қажет. Егер Сіз ауыспалылардың ешқайсысы қауіпті болмайтынын дәлелдей алсаңыз, Сіздің бастапқы долбарыңыз (есеп қауіпті болуы мүмкін) бұрыс болғанын дәлелдей аласыз.

Қарсы бағыттағы қозғалыс кодқа шығу жағдайларының тууына соқтыратын соңғы жағдайлардан басқа барлық жағдайларды елемеуге мүмкіндік береді. Алдыңғы мәндер жүйенің қауіпсіздігіне ешбір әсер етпейді. Бұл мысал жағдайында, Сізге «инсулинды енгізу» командасын орындау алдында енгізілетін мөлшер мәніне ғана назар аудару қажет. Сіз 15.10-суреттегі «егер» операторы жағдайындағы тәрізді есептерді елемей қоя аласыз, себебі олардың нәтижелері бағдарламаны келесі операторларымен алмастырылады.

«Инсулинді енгізу» командасын орындаудың барлық мүмкін жолдары талданды:

1. «егер» 2 операторының ешбір тармағы орындалмайды. Бұл енгізілетін мөлшер минималды мөлшерден артық не тең, максималды мөлшерден кем не тең болған жағдайда ғана орын алуы мүмкін. Бұл соңғы шарт – оператор орындаған соң жарамды болатынын пайымдау.
2. «егер» 2 операторының «then» тармағы орындалады. Бұл жағдайда енгізілетін мөлшерге нөл мәні беріледі. Осылайша соңғы шарт «енгізілетін мөлшер = 0» болады.
3. «егер» 2 операторының «else» тармағы орындалады. Бұл жағдайда енгізілетін мөлшерге максималды мөлшер мәні беріледі. Осылайша, соңғы шарт «соңғы мөлшер = максималды мөлшер» болады.

Барлық үш жағдайда соңғы шарттар қауіпті жағдайдың міндетті шарттарына қайшы келеді: енгізілетін мөлшер максималды мөлшерден асады. Осылайша, жүйе есебі қауіпсіз болады деп пайымдай аламыз.

Құрамдастырылған аргументтер жүйенің белгілі бір қасиеттері шынайы болатынын көрсету үшін қолданылады. Егер Сіз, мысалы, жүйенің есебі қорларға рұқсаттың өзгеруіне әкелмейтінін дәлелдегіңіз келсе, қауіпсіздіктің құрамдастырылған аргументін қолдана аласыз. Алайда, құрамдастырылған аргументтерге негізделген дәлелдер қауіпсіздік валидациясына қарағанда сенімсіздеу болады. Бұл бұзушының жүйе кодын бұзу мүмкіндігімен байланысты. Бұл жағдайда қауіпсіздігі дәлелденген орындалған код күшін жояды.

НЕГІЗГІ ТҰЖЫРЫМДАР

- Статикалық талдау – бұл қателіктер мен ақауларды анықтау мақсатында жүйенің бастапқы кодын (немесе басқа көрінісі) тексеруге негізделген ве-

рификация және валидация әдісі. Бұл жүйені тестілеу кезінде ғана емес, бағдарламаның барлық бөліктерін тексеруге мүмкіндік береді.

- Үлгіні тексеру – бұл ықтимал қателердің болуын анықтау мақсатында жүйенің барлық жағдайларын толықтай тексеруге негізделген статикалық талдаудың формалды әдісі.
- Статистикалық тестілеу бағдарламалық қамтамасыз ету сенімділігін бағалау үшін қолданылады. Бұл әдіс бағдарламалық қамтамасыз етудің функционалдық бөлігін көрсететін тестілеу жиынтығының көмегімен жүйені тестілеуге негізделген.
- Қауіпсіздік валидациясы күрделі міндеті, себебі қауіпсіздік талаптары жүйенің нені жүзеге асыруы тиістігін көрсетеді. Сондай-ақ, жүйені бұзушылар білімді және қауіпсіздікті тестілейтін мамандарға қарағанда, жүйе кемшіліктерін анықтауға уақыттары көбірек болады.
- Қауіпсіздік валидациясы тәжірибе негізіндегі талдау, аспаптық талдау немесе жүйе шабуылын сылтауратын «шабуыл командаларының» көмегімен жүзеге асырылады.
- Қауіпсіздік критикалық параметр болатын Жүйе құрылысының нақты анықталған, сертификатталған үдерістерін басқарған маңызды. Үдеріске сәйкестендіру мен ықтимал қауіп-қатердің мониторингі енуі тиіс.
- Қауіпсіздік пен фунционалдық сенімділік бойынша есептерге жүйенің қауіпсіздігі мен фунционалдық сенімділігін көрсететін дәлелдер жатады. Қауіпсіздік есептері қолдану алдында сыртқы бақылау органы сертификатталған жүйе болу тиістілігі кезінде қажет болады.
- Қауіпсіздік есептері құрамдастырылған аргументтерге негізделген. Қауіпсіздіктің құрамдастырылған аргументтері қауіпті жағдайлардың пайда болуына және бұл жағдайдың пайда болу мүмкіндігі болмауына әкелетін барлық бағдарламалық жолдарды есепке алу тәсілімен сәйкестендірілген қауіпті шарттың болмайтынын дәлелдейді.

ҚОСЫМША ӘДЕБИЕТТЕР

Software Reliability Engineering: More Reliable Software, Faster and Cheaper, 2nd edition. This is probably the definitive book on the use of operational profiles and reliability models for reliability assessment. It includes details of experiences with statistical testing. (J. D. Musa, McGraw-Hill, 2004.)

'NASA's Mission Reliable'. A discussion of how NASA has used static analysis and model checking for assuring the reliability of spacecraft software. (P. Regan and S. Hamilton, *IEEE Computer*, **37** (1), January 2004.) http://dx.doi.org/10.1109/MC.2004.1260727.

Dependability cases. An example-based introduction to defining a dependability case. (C. B. Weinstock, J. B. Goodenough, J. J. Hudak, Software Engineering Institute, CMU/SEI-2004-TN-016, 2004.) http://www.sei.cmu.edu/publications/documents/04.reports/04tn016.html.

How to Break Web Software: Functional and Security Testing of Web Applications and Web Services. A short book that provides good practical advice on how to run security tests on networked applications. (M. Andrews and J. A. Whittaker, Addison-Wesley, 2006.)

'Using static analysis to find bugs'. This paper describes Findbugs, a Java static analyzer that uses simple techniques to find potential security violations and runtime errors. (N. Ayewah et al., *IEEE Software*, **25** (5), Sept/Oct 2008.) http://dx.doi.org/10.1109/MS.2008.130.

ЖАТТЫҒУЛАР

15.1. Қандай жағдайларда бағдарламалық қамтамасыз етудің критикалық жүйесін құрастыру кезінде формалды спецификация мен верификацияны қолдану тиімді болатынын түсіндіріңіз. Сіздің ойыңызша критиклық жүйе құрастырушылары формалды әдістерді қолдануға неге қарсы шығады?

15.2. Сіз қолданатын Java, C++ немесе басқа тілдер статикалық анализаторларының көмегімен анықталуы мүмкін шарттардың тізімін жасаңыз. Өз тізіміңізді 15.20-суреттегі тізіммен салыстырып, нәтижесін түсіндіріңіз.

15.3. Жүйенің қызмет көрсету мерзімінде ақаулардың шектеулі саны болған жағдайда, сенімділік спецификациясының валидациясы неге мүмкін болмайтынын түсіндіріңіз.

15.4. Жүйенің сенімділігін қамтамасыз ету неге жүйе қауіпсіздігінің кепілі болмайтынын түсіндіріңіз.

15.5. Мысалдар жүзінде қауіпсіздікті тестілеу неге күрделі үдеріс болатынын түсіндіріңіз.

15.6. Сіз құрастыратын қосымша үшін пароль арқылы қорғау жүйесінің валидациясын қалай жүзеге асырасыз? Сіз ұсынып отырған әр аспаптың қызметі туралы айтып беріңіз.

15.7. Медициналық мекемелердің мәліметтер базасының бағдарламалық қамтамасыз етуін нәтижесінде емделуші туралы құпия мәлімет жария болатын бұзудан сақтау қажет. Шабуылдың кейбір түрлері 14-тарауда сипатталған. Осы ақпаратты қолдана отырып, бағдарламалық қамтамасыз етудің қауіпсіздігін тестілеу бойынша мамандарға ұсынымдар беру үшін 15.5-суретте келтірілген бақылау тізімін кеңейтіңіз.

15.8. Бағдарламалық қамтамасыз етудің қауіпсіздік есебі қажет болатын жүйенің төрт түрін атаңыз, оларға неге мұндай талаптардың қойылатынын түсіндіріңіз.

15.9. Қызмет қауіпсіздігін қамтамасыз ету үшін ядролық қалдықтарды сақтауға арналып жасалған үй-жай есігінің блоктануын бақылау механизмі. Ол радиациядан қорғау экрандары қолданылғанда немесе үй-жайдағы радиация деңгейі берілген мәннен төмен болғанда (dangerLevel) ғана үй-жайға кіру рұқсатын береді. Осылайша:

i. Үй-жай ішінде радиациядан қорғайтын қашықтықтан басқару экрандары болған жағдайда, авторластырылған оператор есікті аша алады.

ii. Үй-жай ішінде радиация деңгейі белгілі бір мәннен төмен болғанда, авторластырылған оператор есікті аша алады.

```
1     entryCode = lock.getEntryCode () ;
2     if (entryCode == lock.authorizedCode)
3     {
4               shieldStatus = Shield.getStatus ();
5               radiationLevel = RadSensor.get ();
6               if (radiationLevel < dangerLevel)
7                         state = safe;
8               else
9                         state = unsafe;
10    if (shieldStatus == Shield.inPlace() )
11                        state = safe;
12    if (state == safe)
13                        {
14                                  Door.locked = false ;
15                                  Door.unlock ();
16                        }
17              else
18              {
19                        Door.lock ( );
20                        Door.locked := true ;
21              }
22    }
```

15.12-сурет. Есіктің рұқсат коды

iii. Оператордың кіру құқығы арнайы рұқсат кодын енгізу арқылы анықталады. 15.12-суретте ұсынылған код (әрі қарай қара) есіктің блоктаушы механизмін бақылайды. Қауіпсіз жағдайда есікті ашуға тыйым салынғанына назар аударыңыз. 15.5.2-бөлімде сипатталған тәсілді қолдана отырып, осы код үшін қауіпсіздік аргументін жасаңыз.

Белгілі бір операторларды көрсету үшін жол нөмірін қолданыңыз. Кодтың қауіпсіздігіне күмәніңіз туса, қауіпсіздікті қамтамасыз ету мақсатында оны түрлендіру бойынша ұсыныстарыңызды енгізіңіз.

15.10. Сіз өзіңізді бұзылған, нәтижесінде қоршаған ортаның ластануына соқтырған химиялық зауыт үшін бағдарламалық қамтамасыз етуді құрастырған команда мүшесі деп есептеңіз. Сіздің басшыңыз теледидар бағдарламасы арқылы ақаулардың жоғы туралы сұхбат берді. Оның сөзінше апат зауыт қызметкерлерінің тиісті жұмыс істемеуінен болған. Газет тілшілері Сіздің пікіріңізді сұрауда. Сіз өзіңізді мұндай сұхбатта қалай ұстайтыңызды айтып беріңіз.

ПАЙДАЛАНЫЛҒАН ӘДЕБИЕТТЕР

Abrial, J. R. (2005). *The B Book: Assigning Programs to Meanings*. Cambridge, UK: Cambridge University Press.

Anderson, R. (2001). *Security Engineering: A Guide to Building Dependable Distributed Systems*. Chichester, UK: John Wiley & Sons.

Baier, C. and Katoen, J.-P. (2008). *Principles of Model Checking*. Cambridge, Mass.: MIT Press.

Ball, T., Bounimova, E., Cook, B., Levin, V., Lichtenberg, J., McGarvey, C., Ondrusek, B., S. K., R. and Ustuner, A. (2006). 'Thorough Static Analysis of Device Drivers'. *Proc. EuroSys* 2006, Leuven, Belgium.

Bishop, P. and Bloomfield, R. E. (1998). 'A methodology for safety case development'. *Proc. Safety-critical Systems Symposium*, Birmingham, UK: Springer.

Chandra, S., Godefroid, P. and Palm, C. (2002). 'Software model checking in practice: An industrial case study'. *Proc. 24th Int. Conf. on Software Eng.* (ICSE 2002), Orland, Fla.: IEEE Computer Society, 431–41.

Croxford, M. and Sutton, J. (2006). 'Breaking Through the V and V Bottleneck'. *Proc. 2nd Int. Eurospace—Ada-Europe Symposium on Ada in Europe*, Frankfurt, Germany: Springer-LNCS, 344–54.

Evans, D. and Larochelle, D. (2002). 'Improving Security Using Extensible Lightweight Static Analysis'. *IEEE Software*, **19** (1), 42–51.

Graydon, P. J., Knight, J. C. and Strunk, E. A. (2007). 'Assurance Based Development of Critical Systems'. *Proc. 37th Annual IEEE Conf. on Dependable Systems and Networks*, Edinburgh, Scotland: 347–57.

Holzmann, G. J. (2003). *The SPIN Model Checker*. Boston: Addison-Wesley.

Knight, J. C. and Leveson, N. G. (2002). 'Should software engineers be licensed?' *Comm.* ACM, **45** (11), 87–90.

Larus, J. R., Ball, T., Das, M., Deline, R., Fahndrich, M., Pincus, J., Rajamani, S. K. and Venkatapathy, R. (2003). 'Righting Software'. *IEEE Software*, **21** (3), 92–100.

Musa, J. D. (1998). *Software Reliability Engineering: More Reliable Software, Faster Development and Testing*. New York: McGraw-Hill.

Nguyen, T. and Ourghanlian, A. (2003). 'Dependability assessment of safety-critical system software by static analysis methods'. *Proc. IEEE Conf. on Dependable Systems and Networks (DSN'2003)*, San Francisco, Calif.: IEEE Computer Society, 75–9.

Pfleeger, C. P. and Pfleeger, S. L. (2007). *Security in Computing, 4th edition*. Boston: Addison-Wesley.

Prowell, S. J., Trammell, C. J., Linger, R. C. and Poore, J. H. (1999). *Cleanroom Software Engineering: Technology and Process*. Reading, Mass.: Addison-Wesley.

Regan, P. and Hamilton, S. (2004). 'NASA's Mission Reliable'. *IEEE Computer*, **37** (1), 59–68.

Schneider, S. (1999). *Concurrent and Real-time Systems: The CSP Approach*. Chichester, UK: John Wiley and Sons.

Visser, W., Havelund, K., Brat, G., Park, S. and Lerda, F. (2003). 'Model Checking Programs'. *Automated Software Engineering J.*, **10** (2), 203–32.

Voas, J. (1997). 'Fault Injection for the Masses'. *IEEE Computer*, **30** (12), 129–30.

Wordsworth, J. (1996). *Software Engineering with* B. Wokingham: Addison-Wesley.

Zheng, J., Williams, L., Nagappan, N., Snipes, W., Hudepohl, J. P. and Vouk, M. A. (2006). 'On the value of static analysis for fault detection in software'. *IEEE Trans. on Software Eng.*, **32** (4), 240–5.

ИАН СОММЕРВИЛЬ

БАҒДАРЛАМАЛЫҚ ЖАСАҚТАМА. 1-БӨЛІМ

Оқулық

Басуға қол қойылған күні 27.12.2013 ж.
Қағазы офсеттік. Қаріп түрі «Times».
Пішімі 70 × 100$^{1/16}$. Шартты баспа табағы 29,0.
Таралымы: мемлекеттік тапсырыс бойынша – 680 дана
+ баспа есебінен – 20 дана. Тапсырыс № 8830.

Тапсырыс берушінің файлдарынан Қазақстан Республикасы
«Полиграфкомбинат» ЖШС-де басылды.
050002, Алматы қаласы, М. Мақатаев көшесі, 41.