ҚҰРАМЫНДА АКТИВТІ ЗАТТАР БАР КОНТЕЙНЕРЛЕРДІ АЛУ ҮШІН КЕНЕТТЕН ЭМУЛЬГИРЛЕНГЕН МАЙ/СУ ЭМУЛЬСИЯЛАРДЫ ЗЕРТТЕУ

С. Б. Айдарова[1], А. Б. Тілеуова[1,2], А. А. Шәрипова[1,2], Н. Е. Бектұрғанова[1], Д. О. Григорьев[2], Р. Миллер[2]

[1]Казахский национальный технический университет им. К. И. Сатпаева, Алматы, Казахстан,
[2]Макс-Планк институт коллоидов и межфазных поверхностей, Потсдам, Германия

Тірек сөздер: Пикеринг эмульсиялар, наноэмульсиялар, микрокапсулдау, кенеттен эмульгирлеу, субмикрокапсулалар, нанокапсулалар.

Аннотация. Құрамында активті заттар бар контейнерлерді алу үшін кенеттен эмульгирленген май/су Пикеринг эмульсиялар қолданылды. Еркінрадикалды полимерлеу әдісін қолдану арқылы 3-(Триметоксисили)пропил метакрилат (ТПМ) пен гексадецилтриметоксисиланның (ГДТМС) капсулалары (150-400 нм) алынды. Олардың өлшемдері, зета-потенциалы, құрылыс-морфологиялық қасиеттері зерттелінді.

Поступила 22.05.2015 г.

BULLETIN OF NATIONAL ACADEMY OF SCIENCES
OF THE REPUBLIC OF KAZAKHSTAN
ISSN 1991-3494
Volume 4, Number 356 (2015), 13 – 21

TESTING OBJECT-ORIENTED SYSTEMS

A. K. Mustafina, J. M. Alibieva, G. S. Beketova, A. U. Utegenova, A. B. Berlibaeva

Kazakh National Technical University after K. I. Satpayev, Almaty, Kazakhstan.
E-mail: alibieva_j@mail.ru

Key words: object focused systems, testing, inspection, verification, certification, testing metrics, constructive model

Abstract. Testing plays the vital role in development of the qualitative software. Nevertheless, in many companies which are engaged in development of the software, processes of testing are insufficiently organized therefore performers are compelled to go a difficult way, trying to achieve desirable results. And in testing of the object-oriented software, the main attention is paid to real planning and effective realization of process of testing of the object-oriented and component software. Development begins with creation of the visual models reflecting static and dynamic characteristics of future system. In the beginning these models fix initial requirements of the customer, then formalize implementation of these requirements by allocation of objects which interact with each other by means of transmission of messages. The most part of expenses of object-oriented process of development are the share of designing of models. If to add to it that the price of elimination of a mistake promptly grows with each iteration of development, it is absolutely logical to test the requirement object-oriented models of the analysis and design.

This article considers advantages of use of modern programs of testing, their types, levels, a cost assessment of productivity of work of the program by means of basic formulas of calculation, the model of the functional directed metrics is given.

УДК 57.087.1: 004. 4 (075)

ОБЪЕКТІГЕ БАҒЫТТАЛҒАН ЖҮЙЕЛЕРДІ ТЕСТІЛЕУ

А. К. Мұстафина, Ж. М. Әлибиева, Г. С. Бекетова, А. У. Өтегенова, А. Б. Берлібаева

Қ. И. Сәтбаев атындағы Қазақ ұлттық техникалық университеті, Алматы, Қазақстан

Тірек сөздер: объектілі бағытталған жүйелер, тестілеу, инспекция, верификация, аттестация, тестілеу метрикалары, конструктивті модель.

Аннотация. Тестілеу сапалы программалық қамтамаларды өңдеуде маңызды роль атқарады. Осыған қарамастан, программалық қамтамаларды өңдеумен айналысатын кәсіпорындарда, тестілеу процесі жеткілікті деңгейде ұйымдастырылмаған, сондықтан орындаушылар жеткілікті деңгейдегі нәтижелерге жету үшін қиын жолдармен өтуі керек болады. Ал объектілі бағытталған программалық қамтамаларды тестілеу кезінде, негізгі көңіл нақты жобалауға және объектілі-бағытталған әрі компоненттік программалық қамтамаларды тестілеу процесін тиімді таратуға бөлінеді. Өңдеу келешекте пайда болатын жүйенің статикалық және динамикалық сипаттамаларын бейнелейтін визуальды моделдерді құрудан басталады [9-11]. Бастапқыда бұл моделдер тапсырыс берушінің бастапқы талаптарын, фиксирлейді, одан кейін бұл талаптардың объектілерді ерекшелеу жолымен таратылуын қалыптастырылады, олар бір-бірімен хабарламалар алмасу арқылы әрекеттеседі. Объектілі-бағытталған процесстерді өңдеу кезінде моделдерді құрылымдау үшін көптеген шығындар кетеді. Егер осыған, әрбір итерация сайын пайда боатын қателерді жоюды қосатын болсақ, онда объектілі-бағытталған моделдерді талдау және жобалауды тестілеуден өткізу керектігін талап ету әдбден мүмкін болады.

Бұл қарастырылып отырған мақалада қазіргі таңда объектілі бағытталған жүйелерді тестілеу маңыздылығы, тестілеу түрлері, тестілеу деңгейлері, программалық жобалардың құнын бағалау ұсыныстары белгілі формулалармен берілген, әрі функциональды-бағытталған метрикалары мен бағалаудың конструктивті моделі келтірілген.

Кіріспе. Программалық қамтаманы тестілеу – жүйенің дұрыс жұмыс жасайтындығын тексеру үшін программалық өнімнің жұмыс сипаттамалары, орындалатын кодтың шығу деректерінің зерттелуі және тестілік деректерді жіберу [12].

Тестілеу – бұл верификация және аттестациялаудың динамикалық әдісі, себебі орындалатын жүйеде қолданылады.

Верификация және аттестация – деп тексеру және талдау процессі аталады, олардың жұмысы кезінде программалық қамтама өз спецификациясына және тапсырыс беруші талаптарына сәйкестігі тексеріледі. Верификация және аттестация программалық қамтаманың өмірлік циклын толық қамтиды, олар талаптарды талдау кезеңінде басталады және дайын программалық жүйенің тестілеу кезеңінде программалық кодты тексерумен аяқталады.

Верификация жүйе дұрыс құрылды ма деген сұраққа, ал аттестация жүйе дұрыс жұмыс жасайды ма деген сұраққа жауап береді [1, 5].

Осы берілген анықтамаларға сәйкес, верификация программалық қамтаманың жүйелік спецификацияға сәйкестігін тексереді, негізінде функциональды және функциональды емес талаптар тексеріледі. Аттестация – бұл верификацияға қарағанда жалпы процесс, аттестация уақытында программалық өнім тапсырыс берушінің күтілген ойына сәйкестігін тексереді. Аттестация верификациядан кейін жүргізіледі. Верификацияда, әрі аттестацияда жүйені тексеру және талдаудың негізгі екі әдістемесі қолданылады.

Программалық қамтама инспекциясы – әзірлеу процессінің барлық кезеңдерінде жүйенің әртүрлі көрсетілімдерін (артефактілерін) тексеру және талдау. Инспекциялау – бұл верфикация және аттестациялаудың статикалық әдісі, себебі оларға орындалатын жүйе талап етілмейді [1, 13].

Зерттеу әдістемесі және жалпы ақпараттар. Программалық қамтаманы тестілеуге арналып жазылған көптеген әдебиеттерде, программалық қамтама функционалды моделін тарататын, программалық жүйелердің тестілеу процестері сипатталады, бірақ объектіге бағытталған жүйелердің жеке тестілеуі қарастырылмайды [2, 15].

Функциональды модельдер бойынша және объектіге бағытталған жүйелер бойынша өңделген жүйелердің маңызды ерекшеліктері бар:

- объектілер жеке ішкі программа мен функцияларға қарағанда маңыздырақ болады;
- ішкіжүйелерге интеграцияланған объектілер әдетте өзара оңай байланысқан, сондықтан жүйенің ең жоғарғы деңгейін анықтау қиынға түседі;
- қайта қолданылатын объектілерді талдау кезінде, олардың орындалатын кодты тестілеуші үшін қолжетімді емес болуы мүмкін.

Бұл ерекшеліктер объектілерді тексеру кезінде оларды кодты талдауға негізделген ақ жәшік әдісімен тексеруге болатындығын білдіреді, ал жинау кезіндегі тестілеуде басқа жолдарды қолданған дұрыс.

Объектіге бағытталған жүйелерге қолдануға болатын келесі тестілеу деңгейлерін анықтауға болады [2, 3, 7, 14]:

Объектілермен ассоциацияланған жеке әдістерді (операцияларды) тестілеу. Әдетте әдістер өздерімен функция немесе процедураларды көрсетеді. Сондықтан мұнда қара және ақ жәшік әдістерімен тестілеу жүргізуге болады.

Функционалды тестілеу немесе қара жәшік әдісімен тестілеу жүйенің немесе оның компоненттеріне спецификацияланған барлық тестілерге негізделіп базаланады. Қара жәшік сияқты жүйенің тәртібін тек оның кіріс сәйкесінше шығыс деректерін оқып үйрену арқылы анықтауға болады, яғни программалық қамтама таратылуы тексерілмейді, ал оның орнына оның орындалатын функциялары тексеріледі. Ақ жәшік әдісі деп аталатын құрылымдық тестілеу әдісі, жүйенің құрылымы және оның таратылуына негізделіп құрылады. Мұндай әдіс тәртіп бойынша салыстырмалы түрде үлкен емес, программалық элементтерге қолданылады, мысалы объектілермен ассоциацияланған ішкі программалар немесе әдістер. Мұндай жол кезінде жобалаушы тестілік деректер алу үшін компонент құрылымы жайында білімдерін қолданады, программалық кодты талдайды [6, 7].

Объектінің жеке кластарын тестілеу. Қара жәшік әдісімен тестілеу принципі еш өзгерусіз қалады, бірақ «эквивалентті класс» түсінігін кеңейту керек [7, 16].

Жүйені тестілік жабу жолы программадағы барлық операторлар ең болмағанда бір рет орындалуын, сонымен қатар, барлық программаның тармақтары орындалуын талап етеді. Объектілерді тестілеу кезінде толық тестілік жабу төмендегілерден тұрады:

- объектілермен ассоциацияланған барлық әдістерді бөлек тестілеу;
- объектілермен ассоциацияланған барлық атрибуттарды тексеру;
- күйлерді модельдеу үшін керек, объект күйлерін өзгертуге әкелетін, объектінің мүмкін болатын барлық күйлерін тексеру. Мысалы, инсталляциядан кейін объект атрибуттарының тапсырмаларын тексеретін тестілер керек болған кезде. Сонымен қатар объект әдістері үшін бақылау тестілерін анықтап алу және осы әдістерді тәуелсіз тестілеуден өткізу керек. Объект күйлерін тестілеу кезінде оның күйлерінің моделі (UML күйлер диаграммасы) қолданылады, оның көмегімен тестілеуден өткізу керек күйлер тізбегін анықтап алуға болады.

Мұрагерленуді қолдану объект кластары үшін тестілерді әзірлеуді қиындатады. Егер класс ішкі кластардан мұрагерленген әдістерді көрсететін болса, онда барлық ішкі кластарды барлық мұрагерленген әдістерімен бірге тестілеуден өткізу керек [5, 7].

Объектілердің кластерлерін тестілеу. Бәсеңдейтін және өспелі жинау жолдары байланысқан объектілер тобын құру үшін жарамайды. Сондықтан мұнда басқа тестілеу әдістерін қолданған дұрыс, мысалы, сценарийлерге негізделген әдістер. Объектіге бағытталған жүйелерде модулдерді тестілеу үшін тікелей эквиваленттер жоқ. Бірақ сервистер жиынын бірге беретін кластар тобын бірге тестілеуге болады деп есептеледі. Мұндай тестілеу түрін кластерлерді тестілеу деп атайды [8, 14].

Кластерлерді құру, осы кластерлердің көмегімен таратылатын әдістер мен сервистерді ерекшелеуге негізделеді. Объектіге бағытталған жүйелердің жиындарын тестілеу үшін үш жол қолданылады.

- Қолдану варианттарын және сценарийлерді тестілеу. Қолдану варианттары (Use Case) немесе сценарийлер жүйенің қандай да бір режимінің жұмысын сипаттайды. Тестілеу берілген Use Case тарататын осы сценарийлер мен объектілердің кластерлерін сипаттауға негізделеді.
- Ағымдарды тестілеу. Бұл жол жүйелік откликтерді деректерді енгізу немесе енгізілетін оқиғалар тобын тексеруге негізделеді. Объектіге бағытталған жүйелер тәртіп бойынша, оқиғалы-басқарылатын болып табылады, сондықтан олар үшін берілген тестілеу түрі әбден сәйкес келеді. Бұл жолды қолданған кезде жүйедегі негізгі және альтернативті оқиғалар ағымын әзірлеу қалай жүзеге асатындығын білу керек.
- Объектілер арасындағы қарым-қатынастарды тестілеу. Бұл байланысатын объектілер тобын тестілеу әдісі. Бұл жүйені жинау аралық тестілеу деңгейі жолдарды «әдіс-хабарлама», объектілер арасындағы байланыстар тізбегін қадағалауды анықтауға негізделген. UML тізбек диаграммаларын тестілік сценарийлерді әзірлеу үшін тестіленетін операцияларды анықтауда қолдануға болады. Сценарийлерді тестілеу көп жағдайларда басқа тестілеу әдістеріне қарағанда тиімді болып табылады. Тестілеу процессінің өзін бірінші кезекте мүмкін болатын сценарийлер тексерілетінде, сосын ғана мүмкін болмайтын сценарийлер тексерілетін етіп жобалауға болады. Сценарийлер өңделген қолдану варианттарынан алынып анықталады, және керек кезде, жүйелік объектілердің көмегін қолдана отырып қолдану варианттарын бейнелейтін әрекеттесу диаграммаларын қолдануға болады.

Жүйені тестілейтін сценарийлерді таңдағаннан кейін әрбір кластың әдістері болмағанда бір рет орындалатындығына көз жеткізу керек. Әрине, әдістердің барлық комбинацияларын орындау мүмкін емес, бірақ, ең болмағанда, барлық әдістердің қандай-да бір орындалатын әдістер тізбегі ретінде тестілеуден өткізілгендігіне көз жеткізуге болады [6, 8].

Интерфейстерді тестілеу. Тәртіп бойынша, интерфейстерді тестілеу модульдер немесе ішкі жүйелер үлкен жүйелерге интеграцияланған жағдайлар кезінде орындалады. Әрбір модуль немесе ішкі жүйенің басқа компоненттерімен шақырылатын берілген интерфейстері болады. Интерфейсті тестілеудің мақсаты – жүйеде пайда болған қателерді, интерфейстегі қателердің салдарын немесе интерфейстер жайындағы қате тұжырымдамаларын табу [2].

Берілген тестілеу түрі объектіге бағытталған жобалау кезінде ерекше маңызды, ашып айтар болсақ, объектілер және объект кластарын қайта қолданған кезде пайдалы. Объектілер маңызды деңгейлерде интерфейстердің көмегімен анықталады және әртүрлі жүйелердің, әрі әртүрлі объектілердің әртүрлі комбинацияларында қайта қолданылуы мүмкін. Жеке объектілерді тестілеу уақытында интерфейс қателерін шығару мүмкін емес, өйткені олар бір объектінің жекеленген тәртібіне қарағанда объектілер арасындағы қарым-қатынас нәтижесі болып табылады.

Программа компоненттерінің арасында көптеген интерфейс түрлері болуы мүмкін және сәйкесінше әртүрлі интерфейс қателері де болады. Мұндай интерфейстерге келесілерді жатқызады: параметрлік интерфейстер, бөлінетін жады интерфейстері, процедуралық интерфейстер және хабарламаларды беру интерфейстері.

Интерфейстегі қателер қиын жүйелердегі қателердің кеңінен таратылған түрлеріне жатады және үш класқа бөлінеді: интерфейстерді дұрыс емес қолдану, интерфейстерді дұрыс емес түсіну және синхронизация қателері.

Интерфейстерді тестілеудің бірнеше жалпы тәртіптері бар, олар:

- сыртқы компоненттермен берілетін параметрлердің экстремалды параметрлерін қолдану, ол жоғарғы ықтималдылықпен интерфейстердегі сәйкес еместікті табады;
- көрсеткіштің нөлдік параметріндегі интерфейсті тестілеу;
- компонентті процедуралық интерфейс арқылы шақырған кезде, компоненттің жұмысына кедергі болатын тестілерді қолдану;
- хабарламаларды беру жүйесінің әдеттегі жұмысына қарағанда бірнеше рет хабарламалар санын артығырақ генерациялайтын тестілерді өңдеу;
- бөлінетін жады арқылы бірнеше компоненттердің әрекеттесуі кезінде, компоненттер активизациясының тәртібін өзгертетін тестілерді өңдеу.

Программалық жобаның құнын бағалау. Жобаға тапсырыс берушілер үнемі күшейтілген қызығушылықпен программалық жобаның бағасына көңіл қояды. Бағалық құны дұрыс емес есептелген жағдайда ең жақсы өнімнің өзі фиаскоға ұшырауы мүмкін. Жалпылай алсақ, жобаның құны фиксирленген түрде болуы керек екендігі міндетті емес. Егер жобаның құны фиксирленген болған күннің өзінде, оның бастапқыда берілген құнына сәйкес екендігіне көз жеткізу және сенімді болу үшін берілген талаптар жиынтығының таратылу құнын бағалау, әрі шығындар құнын жаңадан есептеу керек. Барлық уақытта ең болмағанда талаптарды талдау фазасында құндық диапазон бағасын шамамен шығарып отыру керек. Өнімге қойылатын талаптарды неғұрлым көп білген сайын, жобаның соғұрлым дамығандығын, әрі жоба құнының бағасын нақтырақ айтуға болады [2, 6].

Жобаның орындалу уақытын және құнын ертерек бағалау әдісінің дәстүрлі сұлбасы келесі ұсыныстардан тұрады:

- жоба құнын және жоба ұзақтығы немесе кодтың жолдарының санын тікелей бағалау үшін өлшемдік-бағытталған метрикаларды қолдана отырып алдыңғы жұмыстармен салыстыру;
- кодтың жолдарының санын бағалау үшін функционалды-бағытталған метрикалар әдісін қолдану, ол өз кезегінде жақындатылған функциональды өлшемді есептеу арқылы немесе нақтылау процесін қабылдау арқылы жүргізіледі;
- одтың жолдарының санын бағалауды еңбек шығындарын есептеуде әрі жоба ұзақтығын есептеуде COCOMO формуласының көмегімен бірге қолдану.

Кодтың жолдарының санын функционалды өлшемдерінсіз бағалау.

Мұндай бағалау өте ерте фазаларда жобалау және кодтауға дейін мүмкін және өлшемдік-бағытталған метрикаларға негізделген.

Өлшемдік-бағытталған метрикалар тікелей программалық өнімді және оны әзірлеу процесін өлшейді. Метрикалар Loc-бағалауға (Lines Of Code) негізделеді, олар бірнеше бір-біріне ұқсас жобаларда орындалып қолданылғандар (LOC – бұл программалық өнімнің жолдарының саны). LOC- метрикалардың бастапқы деректерін есептеу үшін кесте қолданылады.

Кесте

Жоба	Шығындар, адам-ай	Құны, $ мың	Kloc, мың. LOC	Құжаттар беті	Қателер	Адамдар
PR1	24	168	12.1	365	29	3
PR2	62	440	27.2	1224	86	5

Кесте соңғы бірнеше жылдағы жобалар жайындағы деректерден тұрады. Мысалы, PR1 жобасы жайындағы жазба: программаның 12 100 жолы 24 адам – ай уақытында өңделгендігін және оның құны $ 168 000 ақша тұратындығын көрсетеді. Сонымен қатар, PR1 жобасы бойынша 365 құжаттар бетінің өңделгендігін және 29 қатенің тіркелгендігін көруге болады. Ал жобаны үш адам өңдеп шыққан.

Кесте негізінде сапа және өнімділіктің (әрбір жоба үшін) өлшемдік-бағытталған метрикалары есептеледі:

$$\text{Онімділігі} = \frac{\text{узындыгы}}{\text{шыгындар}}\left[\frac{\text{мын.LOC}}{\text{адам.орын}}\right];$$

$$\text{Сапасы} = \frac{\text{кателер}}{\text{узындыгы}}\left[\frac{\text{бірлік}}{\text{мын.LOC}}\right];$$

$$\text{Алыстаган\_nnны} = \frac{\text{nnны}}{\text{узындыгы}}\left[\frac{\text{мын.\$}}{\text{мын.LOC}}\right];$$

$$\text{Кужатталгандыгы} = \frac{\text{кужат\_парактары}}{\text{узындыгы}}\left[\frac{\text{парак}}{\text{мын.LOC}}\right].$$

Өлшемдік-бағытталған метрикалардың артықшылықтары: кеңінен таралған, қарапайым және оңай есептеледі. Өлшемдік-бағытталған метрикалардың кемшіліктері: программалау тілдеріне тәуелді; жобаның бастапқы кезеңдерінде алуға қиын түсетін бастапқы деректерді талап етеді; процедуралы емес программалау тілдерінде қолдануға бейімделмеген.

Функциональды-бағытталған метрикалар. Берілген метрикалар жанама түрде программалық өнімді және оның әзірлеу процессін өлшейді. LOC-бағалауды санаудың орнына, оның өлшемі қарастырылмайды, оның орнына функциональдылық немесе өнімнің пайдалылығы қарастырылады. 1979 жылы Альбрехт фундаменталды түсініктеме ұсынды – ол жобалаудан тәуелсіз кез келген жобаның өлшемін өлшеу үшін функционалды өлшемді (FP – functional points) қолдану болатын. Бұл әдіс қосымшаның барлық мүмкіндіктерін бір бейнелі өлшеуден және қосымшаның өлшемін бір сан түрінде өрнектеуден тұрады, одан кейін ол код жолдарының санын, құнын және жоба мерзімдерін бағалау үшін қолданылады. Функционалды өлшем болашақта пайда болатын программаның мүмкін болатын мәндерін өлшеу үшін таласады [4, 5, 8].

Функционалды өлшемді анықтау әдістері төмендегі қадамдардан тұрады:

1. Қосымшаның барлық функцияларының идентификациясы (мысалы, «деректерді іздеу», «деректерді бейнелеу» және т.б.). Функциялар ерекшеленетін критерийлер, келесі адрес бойынша бейнеленген: http://www.ifpug.org/home/docs/freebies.html. Функциональдылық программалық код деңгейінде емес, пайдаланушы деңгейінде қарастырылады. Әдетте функция бір экран формасын әзірлеуге сәйкестендірілген.

2. Әрбір ерекшеленген функция үшін келесі түрдегі факторлар саны есептелінеді: сыртқы кірістер және сыртқы шығыстар, сыртқы сұраныстар, ішкі логикалық файлдар, сыртқы логикалық файлдар.

3. Әрбір анықталған факторлар қосымшадағы берілген фактордың қиындық деңгейімен анықталатын коэффициентке көбейтіледі. Халықаралық функционалды өлшемді пайдаланушылар тобы (IFPUG-International Function Point Users Groups) функцияларды ерекшелеу критерийлерін және қолданылатын факторларды бағалау кезінде нені «қарапайым» және нені «қиын» етіп санау керек екендігін бөлшектік сипаттау арқылы басып шығарды.

4. Жоба 14 жалпы сипаттама жиынымен бағаланады, олардың әрбіріне қосымша функцияларына сенімсіздік тудыруына тәуелді 0-ден 5-ке дейінгі өлшемдер меншіктеледі. Мұндай сипаттамаларға келесі сұрақтарға жауап бере алатын жауаптар жатады: резервтік көшіру талап етіледі, деректер алмасу талап етіледі, таратылған есептеулер қолданылады, деректерді енгізу үшін көптеген формалар қолданылады, деректер қорының аймақтары оперативті түрде жаңартылады, ішкі есептеулер қиын, кодты қайта қолдануға болады, жаңарту мүмкіндіктерін әрі қолдану қарапайымдылығын қолдау талап етіледі және т.б.

5. Соңында, нақтыланған функционалды өлшем (НФӨ) келесі формула бойынша есептеледі:

$$\text{НФӨ} = [\text{Жақындатылған функционалды өлшем}] \times$$
$$\times\ [0{,}65+0.01\cdot(\text{жалпы сипаттамалардың қосындысы})].$$

Бұл формуланың негізгі мақсаты, егер қосымшаға ешқандай арнайы талаптар қойылмаса (барлық жалпы сипаттамалар нөлге тең болса), онда нақтыланбаған функционалды өлшемді 35%-ға дейін кеміту керек, әйтпесе басқа жағдайларда, нақтылынбаған өлшемді жалпы сипаттамалардың мәндерінің әрбір бірлігі үшін 1%-ға дейін өсіріп отыру керек.

Бағалаудағы қорытынды қадам қодтағы жолдар санын есептеу үшін функционалды өлшемді қолданумен байланысты. Ол өз кезегінде, яғни кодтағы жолдар саны жалпы адам – еңбек сыйымдылығын және жоба мерзімдерін анықтауға көмектеседі.

Интернетте функционалды өлшемдерді есептеу үшін арнайы бос таратылатын аспаптар бар, олар: http://www.construx.com адресі бойынша орналасқан.

Альбрехт бойынша, «функционалды өлшем» метрикасы, функционалды көрсеткіш FP (Function Point) ретінде беріледі және келесі формула бойынша есептеледі:

$$\text{FP} = \text{Жалпы саны} \cdot (0{,}65+0{,}01\cdot \sum \text{Fi}),$$

мұндағы Fi – қиындықты реттеу коэффициенті (салмағы 0-ден 5-ке дейін), i =1,14.

FP есептеп болғаннан кейін, оның негізінде, өнімділік, сапа және тағы басқалары метрикалар қалыптастырылады:

$$\text{Онімділігі} = \frac{\text{функц.корсеткіші}}{\text{шығындар}}\left[\frac{\text{FP}}{\text{адам.орын}}\right];$$

$$\text{Сапасы} = \frac{\text{кателері}}{\text{функц.корсеткіші}}\left[\frac{\text{бірлік}}{\text{FP}}\right];$$

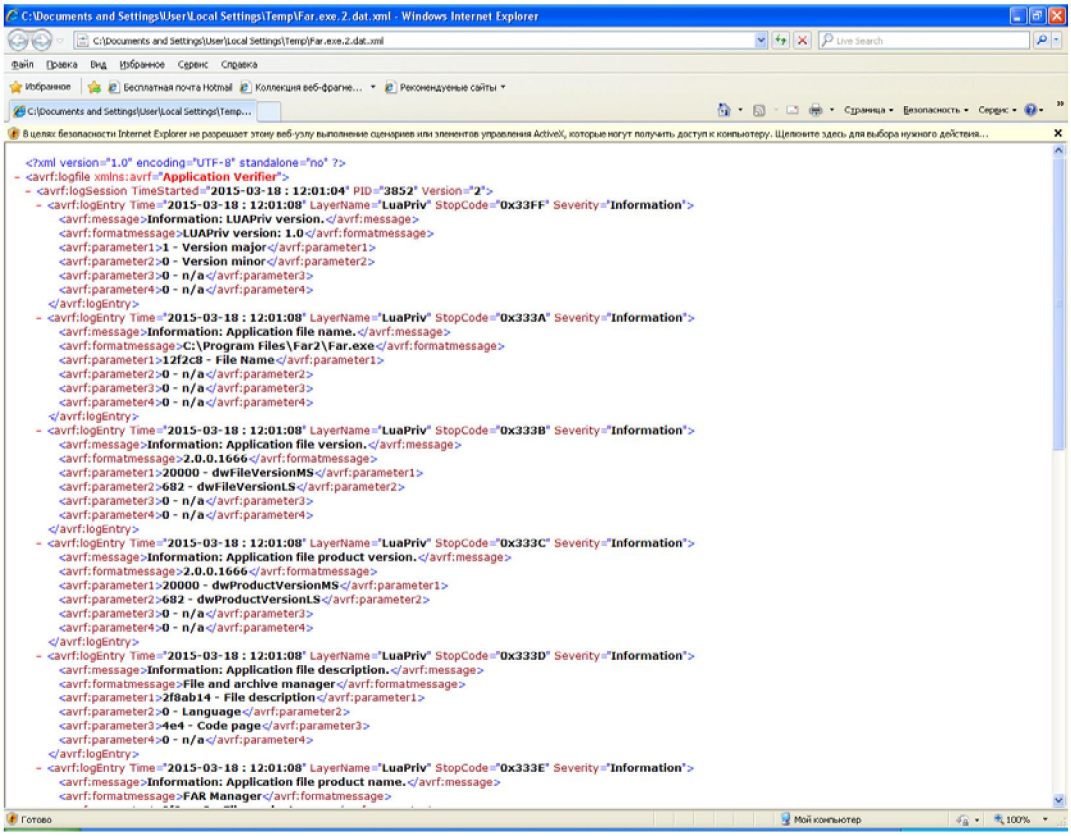
$$\text{Алыстаган\_nnны} = \frac{\text{nnны}}{\text{функц.корсеткіші}}\left[\frac{\text{мын.\$}}{\text{FP}}\right];$$

$$\text{кужатталгандыгы} = \frac{\text{nnжат\_парактары}}{\text{функц.корсеткіші}}\left[\frac{\text{парак}}{\text{FP}}\right].$$

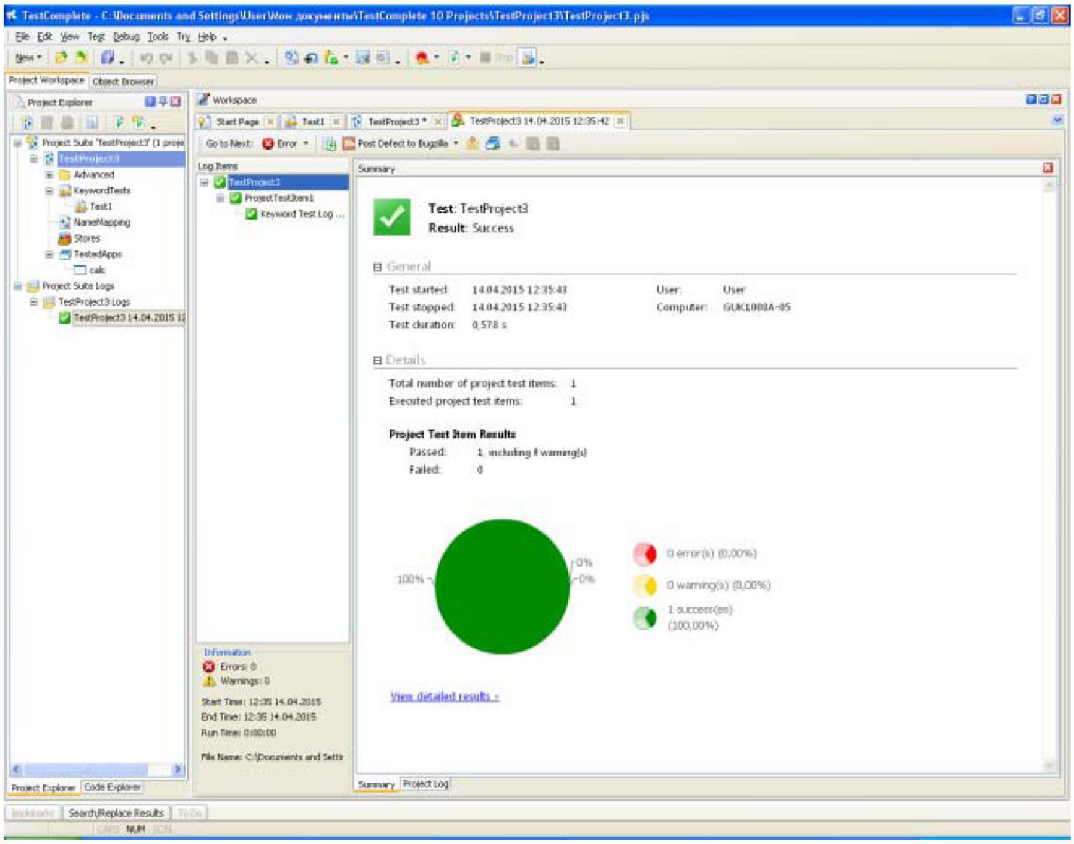
Функционалды-бағытталған метрикалардың артықшылықтары: программалау тіліне тәуелді емес, жобаның кез келген кезеңінде оңай есептеледі.

Метрикалардың кемшіліктері, оның нәтижелерінің субъективті деректерге негізделетіндігі және тікелей емес жанама өлшемдердің қолданылатындығы болып табылады.

Бағалаудың конструктивті моделі. Берілген моделде формулаларды шығару үшін статистикалық жол қолданылған, онда көптеген жоба сандарының ішіндегі нақты нәтижелер ескерілген. Күшті моделдің авторы – Барри Боэм (1981) – оған COCOMO (Constructive Cost Model) атын берген.



1-сурет – Application Verifier тестілеу көрінісі



2-сурет – TestComplete 10 тестілеу көрінісі

Одан әрі замандандырылған COCOMO II моделі XXI ғасыр программалық инжене-риясында қолдануға бағытталған. Берілген модель келесілерден тұрады [8]:

- қосымша композициясының моделі, ол пайдаланушы интерфейстерін макеттеу, программалық қамтама және компьютерлік жүйенің қарым-қатынасы, технологияның даму деңгейі және өнімділікті бағалауды қарастырады, әрі объектілік көрсеткіштерді қолдануға бағытталған;
- ерте жобалау кезеңінің моделі, ол талаптарды нақтылау және базалық программалық архитектураны анықтау кезінде қолданылады;
- пост-архитектура кезеңінің моделі, ол архитектура қалыптастырылып болғаннан кейін және программалық өнімнің ары қарайғы әзірлеуі орындалу кезінде қолданылады.

COCOMO II – программалық жобаларды басқарумен байланысты көптеген есептерді шешуге мүмкіндіктер беретін беделді және көпжоспарлы модель.

Қорытынды. Сонымен қорыта айтар болсақ, тестілеу программаларын қолдану артықшылықтары мен кемшіліктерін біз Қ.И.Сатпаев атындағы ҚазҰТУ докторанттары "Программалық қамтаманы тестілеу және сапамен қамтамасыз ету" пәнін оқу барысында толық қарастырып, бұл программалық қамтамалардың тестілену процесінің жұмысымен Application Verifier, TestComplete 10 программаларын қолданып тәжірибе жүзінде қолданыстан өткіздік, эсперттік сараптамалар жасап, қадағалап үйрендік (жоғарыда 1, 2-суреті қараңыз).

Бақылау, қадағалау және қолдану нәтижесінде келесі жетістіктерге жетуге болатындығын айта аламыз:

- жүйеде пайда болған қателерді, қателердің салдарын немесе қате тұжырымдамаларын табу;
- функционалданудағы өнімділікке жету;
- пайдалунышылардың талаптары және сұраныстарының толық орындалуын қамтамасыздандыру.

ЛИТЕРАТУРА

[1] Вигерс Карл. Разработка требований к программному обеспечению. Пер. с англ. М.: Издательство: торговый дом «Русская Редакция», 2004 г. 576с.

[2] Г.Буч. Объектно-ориентированный анализ и проектирование с примерами приложений на С++, 2-е изд. Пер. с англ. М.: «Издательство БИНОМ», СПб.: Невский диалект,1998 г. 560с.

[3] Е.В.Пышкин. Основные концепции и механизмы объектно-ориентированного программирования. СПб.: БХВ - Петербург, 2005. 640с.

[4] Б.С.Кубеков. Технология разработки программного обеспечения. Учебник. Алматы: Экономика, 2011 г. 307с.

[5] Дж.Макгрегор, Д.Сайкс. Тестирование объектно-ориентированного программного обеспечение. Практическое пособие. ТИД "ДС", 2002 г. 432 с.

[6] Л.Тамре. Тестирование программного обеспечение. Москва: Вильямс, 2003 г. 368 с.

[7] Б. Бейзер. Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем. Питер, 2004 г. 320 с.

[8] И. Винниченко. Автоматизация процессов тестирования. Питер, 2005 г. 203 с.

[9] Амблер С. Гибкие технологии: экстремальное программирование и унифицированный процесс разработки. Библиотека программиста. СПб.: Питер, 2005 г. 412 с.

[10] Брауде Э. Технология разработки программного обеспечения. СПб.: Питер, 2004 г. 655 с.

[11] Алистер Коберн. Современные методы описания функциональных требований к системам. Издательство "Лори", 2002 г. 263 с.

[12] Леффингуэлл, Дин, Уидриг, Дин. Принципы работы с требованиями к программному обеспечению. Унифицированный подход. Пер.с англ. М.: Издательский дом "Вильямс", 2002 г. 448 с.

[13] Орлов С.А. Технологии разработки программного обеспечения. Учебное пособие. 2-е издание. СПб.: Питер, 2003 г. 480 с

[14] Соммервилл, Иан. Инженерия программного обеспечения, 6-е издание.: Пер.с англ. М.: Издательский дом "Вильямс", 2002 г. 624 с.

[15] Jacobson, Ivar, Object-Oriented Software Engineering: A Use Case Driven Approach (Addison-Wesley Object Technology Series), Reading, MA: Addison-Wesley, 1994.

[16] Larry Constantine and Lucy Lockwood, Software for Use, Addison-Wesley, 2000.

REFERENCES

[1] K.Wiegers. Development of software requirements. Trans. from English. M .: Publisher: Trading House "Russian Edition", 2004. 576 p.

[2] G.Buch. Object-Oriented Analysis and Design with Applications in C ++, 2nd ed. Trans. from English. M .: "Binom Publishing", SPb .: Nevsky Dialect, 1998. 560 p.

[3] E.V.Pyshkin. The basic concepts and mechanisms of object-oriented programming. SPb .: BHV - Petersburg, 2005. 640 p.
[4] B.S.Kubekov. Software Engineering. Textbook. Almaty: Economics, 2011. 307p.
[5] Dzh.Makgregor, D.Sayks. A Practical Guide to Testing Object-Oriented Software. TTI "DS", 2002. 432 p.
[6] L.Tamre. Introducing Software Testing. Moscow: Williams, 2003. 368 p.
[7] B. Baser. Black-Box Testing. Techniques for Functional Testing of Software and Systems. 2004. 320 p.
[8] I. Vinnichenko. Automation of the testing process. Peter 2005. 203 p.
[9] Ambler S. Flexible technologies: extreme programming and the unified development process. Library of the programmer. SPb.: Peter, 2005. 412 p.
[10] Braude E. Tekhnologiya of development of the software. SPb.: Peter, 2004. 655 p.
[11] Alistair Kobern. Modern methods of the description of functional requirements to systems. "Lori" publishing house, 2002. 263 p.
[12] Leffinguell, Dean, Uidrig, Dean. The principles of work with requirements to the software. The unified approach. Trans. from English. Moscow: Williams, 2002. 448 p.
[13] S. A Eagles. Technologies of development of the software. Manual. 2nd ed. SPb.: Peter, 2003. 480 p.
[14] Somerville, Ian. Software engineering, 6nd ed.: Trans. from English. Moscow: Williams, 2002. 624 p.
[15] Jacobson, Ivar, Object-Oriented Software Engineering: A Use Case Driven Approach (Addison-Wesley Object Technology Series), Reading, MA: Addison-Wesley, 1994.
[16] Larry Constantine and Lucy Lockwood, Software for Use, Addison-Wesley, 2000.

ТЕСТИРОВАНИЕ ОБЪЕКТНО-ОРИЕНТИРОВАННЫХ СИСТЕМ

А. К. Мустафина, Ж. М. Алибиева, Г. С. Бекетова, А. У. Утегенова, А. Б. Берлибаева.

Казахский национальный технический университет им. К. И.Сатпаева, Алматы, Казахстан

Ключевые слова: объектно ориентированные системы, тестирование, инспекция, верификация, аттестация, метрики тестирования, конструктивная модель

Абстракт. Тестирование играет жизненно важную роль в разработке качественного программного обеспечения. Тем не менее, во многих компаниях, занимающихся разработкой программного обеспечения, процессы тестирования недостаточно организованы, поэтому исполнители вынуждены идти трудным путем, пытаясь добиться желаемых результатов. А в тестировании объектно-ориентированного программного обеспечения, основное внимание уделяется реальному планированию и эффективной реализации процесса тестирования объектно-ориентированного и компонентного программного обеспечения. Разработка начинается с создания визуальных моделей, отражающих статические и динамические характеристики будущей системы. Вначале эти модели фиксируют исходные требования заказчика, затем формализуют реализацию этих требований путем выделения объектов, которые взаимодействуют друг с другом посредством передачи сообщений. На конструирование моделей приходится большая часть затрат объектно-ориентированного процесса разработки. Если к этому добавить, что цена устранения ошибки стремительно растет с каждой итерацией разработки, то совершенно логично требование тестировать объектно-ориентированные модели анализа и проектирования.

Данная статья рассматривает преимущества использования современных программ тестирования, их виды, уровни, стоимостную оценку производительности работы программы с помощью основных формул расчета, приведена модель функционально-направленной метрики.

Поступила 22.05.2015 г.