Софийски университет "Св. Климент Охридски"
Факултет по математика и информатика
Катедра "Софтуерни технологии"

Дипломна работа

ТЕМА:
Методика за тестване на нефункционални изисквания към сървърни приложения

Дипломант: Петя Василева Митовска
Специалност: Софтуерни технологии
Факултетен номер: М21511

Научен ръководител: доц. Силвия Илиева, катедра "Софтуерни технологии", ФМИ, СУ "Св. Климент Охридски"
Консултант: Илина Манова, Рила Солюшънс

София, Октомври 2007

Съдържание

Увод

Осигуряването на качеството е от голямо значение при сървърните приложения, тъй като при тях става дума за приложения, засягащи не само една машина, а цели организации или дори както е при уеб сайтовете – милиони потребители. Поради тази причина подборът на техниките за тестване трябва да е много прецизен, а съставянето на тестовия план е от водещо значение, тъй като пропускането на някой съществен детайл от работата на клиент-сървър приложението може да доведе до огромни загуби за бизнеса, който го използва.

Сървърните приложенията са софтуерни продукти и техниките, които се използват при тестването на обикновените десктоп приложения, остават валидни и за тях. Разликата в дейностите по тестването им идва от необходимостта да бъдат изследвани обменът на данните между различните машини и синхронизацията им, производителността в зависимост от броя едновременни връзки, осигуряването на различни права за различните потребители. Именно в тези специфични дейности се състои сложността на тестването при сървърните приложенията.

За всяко от тези нефункционални изисквания към системата има различни дейности, покриващи съответната област, като за всяка дейност е необходимо да се извърши внимателно планиране и определяне на необходимите ресурси за нейното изпълнение.

Цел на дипломната работа

Целта на дипломната работа е да систематизира тези дейности чрез изработването на методика за планиране, изпълнение и оценка на резултатите от тестването на нефункционални изисквания към сървърните приложения. За осъществяването й ще бъде използвано съществуващо сървърно приложение, разработвано за бизнес нуждите на организация, занимаваща се с кредитиране.

Полза от реализацията на дипломната работа

Методиката, която дипломната работа представя, помага при разработването на план за тестване, като същевременно осигурява, че тестовете обхващат всички изисквания на клиента и резултатите от тях носят отговор на въпросите, свързани с нуждите на бизнеса. Пълнотата, систематичността, повтаряемостта и ясната дефиниция на връзката между резултати и цели гарантират, че дейността по тестването на нефункционалните изисквания оправдава усилията и средствата, вложени в нея.

Структура на Дипломната работа

Изложението на дипломната работа обхваща обзор на ролята, значението и мястото на тестването в жизнения цикъл на разработване на софтуера, избор на конкретен проект и функционалност, която ще бъде изследвана, съставяне на план за тестването на тази функционалност, реализирането му и анализ на получените резултати.

В първата глава е обоснована нуждата от тестване при разработването на софтуер. Разглеждат се ролята на контрола на качеството и позицията му в жизнения цикъл на процеса спрямо модела, избран за разработването на софтуера. Класифицират се типовете тестване по два критерия – готовността на софтуерното решение и функционалните и технически изисквания към системата, които покриват. Описва се ползата от автоматизация на провежданите тестове и както и процесът по избор на подходящ инструмент за автоматизация.

Втората глава разглежда проекта, върху който ще бъде разработена представената от дипломната работа методика. Изброяват се техническите характеристики и бизнес изискванията, върху които се изгражда софтуерът. Обосновава се изборът на функционалността, която ще бъде изследвана по-обстойно.

В следващата глава се описва стъпките при планирането на тестовата дейност, като се използва целево-ориентирана техника. Всяка от десетте стъпки включва правила и насоки, както и конкретното им приложение към избрания проект. Резултатът от тази глава е съставянето на план за тестване на издръжливостта на софтуерния продукт спрямо бизнес целите на организацията, която ще го използва.

Последната четвърта глава включва представяне на инструмента, избран за реализиране на тестовете, както и конфигуриране на скриптовете и сценариите, които ще бъдат използвани. Получените резултати се сравняват и анализират, след което се набелязват насоки за бъдещи дейности свързани с тестването на натоварването.

Заключителната част обобщава извършените дейности и техните резултати.

Като приложения са представени планът за тестване, изгледи от използвания инструмент за тестване на натоварването, кодът на скриптовете, използвани за реализация на измерванията, както и резултатите от самите измервания.

В частта Термини и съкращения са предоставени дефиниции на използваните в дипломната работа термини.

В частта Използвана литература се изброяват статиите, книгите и другите източници на информация, използвани по време на разработване на дипломната работа.

1. Обзор

1.1. Роля и значение на тестването

Разработването на софтуер е сложен процес, в който имат значение редица хора на различни нива в процеса – клиенти, анализатори, дизайнери разработчици. Всеки един от тях е склонен към грешки или пропуски, а тези грешки се отразяват върху качеството на разработвания продукт. Разпространено е мнението, че тестерите носят отговорността за качеството на софтуера. Всъщност тестването не подобрява качеството, а просто гарантира, че предлаганият продукт отговаря на изискванията поставени към него.

Тестването има различни роли при разработването на софтуера. Изпълнението и симулацията на различните функционалности на системата, може би е най-популярната дейност, с която повечето хора свързват думата „тестване". При този процес се откриват отклонения от очакваното поведение, грешки при кодирането, конфигурационни и интерфейсни проблеми.

Сигурността в качеството, която тестването осигурява, обуславя значението му за успеха на продукта. Наред с проверката за правилното изпълнение на всяка функционалност се правят и проверки за действията, които системата не би трябвало да извършва. Голяма част от проблемите в софтуера се откриват именно от негативните тестове. Програмистите пропускат някой от тях, тъй като не всички такива ситуации са изрично описани в спецификацията, а освен това броят им многократно надвишава положителните тестови ситуации.

Тестването не само изисква детайлно познаване на системата, но и на бизнес логиката, която системата имплементира. При планирането на тестовия процес се откриват пропуски в изискванията, бизнес ситуации с неясно решение, потенциални проблеми свързани с тези ситуации, противоречия във функционалността. Изясняването им предотвратява грешки и по този начин спестява редица дейности породени от това – описване на грешката, промяна на кода, повторно тестване. В сила е правилото, че колкото по-рано се открие един проблем, толкова по-малка ще е цената за отстраняването му[2]. Може да се каже, че една от целите на тестването е не само да открие грешките в крайния продукт, но и да не допусне такива в по-ранните етапи от разработката му.

До колко може да се разчита на една система? Какъв е капацитетът й? Може ли той да бъде разширен? На тези и други въпроси се дава отговор чрез измерване на оптималните възможности на софтуера. Параметрите, които се измерват, както и техните стойности, могат да варират в зависимост от нуждите на бизнеса.

Често значението на тестването за крайния продукт се подценява при недостиг на време и ресурси. Обикновено то е дейността, която бива ощетена. Недостатъчното тестване, обаче, би могло да доведе до значително увеличаване на разходите по поддръжка, до незадоволителна производителност и надеждност, а от там и до загуба на клиенти.

За сложните приложения от друга страна, не би могло да има компромис с тестовия процес. При тях планирането и организацията на тестовете може да отнеме повече време и усилия от самото им провеждане, но инвестицията е оправдана от крайната гаранция, която се получава за софтуера.

Исторически погледнато, тестването добива все по-голямо значение през годините. В началото дейността му се е състояла от „демонстриране" на софтуера – изпробване на функционалността му в обичайни и нестандартни ситуации, като по този начин се потвърждава до каква степен е завършен продуктът и дали рискът за възникване на проблеми при ползването му е приемлив. Постепенно към задълженията на тестването се добавя и откриването на проблеми, грешки и недостатъци в софтуера, определяне на възможностите му и крайна оценка за качеството на системата и нейните компоненти. След 90те години се включва и „предотвратяване", при което се изчистват неточностите относно изискванията и производителността, осигурява се информация, която предотвратява или намалява вероятността от допускане на някой типове грешки, откриват се грешки в по-ранни етапи от разработката, идентифицират се риска и потенциалните проблеми, както и начини за избягването им в бъдеще. [9]

Според Съмървил [12] наред с развитието на софтуерните технологии, които спомагат за разработката на все по-големи и сложни системи, процесът на тестването трябва да се усъвършенства, за да може да се справи с новите изисквания. Осигуряването на качеството е предизвикателство не само поради бързо еволюиращите технологии, но и поради факта, че пазарът изисква освен разнообразие от функционалност, тази функционалност да бъде предоставена преди конкурентите.

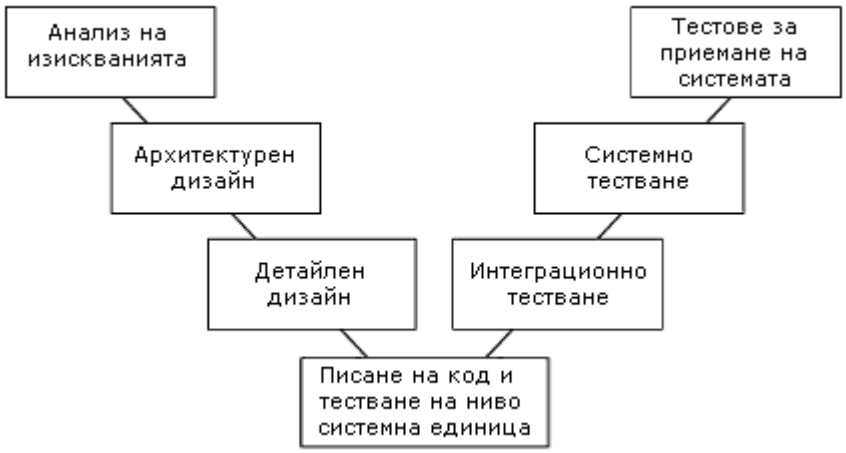
1.2. Мястото на тестването в жизнения цикъл на разработката на софтуер

Различните дейности, които се извършват по време на разработката на софтуера оформят жизнения цикъл на разработването на софтуера (Software Development Lifecycle). Той започва с идентифицирането на изискванията към софтуера и приключва с верификация на разработения продукт спрямо спецификацията му.

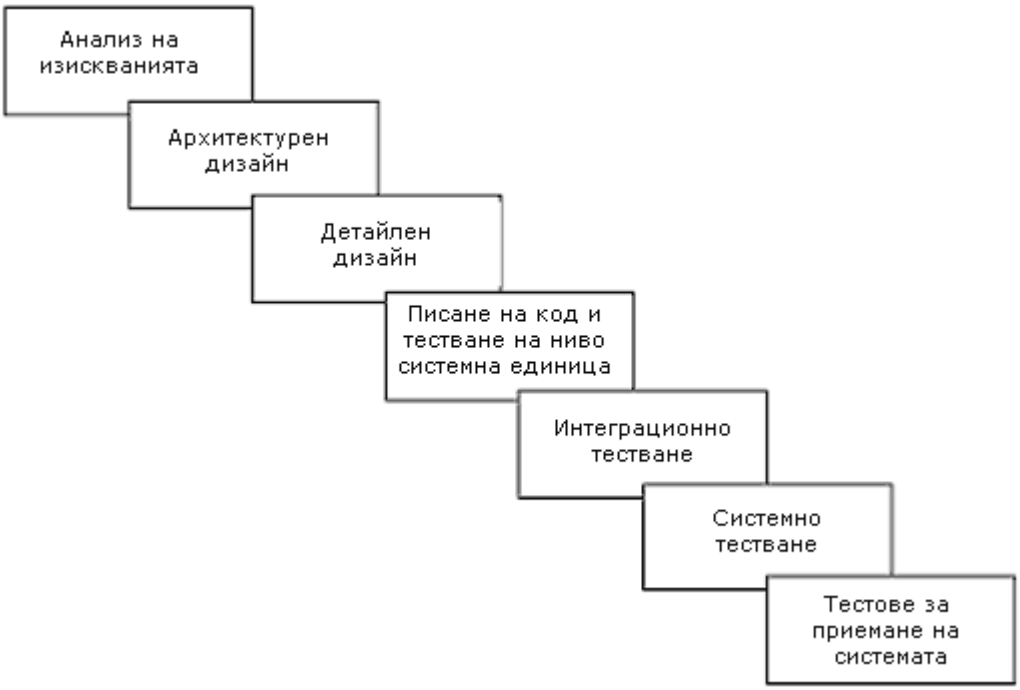
Съществуват много различни модели на жизнения цикъл на софтуер. Всеки един от тях обаче включва в себе си фаза за тестване.

1.2.1. Последователни модели

В миналото моделите използвани за жизнения цикъл на разработката на софтуер са имали последователно развитие, минавайки през редица добре дефинирани фази. Този тип модели се представят от V-модела (Фигура 1) и модела на водопада (Фигура 2).



Фигура 1: V-модел



Фигура 2: Модел на водопада

Има различни вариации и на двата модела, изразяващи се в представянето на различни фази и имащи различни граници между отделните фази. Най-често срещани са следните фази:

- фаза на изискванията – събират се и се анализират изискванията към софтуера, с цел съставяне на пълна и ясна спецификация.

- фаза на архитектурният дизайн – избира се и се проектира софтуерна архитектура, с която да се имплементират отделните компоненти връзките между тях
- фаза на детайлизиране на дизайна – специфицира се подробно имплементацията на всеки един компонент
- фаза на кодиране – имплементират се отделните компоненти на софтуера

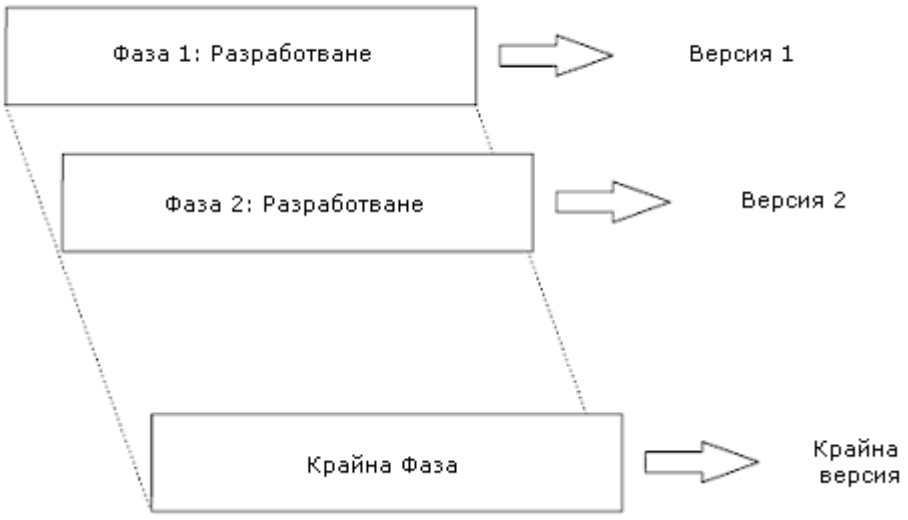
Отделните фази при тестването на софтуера са описани подробно в част 1.3.

Изготвянето на спецификацията на системата става по време на първите три фази от жизнения цикъл на софтуера. Следващите фази, които включват тестване, се базират изцяло на нея.[6]

1.2.2. Прогресивни модели

Моделът на водопада и V-моделът представляват идеализирани модели. Пълното завършване на софтуера е дълъг процес, а често сроковете са кратки и резултат трябва да има бързо, дори и да не е включена пълната функционалност. Решението е компромис между функционалност и срокове, като се представят междинни версии на софтуера с по-малко възможности, които имат за цел стъпка по стъпка да изградят цялата очаквана функционалност. Имплементирането на изискванията към софтуера едно по едно до голяма степен намалява и риска.

Този подход е известен като прогресивно или фазово разработване (progressive/phase development). Съответния му модел е изобразен на Фигура 3 и се състои от отделни фази всяка със свой собствен жизнен цикъл, който обикновено използва модела на водопада или V-модела.

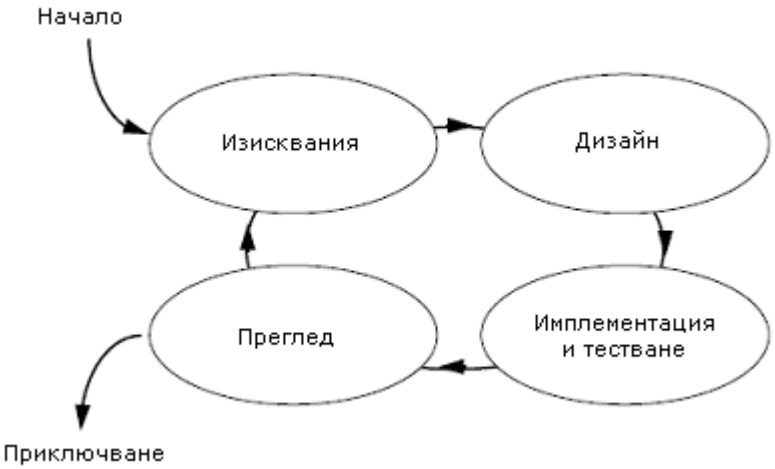


Фигура 3: Прогресивен модел

Софтуерът, който е резултат от всяка фаза на разработка, преминава през серия от тестове, за да се потвърди, че отговаря на изискванията към функционалността, която имплементира. Той може да не бъде използван на практика, но целта му е да послужи като прототип, който да потвърди или отхвърли избраната реализация за дадена критична област от дизайна. При разработването на прототипа се спестява време от тестване, документиране, оптимизация. Ако реализираната в него функция бъде одобрена, е необходимо тя да се пренапише, но без тези „спестявания”.[6]

1.2.3. Итеративни модели

Итеративният модел не се опитва да започне с пълна спецификация на изискванията. Вместо това разработването започва със специфициране и имплементиране само на част от софтуера. След като бъде разгледана тази част се идентифицират допълнителни изисквания. Този процес както е показано на Фигура 4 минава през четири фази и в края на всеки цикъл се получава нова версия, на базата на която да бъдат доразвити изискванията към системата.



Фигура 4: Итеративен модел

Различните фази от итеративния модел включват:

- фаза на изискванията – събират се и се анализират изискванията за софтуера след всяка нова итерация, като по този начин се допълва спецификацията и се достига до нейната крайна версия
- фаза на дизайна – проектира се софтуерно решение, което да отговаря на изискванията, като то може да е изцяло ново или да се базира на вече използван дизайн
- имплементация и тестване – кодира се, интегрира се и се тества софтуерът
- преглед – оценява се разработения софтуер, преглеждат се текущите изисквания и се предлагат промени и допълвания към изискванията

За всеки цикъл от този модел трябва да се реши дали софтуерът, който е резултат от него, ще бъде отхвърлен или ще се използва за отправна точка при следващата итерация. Накрая се достига до момент, в който изискванията са напълно реализирани или не може да се реализира понататъшно развиване на софтуера според изискванията, и се налага да се започне отначало.

Ключът към успешното използване на итеративния метод е задълбочената валидация на изискванията и верификацията на всяка версия от софтуера спрямо тези изисквания. Първите три фази от итеративния метод всъщност са заимствани от последователните методи. Кодът, който е резултат от работата по време на всеки цикъл, има нужда от тестване на ниво системна единица, интеграционно и системно тестване, като тестовете от предходните версии се повтарят и разширяват.[6]

1.3. Типове тестове спрямо фазите от развитието на системата

Независимо кой модел на жизнен цикъл се използва при разработката на софтуера, четири са основните типове тестване, през които се минава в различните фази от развитието на системата.

1.3.1. Тестване на ниво програмна единица (Unit testing)

Този тип тестване изцяло разчита на добро познаване на тествания код, поради което се извършва предимно от програмистите. Обект на тестването е един компонент (системна единица) – неговата функционалност и комуникация с други компоненти.

За целите на този тип тестване се използват няколко подхода:

- Отгоре-надолу (Top-down)
- Отдолу-нагоре (Bottom-up)
- Изолация (Isolation)
- Хибридно тестване (Hibrid Testing)

При първия подход наред с тестването на системната единица се извършва и интеграционно тестване. Започва се с единицата с най-високо ниво в йерархията, като единиците, с които тя комуникира биват заместени от фиктивни единици (stubs). Така за всяко следващо ниво има извикване на тествания компонент от единиците на по-горно ниво, които вече са тествани. Недостатък е, че промени в по-горните нива биха довели до нужда от повторно тестване.

Точно обратният подход се използва при тестването отдолу-нагоре. Започва се с най-ниското ниво в йерархията, като за комуникация с по-горните нива се използват драйвери (test drivers). Този метод е подходящ в случаите, когато се планира повторно използване на компонентите от по-

ниските нива, тъй като се спестява многократното им тестване за различните ситуации.

Подходът на Изолация дава независимост на последователността, в която се провеждат тестовете. При него всяка единица се тества в изолация от компонентите под или над нея в йерархията, използвайки едновременно и драйвери и стъбове. Това допринася за сигурност, че промените в компонента няма да се отразят на компонентите, с които комуникира.

Най-често използван е хибридния метод, който е модификация на методите отдолу-нагоре и изолация. При него единиците, които тестваният компонент извиква се заместват или с вече тествани единици или със стъбове.[14]

1.3.2. Интеграционно тестване (Integration testing)

Интеграционното тестване се извършва след като е преминало тестването на ниво системна единица. По време на интеграционното тестване, системата се изгражда чрез постепенното прибавяне на един или повече модули към ядрото от вече интегрирани модули. Групи от програмни единици се тестват пълно преди системното тестване. Тестването на интегирани модули е проектирано да намира скрити дефекти – например грешки в интерфейса и в базата данни. Тъй като досега са се тествали само индивидуални модули, не биха могли да се забележат няколко типа дефекти – смущения в базата данни, времеви конфликти, несъответствия в интерфейса, препокриване на използваната памет и т.н.. Те се откриват, когато модулите са принудени да работят заедно в интегрирани пакети. Интеграционните тестове използват както валидни съобщения така и невалидни условия и ситуации. Големият брой вариации от входни данни и начални условия изисква прилагане на техники за минимизация на броя изпълнявани тестове. Подходящо е и използването на автоматизационни инструменти.

Тъй като модулите са тествани по време на тестването на ниво системна единица, те могат да се разглеждат като “черни кутии”, което позволява на интеграционното тестване да се концентрира върху интерфейсите на модулите. Целта на интеграционното тестване е да се провери дали всеки модул изпълнява правилно ролята си в структурата и че интерфейсите на модула са коректни.

На всяка стъпка един модул се добавя към програмната структура и тестването се концентрира върху проверката на новодобавения модул. По този начин се улеснява откриването и изолирането на грешките допуснати в отделните компоненти. Когато се изпълнят предвидените тестове и не са открити проблеми или тези проблеми са вече решени, може да се премине към добавянето на нов модул.. Този процес продължава, докато всички модули се интегрират.

Документирането на резултатите от тестовете е много важно при интеграционното тестване. Необходимо е да бъдат документирани грешките, за да може да бъде направен анализ, с който да се локализират уязвимите места в софтуера и в неговото разработване. Необходимо е да бъдат запазени данни за конфигурацията на системата по време на интеграционното тестване, тъй като промените в конфигурацията често се отразяват на поведението на софтуера. След като всички резултати се анализират може да бъде определен статусът на проекта.[14]

1.3.3. Системно тестване (System testing)

След като приключи интеграционното тестване и всички функционални и технически изисквания са имплементирани, започва системното тестване. Въз основа на критериите за качество, описани в спецификацията с изисквания, се изготвят системни тестове. При този тип тестване системата се разглежда като едно цяло и се проверява изпълнени ли са всички функционални изисквания и как е реализиран бизнес процесът. За тази цел се използват както валидни така и невалидни данни. Необходимо е да се гарантира, че:

- системата не притежава нежелана функционалност
- на коректно подадени данни се връща очаквания резултат
- всяко бизнес правило може да се реализира чрез системата
- неверните данни се отчитат от системата и на потребителя се връща смислено съобщение за възникналия проблем
- до някой специфични участъци се достига само след определена последователност от стъпки
- системата се справя с обработката на големи количества данни
- може да оперира с определен брой заявки едновременно, като това не води до деградация на представянето

Проверката на тези изисквания (както и редица други) са цел на различните видове системно тестване – тестове за производителност, функционални тестове, тестове за съвместимост, тестване за сигурност, тестване за издръжливост, тестване на документацията, тестване на конфигурацията, регресионно тестване. За тяхната реализация се ползват и трите вида основни техники:

- Непрозрачното тестване (black box) е техника, която фокусира върху тестването на функционалността на програмата спрямо спецификациите.
- Прозрачното тестване (white box) е техника, при която пътищата на логиката се тестват, за да се определи колко добре те предоставят предвидим резултат.
- Полупрозрачното тестване(gray box) е техника, която комбинира тези два метода и често се прилага по време на системни тестове.

1.3.4. Тестове за приемане на системата (Acceptance testing)

Този тип тестване има за цел да потвърди, че изготвената система отговаря на нуждите на клиента. Извършва се едва след като софтуерът премине успешно всички системни тестове. Тъй като за крайния клиент не е важна конкретната избрана архитектура или метод за имплементация, софтуерът се третира като "черна кутия" и за тестовете не се използват знания за структурата му. Тестовете се планират изцяло на базата на изискванията на потребителя или клиента.

Една от особеностите на тестовете за приемане на системата е, че когато е възможно, се изпълняват от потребителите на системата. По този начин има възможност за откриването на проблеми свързани с интуитивността на навигацията или с възприемането на системата от крайния потребител. Хората, които не са запознати със системата, могат да въведат данни по неправилен, но технически възможен, начин. Могат да натиснат грешни бутони или правилни бутони, но в грешна последователност. Резултатът от тези неочаквани или неправилни ситуации е много важен за клиентите, които не искат тяхната система да се провали заради човешки грешки. Основно правило за всяка система е, че трябва да извършва функцията, която се очаква от нея. Това означава, че ако са извършени погрешни действия или са въведени некоректни данни, системата не само ще прекъсне работа, но и ще съобщи на потребителя какво е погрешно и ще му предостави възможност да повтори действието или въвеждането на данните. Невалидните данни, получени от външни източници, също трябва да се разглеждат по такъв начин, че да се попречи на пропадането на системата.

Важна част от тестовете за приемане на системата е да се потвърди, че новият софтуер не предизвиква промени в процеса на работа или отговорностите на потребителя.

Тестовете за приемане на системата са последната стъпка преди системата да стане притежание на потребителя или клиента. Важно е, че хората, носещи отговорност за качеството на софтуера и за управлението на конфигурациите, играят активна роля в проверката и изпълнението на тестовете, както и в управлението на промените на системата през този период.

Планът за приемане на системата дефинира процедури за изпълнение на тестовете; хората, които са включени в тестването; метрики определящи доколко софтуера отговаря на критериите; план за обучение на крайните потребители; процедура, по която се отстраняват новооткрити проблеми; протокол за приемане на системата. Тестването за приемане на системата продължава дори, когато са открити грешки, освен ако грешката не води до прекъсване на работа. Някои проекти не изискват формално тестване за приемане – например когато потребителят или клиентът е удовлетворен от

останалите системни тестове или когато крайният потребител е бил включен непрекъснато по време на цикъла на разработване.

1.4. Типове тестове спрямо изискванията към системата

Освен че различните видове тестове зависят от степента на завършеност на системата, те се разделят и спрямо каква функционалност от системата проверяват. По този критерий тестването може да бъде разделено на множество видове, не всички от които са валидни за повечето системи - някой от тях са специфични и се прилагат само в определени ситуации.

1.4.1. Функционално тестване (Functional)

Този тип тестване е най-широко разпространения в практиката и няма система, която да не се нуждае от него, тъй като е насочен към проверка на изискванията към функционалността на приложението на базата на спецификацията. При него няма нужда от информация за структурата на приложението и методите използвани за имплементацията му. Изцяло се разчита на непрозрачно (black box) тестване. Използват се различни техники като граничен анализ, класове на еквивалентност, таблици на решенията, за да се оптимизират тестовите данни и да се намали броят на извършваните тестове, но така че да се гарантира надеждността на системата. Има голям избор от инструменти за автоматизирането на този вид тестване.[30]

1.4.2. Регресионно тестване (Regression)

Регресионното тестване подобно на функционалното има за цел да открие проблеми свързани с очакваното поведение на софтуера. Разликата е, че при регресионното тестване се следи за новопоявили се проблеми във вече тествани части на софтуера, вследствие на отстраняването на предишни проблеми. Тъй като често функционалностите на една система са в зависимост помежду си, естествено е промените в една функционалност да се отразят на работата на друга. Ето защо се е наложило наред с повторното тестването на оправените вече проблеми, да се проверяват и области свързани с тези проблеми дори досега те да са функционирали правилно.

Този тип тестване трябва да се прилага след всяка промяна във функционалността, което от своя страна означава много време и усилия. Също така регресии могат да се наблюдават и при промяна на конфигурацията на системата. С цел улесняване на този вид тестване е прието един набор от основни тестове, обхващащи функционалността на системата, заедно с тестове, описващи наблюдавани, но вече поправени проблеми, да бъдат автоматизирани [7].

1.4.3. Тестване на потребителския интерфейс (User Interface)

При тестването на потребителския интерфейс от значение са детайли, които иначе не биха пречили на правилното функциониране на системата, но имат за цел да допринесат за удобството при работата на потребителя със системата. Необходимо е да се провери:

- има ли консистентност в използваните стилове – грешките се визуализират по един начин, важните съобщения по друг, а нормалният текст по трети
- използването на табулация обхожда ли в правилна последователност всички полета
- ако се използват икони или картинки – дали значението им е достатъчно интуитивно
- не трябва да има повтарящи се линкове, картинки, иконки ако се използват с различно значение
- навигацията трябва да е интуитивна и лесно запомняща се

1.4.4. Тестване на използваемостта на системата (Usability)

Подобно на тестовете за потребителския интерфейс, този тип тестване е насочен към начина, по който потребителят възприема системата. Целта е да се открие дали потребителят има проблеми с разбирането на инструкции и работата с функционалността на системата. Това може да се постигне чрез поставяне на реалния потребител в ситуация, в която да използва функционалността на системата. Изготвянето на тест за използваемост изисква внимателна подготовка на сценарии или реалистична ситуация, в която потребителят, следвайки инструкции и използвайки софтуера, изпълнява списък от задачи. За да се интерпретира доколко успешен е тестът е необходимо да се наблюдава колко бързо се ориентират хората в новата средата (използват се методи като “мислене на глас” и “проследяване на погледа”). Използват се и въпросници, с които да бъдат събрани впечатленията от тествания продукт. Една от методологиите, които се прилагат и за която се смята, че разкрива около 95% от проблемите в използваемостта, се нарича Холуей (Hallway testing) и се състои в избора на случайни 5 или 6 потенциални потребители на системата, на които се дава възможността да изпитат приложението.[18]

1.4.5. Тестове за производителност (Performance)

Този тип тестване спада към групата на нефункционалните тестове. За изпълнението им са необходими инструменти за автоматизация, тъй като се измерва време за отговор, натовареност на системните ресурси и други метрики изискващи прецизно отчитане. Основна цел е да се съпостави реалното представяне на системата спрямо нефункционалните изисквания към нея. Тази дейност би следвало да се извършва след като са приключили

функционалните тестове и системата е изчистена от значимите проблеми. В противен случай измерванията за представянето на системата могат да се окажат некоректни. Има няколко вида тестове, които имат за цел да измерят производителността на системата.

Тестовете за натоварване (load tests) отчитат изменението във времето за отговор от системата спрямо нарастването на потребителите (броя заявки, които се изпращат към системата). Целта е да се установи оптималното количество конкурентни връзки, при които приложението все още успява да се справи с поставените му задачи.

Тестовете с голям обем данни (volume tests) (големи файлове, голям брой отделни файлове, голям брой записи в базата) изследват как количеството обработвани данни се отразява на функционалността и бързината на системата. Тъй като обикновено тестовата база не е с голям обем, за приложения, които очакват поддръжка на голям брой записи в базата, се налага изкуственото запълване на базата (отделни таблици от базата) до предполагаемия обем, с който ще работи. [28]

До колко системата, която е разработена би издържала на промени, било то в натоварването или в обема обработвани данни, е предмет на тестването на възможностите за разширяване на приложението (scalability).

1.4.6. Стрес тестове (Stress)

Стрес тестовете следят поведението на системата в ситуации, с които тя не би могла да се справи. Ако реално е установено, че не повече от 200 потребителя ще ползват едновременно системата и тя успешно се справя с това количество, тогава какъв е смисълът от опитите с 2 или 3 пъти повече заявки? Смисълът е в това, че една система трябва да се справя успешно не само с предвидените, но и с непредвидените ситуации. Идеята е, че при претоварване не трябва да се губи контрол – може една част от потребителите да не получат желания резултат, но дали софтуера ще ги информира за временна недостъпност или ще получат системна грешка може да се окаже от съществено значение за бизнеса. В критични ситуации има опасност и от загуба на данни или повреждането им. При прекъсване на тока, спиране на мрежата или интернет връзката има реална възможност транзакцията, извършвана в този момент, да се изгуби, а заедно с нея и ценна информация. Стандартни ситуации използвани при стрес тестването са:

- изчерпване на дисковото пространство
- прекъсване на захранването
- претоварване
- заемане на ресурси от други програми

1.4.7. Тестове за възстановяване на системата (Recovery)

Наред със стрес тестовете необходимо е да се изпълняват и тези за възстановяване. Естествено е да се разчита, че в случай на извънредни обстоятелства водещи до прекъсване работата на системата, това ще има възможно най-малко влияние върху бизнеса. Освен поддържането на периодични копия на системата (backup) трябва да се осигури и минимална загуба на данни от последното копие до момента, в който настъпва проблема. Целта на тестовете за възстановяване е не да се изследва как реагира системата на екстремни ситуации, а да се проверят данните и функционалността след тези ситуации.

1.4.8. Тестване на сигурността (Security)

Този тип тестване е изключително важен, когато става въпрос за конфиденциални данни или финансови операции. Банкови, военни или държавни системи трябва да гарантират, че недоброномерен достъп до техните данни, както и възможността за намеси в тези данни, са изключени. Тестването на сигурността използва редица негативни сценарии, за да докаже, че няма оставени "вратички", през които нарушителят може да проникне до системата. Тази дейност включва проверка на правата за достъп, възможност за авторизация и автентикация, криптиране на по-важните данни в базата. Използват се техники, които са били прилагани успешно от "хакери", за да получат неправомерен достъп, като например атаки за блокиране на достъпа до даден ресурс (Denial of service attacks). Проблемите свързани със сигурността не зависят изцяло от пропуските на програмистите, имплементиращи системата – средата и технологията за разработка също съдържат в себе си потенциални пролуки. Поради тази причина се извършват проучвания за откритите вече проблеми в конкретната среда, инсталират се поправки от производителя (patch) или се разработват такива, и се поставят допълнителни бариери пред нарушителите, като се прикрива до колкото е възможно, използваната технология.

1.4.9. Тестване на различни конфигурации (Configuration)

Наред с поддържаните от системата хардуер, операционна система и допълнителен софтуер, който приложението ползва, трябва да се има предвид, че всеки един от тези компоненти от средата на системата може да има различни конфигурации. Като се умножава броят на различните конфигурации за всеки един компонент, се получава наистина голям брой потенциални ситуации, които трябва да бъдат проверени. На практика няма как това да се осъществи, затова се избират комбинации от различните компоненти, така че всяка основна конфигурация да присъства в поне една комбинация. Крайната цел на този тип тестване е да осигури сигурност, че

приложението ще може да работи и при различните хардуерни и софтуерни вариации.

1.4.10. Инсталационни тестове (Installation)

В по-голямата си част софтуерът пристига при клиента под формата на пакет, който той трябва да инсталира. Инсталиращата програма (wizard), често предлага възможност за избор на различни компоненти от крайния продукт, параметри и настройки. Някой от комбинациите, които се получават могат да се окажат недействителни (например ако има йерархично подреждане на компонентите, не би трябвало да може да се инсталира подкомпонент на някой компонент, който не е инсталиран), а други комбинации, въпреки че би трябвало да работят може да имат проблеми, които не са забелязани на по-ранни етапи. При инсталационните тестове е важно да се пусне поне по една инсталация за всяка смислена комбинация от параметри и настройки, както и да се провери, че на потребителя няма да се позволи да инсталира с невалидна комбинация или параметри. Необходимо е също така да се провери какви проблеми се получават при прекъсване на процеса. Изискванията налагат при проблеми с инсталацията, системата да бъде върната в първоначалното й състояние. Същото важи и за деинсталацията на продукта (единствената разлика са може би записите свързани с времеви лицензи, които би трябвало да останат и след деинсталиране). [21]

1.4.11. Тестване за съвместимост с други системи (Compatibility)

Този тип тестване има за цел да определи изискванията към хардуера и софтуера, от който системата се нуждае, за да работи правилно:

- минимална памет
- процесор
- операционни системи
- периферни устройства
- версии за софтуер, който използва

При изпълнението на тестовете е необходимо всеки един критерий да бъде изследван по отделно като за паметта и процесора е необходимо да се намери минимума, с който системата функционира, а за софтуера трябва да бъде проверена всяка една версия до текущия момент (или всички версии след някоя, която изискванията са поставили като минимум). [18]

1.4.12. Тестване за поддръжка на системата (Maintenance)

Поддръжката на една система включва справяне с проблеми възникнали по време на работата, преминаване към по-нова версия,

периодична проверка на състоянието, в което се намира системата. Целта на този тип тестване е да осигури, че системата ще изпълнява функциите си и ще запази производителността си и след определен период от време. [24]

1.4.13. Тестване на документацията (Documentation)

Смисълът на документацията е да услесни разбирането и работата с изготвения продукт. При тестването й е необходимо да се потвърди, че е разбираема, лесна за ориентиране, пълна и има възможност за бързо намиране на информация. Няколко са методите за тестване на документацията:

- Чрез демонстрация – Това е може би най-ефективния метод, при който се изпълняват стъпките в документацията, в реда и по начина, по-който са описани.
- Чрез симулация – Прилага се само, когато става въпрос за хардуер

Тестването на документацията трябва да се извърши от хора, имащи опит с документирането.
[11]

1.5. Значение на нефункционалните тестове

Успехът на една система зависи както от услугите, които предлага така и от редица допълнителни изисквания, които крайният потребител има, за да предпочете използването на една система пред друга. Нефункционалните характеристики като сигурност, производителност, съвместимост с различни среди, лесно използване изискват много усилия, за да бъде подобрено качеството им. Но именно разликата в тях изиграва голяма роля при избора на клиента.

Оставянето на нефункционалните изисквания за по-късен период от развитието на системата може да се окаже трудно поправима и много скъпа грешка. Изборът на архитектура и технология за имплементация трябва да е съобразен с нефункционалните изисквания към системата. За съжаление в спецификацията нефункционалните изисквания често не са ясно дефинирани, което от своя страна повлиява тяхната имплементация. [4]

Изпълнението на нефункционалните тестове често пъти се отлага за сметка на функционалните. При тестването на сигурността това би могло да се окаже и фатално. Защитата на информацията не бива да се оставя на последно място в тестването, особено в случаите, когато става въпрос за он-лайн разработване. Времето, през което системата не е защитена може да се окаже достатъчно за нейното компрометиране.

Тестовете с обемни данни също би трябвало да бъдат изпълнени в по-ранен етап от развитието на системата, тъй като проблемите, открити при тях водят до промени в дизайна или използваната технология. За да се осъществи този тип тестване е необходимо да има ясно дефинирани

очаквани резултати при определен обем от данни (времето за отговор и обработка на информацията), както и действителни данни или възможност за тяхното симулираме. Какви точно могат да бъдат данните и колко е максималното количество данни, така че то да отговаря на реалните нужди на системата, би трябвало да бъде определено от клиента съвместно с бизнес анализатора още във фазата на събиране на изисквания. [27]

Съвместимостта с различни операционни системи, други програми или хардуер оказва голямо влияние върху кръга от клиенти, които се очаква системата да има. За да се постигне тази съвместимост, трябва да се съобрази дизайнът. Ако проблемите бъдат забелязани късно, необходимите промени може да се окажат много големи.

Едни от малкото тестове, които трябва да бъдат изпълнени едва след като е осигурена известна стабилност във функционалността на приложението, са тестовете за натоварване. Целта им е да установят проблеми свързани с неправилното разпределение на ресурсите, липсата на оптимизация или недостига на ресурси. Наличието на грешки във функционалността може да възпрепятства имплементирането на тестовете за натоварване и най-вече да доведе до некоректни резултати. При тестовете за натоварване основните метрики, на които се обръща най-голямо внимание, са: времето за отговор, данните, които могат да преминат за единица време и времето, през което услугата е достъпна.

Достъпността се измерва като процент от времето, през което клиентите имат достъп до услугите на сървърното приложение. Стойностите на очакваните резултатите, които да потвърдят, че системата отговаря на нуждите варират в зависимост от характера на бизнеса, който системата обслужва. „Критичните” системи като например банкови, брокерски или такива, от които зависят човешки животи се нуждаят от 100% достъп до тях, докато за приложенията с не толкова „критично” значение може да се допусне и по-малък процент като приемлив. За да се осигури надеждността на тези системи, при тестването на производителността е необходимо да се проведат и тестове с по-голяма продължителност, при които системата да е подложена на постоянно натоварване в продължение на дни или дори седмици [3].

Значението, което нефункционалните тестове имат, е следствие от нарастващите изисквания на пазара. Борбата за клиенти води до стремеж към усъвършенстване на предлагания софтуер. Например при уеб-приложенията конкуренцията е твърде голяма и бързината и достъпа до една услуга могат да се окажат по-важни от колкото качеството й. Независимо колко разнообразна функционалност предлага един сайт, ако работата с него изисква твърде много усилия или пък не е лесен за ориентиране и ползване, той няма да добие голяма популярност. До колко е използваем един сайт също е обект на нефункционалното тестване. Наличието на консистентност,

интуитивност и удобство на интерфейса трябва да се следи през целия процес на разработка на приложението.

Въпреки че смисълът на едно приложение е заложен в неговата функционалност, не бива да се забравя, че същата или подобна функционалност може да бъде реализирана и в десетки други приложения. В такъв случай остават нефункционалните характеристики, които макар и не винаги да са така видими както функционалните често се оказват определящи при крайния избор на потребителя. За да се постигне усъвършенстване в тези изисквания понякога само усилията на тестерите не са достатъчни. Именно при имплементацията на нефункционалните тестове нуждата от автоматизация е най-осезаема.

1.6. Причини за автоматизация на тестването

Тестването изисква усилия, концентрация, време и ресурси. Всяко от тези изисквания означава повече пари. За бизнеса е важно разходите да бъдат сведени до минимум, поради тази причина намаляването на коя да е от изброените нужди на тестването е търсена печалба. Автоматизацията на тестовия процес е може би най-ефективното налично средство. Тя не може да замести напълно ръчното тестване, но има редица преимущества, които я определят като необходимост и неразделна част от съвременния тестов процес. [5]

Едно от най-големите преимущества е възможността за изпълнение на едни и същи тестове върху различни версии на продукта. Това е може би и най-често срещаното използване на автоматизацията. То се налага особено в случаите, когато има чести промени в тествания софтуер.

Друго предимство на автоматизацията е изпълнението на по-голямо количество тестове и по-често. Автоматизираният тест протича по-бързо отколкото ръчното му изпълнение - по-малко време за мислене, няма разконцентриране, стъпките са ясно дефинирани и последователни. Спестяването на време от всеки тест по отделно позволява за времето, което един тестер проверява един компонент, да бъдат пуснати много повече тестове за същия компонент като по този начин се осигурява по-голяма надеждност на продукта.

Понякога автоматизацията може да е единственото решение, тъй като има тестове, който ръчно не биха могли да се изпълнят или се осъществяват трудно. Да се провери един сайт за натоварване означава да се наблюдава поведението му при едновременна работа на стотици потребители. За ръчното осъществяване на тази цел трябва да се заеме от времето например на 100 човека, всеки от които има нужда от ресурси, които да ползва при тестването и да се синхронизира работа на тези хора. Ако няма проблеми и сървърът се справя успешно, би могло да се измери натоварването на ресурсите му и евентуално времето за отговор на един от потребителите. Ако, обаче, има проблеми, проследяването им би се оказало непосилна

задача, както поради броя на едновременните заявки така и поради факта, че сценариите, които различните потребители ползват могат да се разминават. Всички тези проблеми не съществуват, когато се използва инструмент за автоматизация. Една машина е достатъчна за симулирането на 100, 200 или повече едновременни сесии. Времето за отговор на всяка заявка може да бъде пресметнато и да се получи средна стойност. Може да се наблюдава изменението в поведението на софтуера спрямо нарастването на броя на сесиите и да се установи с голяма точност прагът на натоварване [25].

Автоматизацията помага и за по-рационално използване на ресурсите. Спестява се от времето за работа на тестерите, което от своя страна им предоставя повече време за концентрация върху тестовите сценарии и ситуациите, които да обхванат. Също така се спестява и от ресурсите – тестовете могат да се пускат нощно време, когато никой друг не ползва машините.

Благодарение на скриптовете се постига консистентност и лесна повторяемост на тестовете. Автоматизираните тестове изпълняват едни и същи стъпки всеки път, което гарантира лесно възпроизвеждане на намерените проблеми. Едни и същи тестове могат да се пуснат в различни среди – различна операционна система, различна база, което дава сигурност в консистентността на продукти предназначени за различни платформи.

Друго предимство е повторно използване на тестовете. Усилията, положени в решението какво да се тества, в планирането и записването на тестовете, са функция на броя на изпълненията на теста и цената за едно изпълнение – колкото повече пъти е използван, толкова по-ниска става цената му. За това в конструирането на тестовете, които се знае, че ще бъдат използвани многократно, се влага повече време [25].

Автоматизацията може да помогне за по-ранно пускане на софтуерния продукт на пазара. По-малкото време необходимо за изпълнение на автоматизираните тестове допринася за намаляване на времето необходимо за цялостното тестване.

Още една причина за използването на автоматизация е, че тя дава увереност в разработената система. Възможността да се разшири наборът от тестове при използването на автоматизация допринася за по-голямата сигурност, че няма да има пропуснати грешки.

Възможността за ефективно използване на автоматизацията на тестването се обуславя и от начина на разработка на софтуера. Итеративните методи изискват регресионни тестове, които ако се автоматизират спестяват време и ресурси. При клиент-сървър системите необходимостта от автоматизация на нефункционалните тестове е очевидна. За приложения, в които е от съществено значение точното разположение на графичните обекти по екрана (програми за чертане, печат, рисуване), за да се запишат координатите им, също трябва да се използват инструменти.

Идеята за автоматизация може да се окаже колкото полезна толкова и неоправдана понякога. Не всеки тестов сценарий е подходящ за автоматизация. Преди да се започне автоматизирането на процеса е необходимо да се прецени каква ще е възвращаемостта на тестовете, които ще се автоматизират, тъй като самата автоматизация не е бърз процес и често времето за автоматизация на един сценарии може да отнеме 5 до 10 пъти времето необходимо за ръчното му изпълнение.

1.7. Избор на инструмент за автоматизация

Основен риск при автоматизацията освен неправилния подбор на тестови сценарии, е възможността избраният инструмент за тестване да не може да се справи с поставените задачи. За да се избегне тази възможност е необходимо да се извършат няколко дейности.

1.7.1. Определяне на изискванията

Много са изискванията към инструмента за автоматизация. Някои са свързани с конкретни проблеми, които би трябвало да се решат с помощта на автоматизацията. Други касаят технически и нетехнически ограничения върху избора. За да може да се определи най-подходящият инструмент, трябва да се започне именно с дефинирането на изискванията, спрямо които ще се оценяват всички инструменти [29].

Започва се с изготвяне на списък от проблемите, които се очаква да бъдат решени чрез автоматизация. Точното описание на проблемите гарантира, че всички включени в процеса ще знаят каква точно е задачата и критериите за успешно преминала автоматизация няма да се разминават. Някои често срещани проблеми от списъка са:

- проблеми при ръчно тестване (допускат се грешки, твърде е бавно)
- недостига време за регресионните тестове
- не е ясно какъв процент от продукта е минал тестове
- в подготовката на тестовите данни се допускат грешки

Друг основен критерий е цената, която може да се плати. Дори бюджетът да не е ограничен, преди да се закупи един продукт е необходимо да се пресметне стойността на ползите от него и за какъв период би се изплатил.

1.7.2. Определяне на ограничаващите условия

Ограничаващите условия помагат да се отхвърлят голям брой от автоматизиращите инструменти предлагани на пазара. Отхвърлянето на даден инструмент още в самото начало спестява много време и пари, които биха били загубени в по-детайлното му проучване.

Едно от първите ограничаващи условия може да е средата (софтуерна или хардуерна), в която автоматизиращия инструмент би трябвало да работи.

Ако той не поддържа средата, в която се разработва тестваният проект, твърде вероятно е допълнителните му настройки (ако такива въобще могат да се направят) да струват твърде скъпо или да отнемат много време. Възможно е, обаче, дори инструменти поддържащи необходимата среда, да изискват допълнителен хардуер или софтуер, с което да обезсмислят понататъшното им проучване.

Има и ситуации, в които не е необходимо и продуктът и автоматизиращият инструмент да се намират в една среда (например при тестовете за натоварване машината, от която се извършват тестовете не се влияе от средата на сървъра, към който се отправят заявките). До колко средата е определящ фактор при избора, изцяло зависи от целите, които трябва да бъдат реализирани.

Произходът на един инструмент също може да повлияе върху оценката му. Ролята на фирмата производител може да се окаже съществена в използването на този инструмент, за това няколко са нещата, които трябва да се проучат за нея:

- колко зряла е организацията и нейния продукт
- осигурява ли се техническа поддръжка и до колко надеждна е тя
- колко други организации са закупили този инструмент
- какви са отзивите от компаниите, които го ползват
- от колко време е на пазара
- колко често се обновява
- има ли нерешени проблеми по него, които са били описани

Крайната цена, която се плаща за дадения продукт, също е решаващ фактор. При пресмятането й трябва да се имат предвид следните възможности:

- в цената включен ли е постоянен лиценз или трябва да се подновява през определен период от време
- има ли ограничения за брой компютри или потребители, които да го ползват
- цена на обучението за работа с него
- има ли допълнителен софтуер или хардуер, от който да се нуждае
- цена на поддръжката
- вътрешни разходи (за поддръжката, обучението и прилагането на закупения инструмент)

Качествата в спецификацията на инструмента, на които трябва да се обърне внимание са:

1. колко максимално потребители могат да го използват едновременно
2. може ли да се споделят тестовете
3. каква опитност се изисква, за да може да се работи с него

4. колко време се изисква, за да се добият детайлни познания за него
5. какви програмистки умения се изискват, за да се пишат скриптовете
6. има ли хубава документация
7. какви помощни средства предлага
8. с каква честота се получават грешки при използването му
9. има ли шанс да повреди тестовите данни
10. каква част от ресурсите използва за себе си
11. има ли възможност за интеграция с други инструменти, които вече се използват

[5]

1.7.3. Закупуване или разработване на самостоятелен инструмент за тестване

Ако след проучване на пазара не е намерен нито един инструмент, който да покрива изискванията, може да се предпочете изработването на собствен инструмент. Преимущество е, че ще е разработен специално за нуждите на системата, която се тества. При вътрешното разработване на подобни инструменти голяма част от функционалността на комерсиалните продукти липсва, както и не се обръща внимание на интерфейса и сложността за работа. Цената може да се окаже доста висока в зависимост от сложността на поставените задачи. Има опасност готовият продукт да е строго специфичен и да не може да се използва повторно при други проекти.

Друг вариант за решаване на проблема може да е закупуването на един или няколко инструмента и допълнително кодиране така, че да се покрият максимално нуждите на проекта.

1.7.4. Стесняване на крайния списък с потенциални инструменти за автоматизация

Необходимо е да се изготви допълнителен списък с характеристики, които биха намерили приложение при тестването на системата. Тези характеристики след това трябва да се организират по групи в зависимост от необходимостта им – задължителни, полезни и без значение. След като се проучат възможностите на инструментите в списъка броят им трябва да се сведе до 3-4, с които да се направят реални тестове.

1.7.5. Вземане на решение

За да се вземе крайното решение е необходимо да се изпробва всеки един от предлаганите инструменти. Един от методите е чрез използване на прототипи. Съставят се няколко сценария с различна сложност и се имплементират, използвайки инструментите, които трябва да бъдат оценени. В случай, че нито един инструмент не отговаря напълно на нуждите на

тестването, може да се използва комбинация от няколко инструмента, с които да се покрие максимална част от плана. Единственото решение за частта, която остава, е да се разработи допълнителен код, който да допълни тяхната функционалност (в редки случаи изцяло да замести съществуващите инструменти).

Определянето на подходящия инструмент за тестване е свързан с поредица от дейности, които имат за цел да анализират нуждите на системата и ограниченията породени от бизнеса и да ги сравнят с продуктите предлагани на пазара. Решението кой е най-подходящият и дали той заслужава инвестицията зависи от очакваната печалба, която възможностите на инструмента за автоматизация биха донесли на бизнеса.[5]

1.8. Разработване на софтуер на базата на изготвени тестове (Test-Driven Development)

При тази техника (TDD) се използва първо написване на тестовия сценарий, а след това имплементиране на кода, необходим за изпълнението на теста. По този начин се осигурява бърз отговор за възможностите на системата. За пръв път тази техника се споменава през 2000 година като част от Екстремното програмиране (Extreme programming), но все по-често се отнасят към нея като самостоятелна технология, подпомагаща разработването на дизайна за софтуера, а не само като метод за тестване.

Разработването на софтуер на базата на изготвените тестове, изисква автоматизация на ниво системна единица, определяща изискванията към кода преди той да бъде написан. Изготвените тестове осигуряват бързо потвърждаване на коректността на системата по време на изготвянето и преправянето на кода.

Цикълът на разработки при тази технология има пет фази:

1. Добавяне на тест – Тестът, който се добавя включва нова функционалност. Необходимо е програмистът да е добре запознат с изискванията към новата характеристика.
2. Изпълнение на всички тестове и получаване на грешки от нововъведения – Тази фаза осигурява, че написаните досега тестове функционират правилно и че последния тест ще се провали, тъй като не е бил имплементиран специален код за него. За негативните тестове, които се изпълняват е необходимо да се провери, че връщат правилната грешка.
3. Писане на още код – За да може да премине последния добавен тест успешно, трябва да се напише код, който покрива неговата функционалност. В тази фаза не се набляга на качеството на кода, тъй като за него отговаря последната фаза.
4. Изпълнение на всички тестове с цел да преминат успешно – В случай, че няма проблеми при изпълнението на тестовете,

програмистът разполага с добра основа, на която да изгради качествения код.

5. Преработка на кода – Това е последната фаза, в която кодът се изчиства или замества с нов, но така че да не се увреди съществуващата вече функционалност.

Ползите, установени при използването на TDD, са по-рядката необходимост от дебъгване (търсене и отстраняване на дефекти в кода) и възможността по-бързо да бъде разработен софтуерът. Друго предимство е факта, че размерът на стъпката, използвана при разработване, може да бъде променян – когато се налага стъпката може да обхваща много по-малко детайли. Използването на TDD води до модуларизация, гъвкавост и разширяемост на кода.

Някой от недостатъците, които крие в себе си тази технология са възможността крайният код да се окаже с ниско качество, ако не се използва в комбинация с редовни прегледи на кода (code review) и сложността му при използване в някой ситуации като графичен интерфейс и релационни бази данни. Коректността на дизайна и изпълнението се базират изцяло на разработените тестове, затова присъства и рискът някой от тестовете да не отговаря на нуждите или да се отклонява от исканата функционалност.

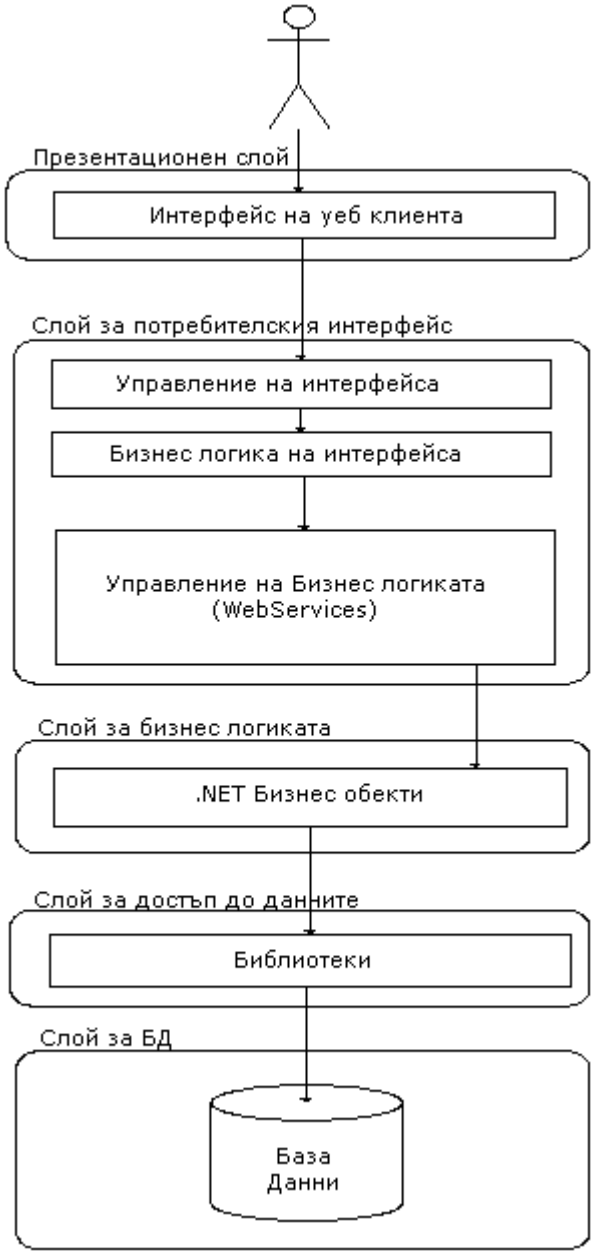
Тъй като всяка техника наред с положителните си качества притежава и негативи, преди да се премине към тестването, е необходимо да се знаят особеностите на избраната архитектура и техниките, включени в нейната имплементация. Поради тази причина следващата глава описва характеристиките на избрания проект.

2. Описание на функционалността и изискванията към проекта

Проектът, който ще бъде разгледан, е реализация на цялостния бизнес процес на фирма за потребителско кредитиране. Обхваща както административни дейности - работа със служители, техните позиции, организациите, в които участват, така и процеса свързан с обслужването на клиентите - заявяване на желания продукт, одобрение, усвояване на продукта и погасяване на дълга към компанията.

2.1. Общо описание на техническите характеристики на проекта

Проектът използва многослойна архитектура, която се състои от Презентационен слой, Слой за управление на потребителския интерфейс, Слой на бизнес логиката, Слой за достъп до данните и Слой за базата данни (Фигура 5). За базата данни е избран MS SQL Server 2005. Достъпът до базата се осъществява чрез .netTiers, който се използва за генериране на заявки и процедури към базата, генериране на обекти и техните връзки въз основа на таблиците от базата. При реализацията на бизнес логиката се ползват ASP.NET 2.0, Windows Services, Windows Workflow Foundation. Управлението на потребителския интерфейс се реализира чрез IIS 6.0, WebServices. Самият потребителски интерфейс е изграден на базата на AJAX, JavaScript, XHTML 1.0, CSS.



Фигура 5: Архитектура на проекта

2.2. Използвани технологии

Както бе описано в предната точка приложението използва следните технологии:

ASP.NET 2.0 е платформа (framework), предлагана от Microsoft за разработване на динамични уеб приложения. Традиционните уеб страници могат да изпълняват код на клиента, с който извършват сравнително прости операции. ASP.NET уеб формите могат да изпълняват и код на сървъра (server-side code), където се генерира HTML и се изпраща като отговор на заявката. Има възможност да се извършват обработки, изискващи достъп до

бази от данни и до ресурсите на самия сървър, генериращи допълнителни уеб форми. Всяка уеб форма в крайна сметка се трансформира в HTML код, пригоден за типа на клиентския браузър. Това позволява улеснена разработка на уеб форми. Те работят практически върху всяко устройство, което разполага с интернет свързаност и уеб браузър. Целта на ASP.NET е да осигури по-добро представяне, като компилира кода за сървъра на един или повече dll файла върху уеб сървъра. Тази компилация се извършва автоматично първия път, когато се заяви страницата. Ресурсните файлове се поставят на виртуален хост в IIS и при това първо извикване на страницата .NET Framework компилира файла и изпраща отговор. Всяка следваща заявка се обслужва от .dll файловете. Едно от решенията на забавянето, което тази първа компилация предизвиква, е да се прекомпилират файловете преди да бъдат внесени в продуктовата среда.

Предимствата на ASP.NET са бързината, шаблоните и огромната библиотека, които осигурява, възможност за кеширане на цели страници, с което да се подобри представянето, възможност за разделяне на бизнес логиката от презентационата, валидация на HTML и XHTML.

Критиките към ASP.NET се дължат предимно на зависимостта на някои от неговите контроли от JavaScript и тясната свързаност на ASP.NET с IIS. Има някои версии на продукта (ASP.NET – Mono), които могат да работят на Apache, но те не съдържат пълния набор функционалности.

.netTiers е група шаблони за генериране на код, които улесняват разработването на Приложния слой за продукти, написани с Microsoft .NET. Те са изработени с помощта на CodeSmith Tools, който е най-мощния инструмент в момента на пазара за генериране на код. Някои от предимствата им са, че:

- генерира решение напълно съвместимо с разработваното приложение
- създава пълен комплект процедури за работа с базата данни, съобразени с архитектурата на домейна
- автоматично генерира същностите и техните връзки на базата на таблиците от базата данни
- създава пълна група от административни уеб контроли, служещи за основа на уеб административната конзола
- генерирания код е с коментари и може да се използва за целите на документацията

Наред с плюсовете, .netTiers притежават и някои минуси, като например:

- както при всеки универсален генератор на код, липсва гъвкавост
- налага някои недокументирани ограничения в именуването на обектите в базата данни
- изчитането на сложна структура от данни, разпределена в множество свързани таблици в релационна база от данни, може да се окаже изключително бавна, когато се използват инструментите на .netTiers.

Windows Workflow Foundation е технология на Microsoft за дефиниране, изпълнение и управление на бизнес процес (Workflow). Този процес се състои от дейности, описващи различните му фази. Данните, които тези дейности ползват, могат да бъдат свързани с процеса чрез деклариране на зависимости. Windows Workflow Foundation дефинира интерфейси за методите и събитията, с които те да комуникират с външни компоненти. При поява на „събитие", процесът съответстващ на него се задейства и му се подават данните, получени с появата на събитието.

AJAX е техника за разработване на интерактивни уеб приложения. Идеята е да се направят страниците, така че да не се налага презареждането им всеки път, когато потребителят поиска промяна. По този начин се увеличава бързината, функционалността и използваемостта на приложението. Програмният език, в който се извикват функциите на AJAX, е JavaScript. Данните, които се получават при тази техника обикновено се форматират като XML.

AJAX е техника, която не е зависима от платформата, на която се използва (операционна система, компютърна архитектура, браузър). Това е само едно от предимствата на тази технология. Друг основен плюс е разделянето на данните, форматирането, стила и функциите.

Минусите се дължат на забавянето, причинено при първото зареждане на данните, както и пренареждането, което се извършва в момента, когато всички данни са заредени. Известни проблеми се наблюдават и при различните типове и версии на браузърите, тъй като JavaScript се имплементира различно на всяка от тях.

Windows Service е приложение, което се стартира заедно с операционната система и работи като фонов процес. Windows Services приложенията са много подходящи, когато става въпрос за сървърни приложения. Те нямат потребителски интерфейс и не връщат визуализирани данни. Съобщенията, които се подават от тях към потребителя, обикновено се записват в Windows Event Log[20]. Контролът над Windows Services може да се извършва от Service Control Manager, където има възможност за тяхното спиране, паузиране или стартиране, както и за промяната на типа им на стартиране на автоматичен (стартира се заедно със зареждането на операционната система) или ръчен (стартира се ръчно – чрез SCM или конзолна команда.

За пръв път Windows Services се появяват в операционната система Windows NT. Примери за използването им са Microsoft Exchange, SQL Server, Windows Time [23]. В зависимост от това чий профил се използва за стартирането им, се определят и правата, с които Windows Services разполагат.

Предимствата при използването на Windows Services в сървърни приложения са:

- автоматичното стартиране на процеса заедно с операционната система
- възможността за автоматично възстановяване на процеса при спиране поради грешка
- евентуално увеличаване на производителността, дължащо се на приоритета, който Windows Services получават от операционната система
- Всички грешки и предупреждения се записват в Windows Event Log

Недостатък може да бъде неправилното определяне на права за процеса в зависимост от профила, под който е бил стартиран, особено когато машината, на която работи Windows Service приложението, има много потребителски профили [22].

Според W3C (World Wide Web Consortium) **Web Services** е „софтуерна система, проектирана да поддържа взаимодействието през мрежата между две машини с различни операционни системи". Това определение най-често се интерпретира като комуникация между клиент и сървър чрез XML, използвайки SOAP стандарта [19].

Web Services имат различни видове приложение. Най-често срещани са RPC (Remote Procedure Calls), SOA (Service-Oriented Architecture), REST (Representational State Transfer).

RPC е технология, която дава възможност на една програма да изпълни процедура, намираща се в друго адресно пространство, без да е необходимо програмистът експлицитно да добавя код, реализиращ детайлите по операцията. Комуникационната единица, която тази технология ползва, е WSDL операция. RPC е широко разпространена и поддържана. Недостатък е, че при реализацията ѝ често Web Services се обвързват с функции на конкретен програмен език [15].

SOA е архитектура, при която основната единица за комуникация е съобщение, а не операция. Тя също е поддържана от много софтуерни производители, но за разлика от RPC по-рядко се обвързва с конкретна имплементация [17].

REST приложението на Web Services служи за симулиране на HTTP и подобни протоколи, ограничавайки интерфейса само до група основни операции. Не използва съобщения или операции за комуникацията, а ресурси, притежаващи различни състояния. Основен минус е сложността и обвързаността с големи софтуерни производители и липсата на имплементации със свободен код [16].

Като цяло основните предимства при използването на Web Services са:

- Независими са от конкретен програмен език
- Поддържат се от много инструменти, поради широкото си приложение

– Имат възможност за многократно използване

За недостатък на Web Services се посочва по-малката ефективност, дължаща се на използването на XML вместо двоичен код. Друг основен недостатък, наследен от използването на XML, се откроява при наличието на циклични връзки между обекти.

2.3. Функционални изисквания към проекта

Основните функционални изисквания към системата са свързани с необходимостта от цялостно покритие на бизнес процеса.

Необходимо е системата да отразява структурата на компанията – нейните подорганизации и отдели, както и бизнес партньори. Поддържа се йерархична организация. За всеки елемент от нея се дефинират различни функционалности и се поддържа детайлна информация. Изтриване на елементите не се позволява, затова се въвежда статус за активност, който забранява работата с неактивните елементи.

Друго изискване е управлението на потребителите. То включва въвеждане и промяна на личните данни, назначаване на определена позиция към организация, възможност за повече от една позиция, за деактивиране на позиции, за създаване на потребителски профил, с който служителят да влиза в системата, допълнителни операции върху самия акаунт. Отново не се позволява изтриване, а само деактивация.

Административната част на системата управлява и параметрите на предлаганите продукти, както и документите свързани с всяка отделна продажба.

За всяка една продажба се поддържа жизнен цикъл – подаване на молба, оценяване на молбата, отхвърляне или одобряване, следене на плащанията по нея в случаите, когато е одобрена, и приключване след като дължимите суми бъдат изплатени.

Всички плащания към банки и търговски партньори се пресмятат от системата. При получаване на плащания от клиенти те се разпределят автоматично по кредитите им или ако има проблем се отделят за ръчна обработка.

Системата предлага статистики и справки свързани с дейността на компанията.

Наред с функционалните изисквания се поставят и редица нефункционални необходими за успеха на бизнеса.

Разработваната система трябва да притежава високо ниво на сигурност. При вход на потребител в системата се отчитат неговите позиции и роли и спрямо тях му се предоставят различни функционалности. Действията на автентикиралите се потребители се пазят под формата на файлове. Неуспешните опити за автентикация се отразяват в системата и водят до блокиране на потребителите при няколко неуспешни опита.

Бързината при работа със системата и подредеността на данните, обработвани от служителите, са изисквания, които обуславят по-качественото обслужване на клиентите на компанията.

Големият брой на извършваните продажби, изисква издръжливост на системата при натоварване. Трябва да се има предвид, че при определени промоции или сезонни празници търсенето може да нараства няколкократно и невъзможността да се поемат всички заявки може да доведе до големи финансови загуби.

2.4. Описание на функционалността, предмет на експеримента, тестовата среда и избрания подход

Тестовете за натоварване изследват поведението на системата при получаването и обработката на молби за кредит. Административната част на проекта включва дейности с малка повторяемост – добавяне на нови организации, промяна на статуса им, корекция на детайлите им, промяна в йерархията на организацията, работа със служителите и техните характеристики. Поради тази причина тяхното включване към тестовете за натоварване не е необходимо, тъй като слабо би повлияло на крайните резултати, а и в практиката тази дейност може да бъде отложена за непикови часове.

Друга част от функционалността, която не оказва пряко влияние върху исканията за бързина на системата, е обработката на получените плащания по кредитите. Тази дейност се извършва в неработно време и може да трае по-дълго без това да навреди на бизнес процеса.

Най-критични за производителността на системата са пристигащите по интернет и по факс молби, които трябва да бъдат обработени в срок от 15 минути, тъй като това е едно обещанията на компанията към нейните клиенти. По-голямата част от натоварването се дължи на попълнените през интернет молби. При тях реално няма ограничение на максималния брой молби, които могат да постъпят, тъй като местата, предлагащи услугите на компанията в цялата страна са над 4000. Единствено статистиката от дългогодишната практика на фирмата може да предложи разумна максимална стойност за тази част от функционалността. При другия тип молби – постъпващите по факс, има известен брой служители, които едновременно да ги въвеждат в системата. Същото важи и за одобряването на молби, което се извършва в централния офис, само от определен брой хора.

Тестовата среда при провеждане на експериментите се състои от два клъстъра, между които се поделя натоварването, и база данни с над 1 000 000 записа. Комуникацията със сървъра се извършва чрез локалната мрежа.

За провеждане на тестовете за натоварване се използва Mercury LoadRunner 8.0, който е подробно описан в точка 4.1. При записването на

скриптовете, които изпълняват подаването на четирите вида молби за различните продукти и одобрението им, се параметризират потребителите, които извършват операциите и подаваните данни. За да се приближи сценарият максимално до поведението при реалната система, се използват статистически данни от изминалите години. По този начин се определя процентното съотношение на отделните функции, които ще бъдат тествани и средното време, което отнема извършването им.

За планирането на тестовия процес от огромно значение е избраният подход. Има редица техники, които се използват за контролирането на качеството в продукта като например “Six Sigma” (“6 Сигма”) и PDCA (Plan-Do-Check-Act – Планиране-Изпълнение-Проверка-Действие). Тези техники, обаче, се прилагат предимно при големи проекти, включващи много човешки ресурси и често се оказват неподходящи за по-малкия бизнес. Поради тази причина е избрана техниката GQM-MEDEA (Goal-Question-Metric – MEtric DEfinition Approach), която се води изцяло от бизнес целите при определяне на необходимите метрики за измерванията и не се влияе от размера на бизнеса. Тя е описана подробно от Парк в книгата му [10], а приложението й за конкретния проект е поместено в следващата точка.

3. Планиране на тестовете за натоварване

Преди да започне планирането на тестовете е необходимо да се знае каква всъщност е тяхната цел и как ще бъде измерено дали резултата е приемлив или не е.

Според Парк[10] основните схващания, на които се базира процесът за определяне на необходимите метрики, са три:

1. Целите при измерванията се извличат от бизнес целите
2. Произхождащите от тях мисловни модели осигуряват контекст и фокус
3. Определянето на целите заедно с въпросите, произлизащи от тях, превръща неформалните цели в изпълними структури от измервания

Десет са стъпките, следвани при този процес:

1. Определяне на бизнес целите
2. Определяне на необходимите знания, свързани с постигането на бизнес целите
3. Определяне на подцели
4. Определяне на същностите и атрибутите свързани с тези подцели
5. Формализиране на целите на измерването
6. Определяне на въпросите за количеството и свързаните с тях индикации, които ще бъдат използвани за постигането на целите на измерването
7. Определяне на данните, които са нужни за съставянето на индикаторите, с които да се отговори на поставените въпроси
8. Дефиниране на измерванията, които ще се ползват
9. Определяне на действията, които ще бъдат извършени за реализацията на измерванията
10. Подготвяне на план за реализация на измерванията

3.1. Определяне на бизнес целите

На базата на извършени интервюта с клиента и обсъждане на поставените изисквания с хора, които са част от бизнес процеса, могат да се дефинират следните цели, които се очаква бизнесът да удовлетвори след завършването и интеграцията на проекта:

1. разрастване на територията на бизнеса
2. разрастване на структурата на бизнеса
3. промени в структурата на бизнеса
4. увеличаване на предлаганите от бизнеса услуги
5. централизация на финансовите дейности

6. увеличаване на ефективността на бизнеса

3.2. Определяне на необходимите знания, свързани с постигането на бизнес целите

За да бъдат постигнати бизнес целите е необходимо да се установят дейностите, водещи до тяхното осъществяване. За всяка от определените в предишната точка цели трябва да бъдат подбрани въпроси, с които да се отговори как точно разработваната система ще достигне до тази цел.

1. Какво може да се включва в разширението на територията на един бизнес?
2. Има ли ограничения за разрастването на структурата, като например отдели или подорганизации, които не могат да притежават собствени подорганизации или имат ограничен брой такива?
3. Какво може да включват промените в структурата на бизнеса? Може ли да се изтриват съществуващи структури? Може ли да се добавят нови функционалности към съществуваща структура? Възможно ли е разместване в йерархията на структурите?
4. Свързано ли е увеличаването на услугите, предлагани от бизнеса, с допълнителни характеристики, които сегашните продукти не притежават? Какви възможности трябва да притежава софтуерът за добавянето на нов продукт?
5. Какви финансови дейности трябва да извършва софтуерът? От къде трябва да се вземат данните за тях? Какви ограничения има при работата с финансите?
6. Какви са критериите за ефективност на бизнеса (по-добра организация на работата, по-голяма печалба, по-малко разходи, възможност за анализ на постигнатите резултати)? Какво е необходимо да осигури системата, за да гарантира постигането на тези критерии (бързина, лесна за използване, сигурност на данните)?

3.3. Определяне на подцели

На базата на получените отговори на въпросите, свързани с бизнес целите, могат да се определят подцелите, които приложението трябва да постигне. Една част от подцелите са свързани с функционалните изисквания, а друга част дефинира нефункционалните. За целите на дипломната работа ще бъдат разгледани само определените подцели свързани с нефункционалните изисквания.

За получаването на реално изпълними подцели е необходимо въпросите да бъдат групирани спрямо частта от продукта, която засягат. Следният

списък съдържа нефункционалните характеристики, заедно с въпросите насочени към тях.

1. Сигурност – Има ли възможност за проникване на външни лица в системата и компрометиране на данните, с които работи? На всеки служител с достъп до системата дават ли се само правата определени от позицията му? Виждат ли служителите функционалности, които нямат право да извършват? Съхраняват ли се данните криптирани?
2. Надеждност – Колко може да издържи системата без да прекъсва работа? При проблеми със софтуера има ли шанс данните да бъдат повредени? Запазват ли се данните редовно така че при проблеми да бъдат възстановени?
3. Издръжливост – Може ли да се поеме натоварването в системата в пиковите часове и периоди? Може ли да се справи с обработката на голямо количество данни?
4. Удобство – Подреден ли е интерфейсът така, че да има лесен достъп до най-необходимите функционалности? Данните на екрана подредени ли са логически? Има ли възможност за бърза работа само с клавиатурата?
5. Разширяемост – Има ли възможност за поддържане на повече от един език? Има ли възможност за добавяне на нови региони? Има ли възможност за поемане на натоварването свързано с разрастване на бизнеса?

Така групирани въпросите дават възможност да се определят подцелите, свързани с различните области от софтуера:

1. Да се увеличи сигурността на данните
2. Да се ограничат функционалностите, разрешени за служителите
3. Да се увеличи производителността на системата
4. Да се изчисти дизайнът и да изведат най-често използваните функционалности на по-лесна позиция, както и да се създадат навигационни връзки между областите, които са свързани логически

3.4. Определяне на същностите и атрибутите, свързани с тези подцели

Разполагането с подцелите и свързаните с тях въпроси позволява да бъдат определени мисловни модели заедно със същностите, които ги изграждат, и техните атрибути.

Да се увеличи сигурността на данните. От тази подцел данните са същностите, а атрибутите, които те имат, са размер, тип, съдържание.

Да се ограничат функционалностите, разрешени за служителите. Същностите свързани с тази под цел са служители, функционалности, позиции на служителите, роли. Атрибутите за служителите са брой позиции и роли, териториален обхват, активност. Функционалностите притежават достъп до определени данни и права за данните. Позициите притежават списък от роли. Ролите определят достъпа до функционалностите.

Да се увеличи производителността на системата. Производителността се определя на базата на общото представяне на различните компоненти. Същностите тук са точно тези компоненти – административна част, финансова част, търговска част, статистическа част. Атрибутите, които характеризират административната част, са брой функционалности, брой потребители, време за извършване на различните функционалности, количество данни, до които има достъп. За финансовата част определящи са големината на данните, с които работи, броя функционалности, пълнотата на данните. Търговската част се състои от брой продукти, брой служители, бързина за извършване на различните функционалности. Статистическата част се изгражда на базата на количество данни, броя на критериите, по които могат да се групират тези данни.

Да се изчисти дизайна и да изведат най-често използваните функционалности на по-лесна позиция, както и да се създадат навигационни връзки между областите, които са свързани логически. Същностите тук са менюта, препратки, икони, клавиши за по-бърза работа. Менютата се характеризират с дълбочина или позиция в йерархията, честота на използване. Препратките, както и клавишите за бърза работа също имат честота на използване. Иконите препращат към определена функционалност и имат степен на интуитивност.

3.5. Формализиране на целите на измерването

Благодарение на резултатите от стъпки 1-4 е възможно превеждането на поставените задачи в добре дефинирани и структурирани цели на измерването. При този процес е необходимо да се вземе предвид, че съществуват два вида цели: активни и пасивни.

Активните цели са насочени към контролиране на процеса или промяна на продукти, процеси, ресурси или работна среда. Тези цели са типични за дейностите, свързани с подобряване и оценка.

Пасивните цели от друга страна са предвидени за обучение и осмисляне. Те допринасят за характеризиране на обекти според тяхната продуктивност или количествен модел. Също така осигуряват възможност за предсказване на поведение или резултати.

Формализираните цели могат да се разглеждат като изградени от четири компонента:

1. обекта, който е от интерес (една същност);

2. причина; перспектива;
3. описание на ограниченията;
4. описание на средата.

Обектът може да е продукт, ресурс, агент, артефакт, дейност, метрика, среда или дори може и да е група от обекти. Важното за него е, че е обект на интерес и има нужда да бъде описан с измерените стойности.

Причината за измерването е да може да бъде разбрана, предсказана, планирана, контролирана, сравнена, оценена или подобрена някоя функционалност или качество на обекта. Примери са цената, размера, надеждността, качеството, времето за пускане на пазара, удовлетворението на клиентите.

Перспективата определя кой е заинтересуван от резултатите от измерването, тъй като в зависимост от ролята, която този човек изпълнява в проекта, представите за постигането на целта може съществено да се различават.

Описанието на средата дава контекст, в който да бъдат интерпретирани резултатите от измерванията. При липси в това описание е възможно тълкуването на резултатите да бъде погрешно или изобщо да не бъде извършено. В описанието се включва всичко, което влияе на обекта от интерес, както и всичко, на което той влияе (размери, ресурси).

За конкретния проект, спазвайки описаните по-горе правила, могат да бъдат формулирани следните формализирани цели:

Обект на интерес: Достъп до данните в базата
Причина: Изследване на достъпа до базата данни през интерфейса на приложението с цел подобрение на защитата на данните
Перспектива: Изучаване на слабите звена в софтуера, които могат да доведат до нелигитимен достъп до базата данни от гледна точка на програмистите разработващи софтуера
Среда: MS SQL 2005, Windows 2003 Server

Обект на интерес: Съхраняването на данните в базата
Причина: Изследване на звената от базата данни, които имат нужда от подобрение
Перспектива: Изследване на бързината на заявките използвани за достъп до данните, формата на съхранение на данните, връзките между различните обекти от гледна точка на хората отговарящи за подобрението на структурата на базата данни.
Среда: MS SQL 2005, Windows 2003 Server

Обект на интерес: Реализацията на позициите и ролите от бизнеса

Причина: Анализиране на методите използвани при определяне на функционалността, която един служител притежава с цел подобряване на сигурността на системата
Перспектива: Изучаване на връзките между позициите и ролите от гледна точка на реализиращите системата
Среда: Windows 2003 Server, .NET 2.0

Обект на интерес: Възможностите за натоварване на системата
Причина: Да се оцени надеждността на системата в пикови ситуации, с цел да бъдат избегнати загуби в бизнеса
Перспектива: Изучаване на границите на допустимото натоварване, спрямо нуждите на бизнеса
Среда:IE 6.0, Mozilla, Windows XP

Обект на интерес: Възможностите за обработка на голямо количество данни
Причина: Оценяване на възможностите на системата да обработва голямо количество данни, за да бъде подобрена ефективността
Перспектива: Изследване на компонентите обработващи големи количества данни от гледна точка на програмистите
Среда: MS SQL 2005, Windows 2003 Server, .NET 2.0

Обект на интерес: Възможност за едновременна работа на голям брой хора
Причина: Изучаване на зависимостта между количеството потребители и бързината на системата с цел предвиждане на влиянието, което ще окаже върху системата разширяването на бизнеса
Перспектива:Определяне на стесняванията в системата от гледна точка на разработващите системата
Среда: MS SQL 2005, Windows 2003 Server, .NET 2.0

Обект на интерес: Потребителски интерфейс
Причина: Анализиране на потребителския интерфейс с цел подобрение на ефективността на бизнес процеса
Перспектива: Изследване на най-използваните функционалности, логически свързаните дейности и интуитивността на интерфейса, от гледна точка на хората, които ще работят с него
Среда: IE 6.0, Mozilla

3.6. Определяне на въпросите за количеството и свързаните с тях индикации, които ще бъдат

използвани за постигането на целите на измерването

Тази стъпка използва парадигмата Цел-Въпрос-Метрика (goal-question-metric), която е разработена от Базили и Ромбах. Тя е полезна не само с факта, че определя точните метрики, които са необходими, но и причините. Въпросът "Защо?" е изключително важен, тъй като той дава смисъл на данните и осигурява база за повторното използване на плановете и процедурите за измерване.

При използването на парадигмата ЦВМ, за да бъдат определени нужните метрики, е необходимо първо да се определят въпросите свързани с количествените характеристики на формалните цели, след което да бъдат избрани "индикации". Под индикация се има предвид един или повече резултата от измерване, които имат за цел да подпомогнат изясняването на какво точно е необходимо да се измери.

Когато се определят въпросите и индикациите е важно да не се забравят целите, към които са насочени и как ще бъдат използвани получените резултати. Например за "Да се оцени надеждността на системата в пикови ситуации" могат да бъдат зададени следните въпроси:

1. Кои са източниците на натоварването?
2. Какви са нормалните нива на натоварване?
3. Колко често се подлага системата на натоварване?
4. До какви стойности може да достигне натоварването?

Като индикации могат да се използват статистически графики показващи промяната в натоварването през деня или за даден период от време (годишен, месечен). От тях може да се получи информация за средните нива на натоварване, за периодичността на натоварването и пиковите му стойности. На първия въпрос може да се даде отговор чрез разглеждането на графика, отразяваща разпределение на натоварването спрямо основните дейностите извършвани от бизнеса (административни дейности, финансови операции, продажби)

3.7. Определяне на данните, нужни за съставянето на индикаторите, с които да се отговори на поставените въпроси

За да се съставят индикаторите от предната стъпка е необходимо да се определи кои са елементите от данни, които те трябва да включват. В процеса на определянето лесно се забелязва кои са елементите използвани най-често, т.е. имат голямо приложение.

От описанието на споменатите в стъпка 6 статистики могат да бъдат извадени следните елементи, описани в Таблица 1:

Таблица 1: Определяне на необходимите елементи и принадлежността им към различните индикатори

Необходими елементи	1	2	3	4
брой заявки, пристигащи към сървъра	x	x	x	x
брой хора, работещи едновременно	x	x	x	
изпращане на молби				x
обработка на молби				x
разпределяне на плащанията по кредити				x
административна дейност				x
изготвяне на статистики				x
обработка на плащанията към банки и партньори				x
големина на потока от данни (throughput)	x	x	x	

Където 1 е индикаторът разпределение на натоварването през деня,
2 е индикаторът месечно разпределение,
3 е годишно разпределение,
а 4 е разпределение на натоварването спрямо дейностите.

3.8. Дефиниране на измерванията, които ще се ползват

Дефинирането на измерванията е необходима дейност, за да се осигури, че данните, които са получени, са правилните. Едни и същи данни могат да се тълкуват по различен начин в зависимост от конкретните нужди. Използването на различни начини за измерване на данните може да доведе до различни резултати. Тъй като е необходимо данните да бъдат събирани от различни хора и в различни среди или за различни версии на продукта, трябва да се гарантира, че тези данни са консистентни. Основната цел на тази стъпка е да се осигури, че всички разбират напълно какво точно представляват измерените стойности и че резултатите ще могат да бъдат правилно интерпретирани.

При определянето на структуриран метод за дефиниране на измерванията, трябва да се има предвид, че нещата, които не са от значение за един, за друг може да са много важни, поради което е необходимо дефинициите да са по-обширни. Дефинициите целят изясняването на детайлите, докато структурираният метод определя използването на всички детайли и тяхното записване.

Операционните дефиниции определят как се събират данните. Тяхната цел е да се осигури комуникацията (какво е било измерено, как е било измерено, какво е било включено и какво изключено) и повтаряемостта (следвайки дефинициите всички достигат до същите по смисъл резултати).

Всеки един от индикаторите, определени в предната стъпка, е необходимо да се дефинира. Най-често използвани за тази цел са въпросници или списъци, детайлно отразяващи качества и характеристики

на индикаторите. Таблица 2 отразява списък с атрибутите, участващи при измерването на натоварването през деня.

Таблица 2: Списък на атрибутите, участващи при измерването на натоварването през деня

Дефиниция за измерването на натоварването през деня		
Компоненти на натоварването	*Включва*	*Изключва*
1. Попълване на молби	+	
2. Обработка на молби	+	
3. Административна дейност		+
4. Обработка на плащанията		+
5. Статистика		+
6. Счетоводство		+
Участници в натоварването		
1. Служители с достъп през локалната мрежа	+	
2. Служители с достъп извън локалната мрежа	+	
Изпращани данни		
1. Заявки, засягащи само една таблица	+	
2. Заявки, засягащи много таблици	+	
3. Заявки за селектиране	+	
4. Заявки за вмъкване на запис	+	
5. Заявки за изтриване на запис	+	
6. Заявки за редактиране на запис	+	

Същата таблица важи и за измерването на натоварването през месеца и през годината, разликата е единствено интервалът на измерването - час, ден, месец.

3.9. Определяне на действията, които ще бъдат извършени за реализацията на измерванията

Деветата стъпка има за цел да събере информация за текущото състояние и използването на набелязаните метрики. Основно може да бъде разделена на следните етапи - анализиране, диагностициране и определяне на действията.

Анализирането включва идентифициране на метриките, вече използвани в компанията, определяне на елементите, необходими за измерването, кои от тях се събират и как се събират, кои процеси осигуряват тези данни и как се съхраняват те.

За конкретния проект данните за елементите, свързани с натоварването на системата, могат да бъдат извлечени от статистическите справки, които компанията поддържа, както и от анализите за бъдещото й развитие и развитието на пазара. В момента, според тези справки, средният брой молби, пристигащи за час, е 150, като при промоционални акции и около празници нараства до 200. Най-високото отчетено ниво в историята на фирмата е от

600 молби за час, а според плановете за развитие при евентуално разширение може да се очакват и до 1000 молби в час.

Диагностицирането означава оценяване на данните, които в момента компанията събира, и доколко те отговарят на нуждите на измерването, което ще бъде извършено. Също така и как могат да бъдат използвани тези данни – дали ще бъдат адаптирани към нуждите на измерването или нуждите ще бъдат адаптирани към съществуващите данни. Необходимо е да се установи има ли данни, които са необходими, а липсват.

За целите на извършваното изследване данните, получени от статистическите справки и бизнес плановете, са достатъчни за определянето на измерванията, които ще бъдат направени.

При определянето на действията се идентифицират елементите, върху които ще се изгражда планът за измерване. Избират се методите за събиране на данните и как ще бъдат записани резултатите, какви инструменти ще бъдат използвани, какви ще са времевите интервали за измерването, документиране на процедурите по събиране на данните, както и кой ще използва данните. Необходимо е също така да бъдат идентифицирани детайлността на данните, хората с достъп до данните, както и дефинициите, които осигуряват правилното интерпретиране на данните.

Елементите, на които ще се изгражда плана за натоварването са дейностите, свързани с най-много човешки ресурси, а следователно имащи най-голям ефект върху производителността. За тяхното измерване е определена среда, в която няма външна намеса и по този начин се осигурява консистентността на получените резултати. Избран е инструмент, за измерване и записване на получените данни, който да има максимално приложение в целия процес – от записа на сценариите, през тяхното изпълнение, до анализа на резултатите. Определя се времето, за което протича един сценарий и грануларността на данните. Конкретните стойности са описани в Плана за провеждане на тестването.

3.10. Подготвяне на план за реализация на измерванията

Въпреки че в зависимост от установените от фирмата правила, плановете имат различна структура, няколко са основните точки, които задължително трябва да присъстват.

На едно от първите места в плана трябва да присъства целта или целите, които са поставени – измерванията, които ще бъдат реализирани; с какво те са важни за организацията и какво се очаква да се постигне като резултат.

Описание на целите и обхвата на дейностите, които ще бъдат извършвани, и как те са свързани с други дейности като например подобряването на процеса, оценка на изразходваните усилия и ресурси.

Секцията за имплементация, включва описание на дейностите, необходими за измерванията, които са били определени в целите. Точно описание на усилията, разпределени по дейности, продукти и задачи. Определяне на последствията и зависимостите, които влияят върху разписанието на дейностите или разпределението на ресурсите.

В плана за дейностите е необходимо да се посочат начало и крайни срокове за различните дейности, продукти и задачи, така че да се следи изпълнението им спрямо графика. Също така трябва да се дефинират ресурсите, нужни за работата, и хората, играещи роля при изпълнението на дейности от плана, както и точните им отговорности. В случай, че някой от измерванията се базират на допускания, необходимо е тези допускания да бъдат описани.

Изготвения за целите на тестването план е поместен в Приложение 1.

4. Имплементация на тестовите измервания

В тази глава се описва последователността от действия при изпълнението на тестовите сценарии – особености при записването на скриптовете, конфигурация на тестовите сценарии, провеждане тестовете. Анализирането на получените резултати ще се осъществи с помощта на Mercury LoadRunner Analysis – инструмент, който предоставя възможност за разглеждане на поведението на системата по време на тестовете за натоварване, избор на графики само на определени параметри и комбинации от параметри и изготвяне на справки на базата на дефинирани от потребителя филтри.

4.1. Инструмент за реализация на плана за тестване на натоварването

За реализация на предвидените дейности в плана за тестване на натоварването е избран инструментът, разработен от фирмата Mercury специално за изследване на натоварването – Mercury LoadRunner 8.0. Той включва в себе си 5 компонента, които дават възможност за записване на скриптове, за изпълнение и конфигурация на сценарии, както и за анализиране на резултатите получени при изпълнението [26].

За записването на скриптовете се използва компонентът Virtual User Generator, който предоставя възможност за следене на реалното време за мислене, за добавяне на транзакции, с които да се отделят действията извършвани в скрипта, параметризиране на данните, а също и създаване на потребителски функции чрез използване на езика за програмиране C.

Конфигурацията и изпълнението на сценарии, състоящи се от записаните скриптове, се намират в отделен компонент – LoadRunner Controller. Чрез него могат да бъдат зададени процентни отношения между изпълняваните скриптове или ръчно да бъдат конфигурирани стойностите. Може да се зададе схема на изпълнение – едновременно или постепенно стартиране на виртуалните потребители, продължителност на тяхната дейност и поведение в края на сценария.

За да се симулира действително натоварване се използва Load Generator, който стартира виртуалните потребители. LoadRunner предоставя възможност за повече от един Load Generator, така че натоварването към сървъра да идва от повече от една машина.

За анализа се използва друг компонент LoadRunner Analysis. Той дава възможност за разглеждане на резултатите като се добавят графики за параметрите, имащи пряко влияние при оценяването на системата. Допълнително тези графики могат да бъдат филтрирани, като се зададе времеви интервал или име на транзакция, име на скрипта или идентификатор на виртуалния потребител. Има възможност и за комбиниране на графиките,

за да се види как два или повече параметъра се изменят в различните моменти от сценария или дори да се представи един параметър като функция на друг.

Компонентът, който осигурява достъп от едно място до всички описани вече компоненти, се нарича LoadRunner Launcher.[13]

4.2. Конфигурация на скриптовете

На базата на извършеното планиране на тестовете се достига до изготвянето на 5 скрипта, с които се покрива функционалността, описана в плана за измерване на натоварването.

Първият скрипт (Приложение 4) описва бизнес процеса “Одобрение на молба”, използвайки следните стъпки:

1. Потребител с права на “кредитен инспектор”(позиция в бизнес организацията, имаща право да одобрява молби) влиза в системата.
2. От главното меню той избира списъка на молбите готови за обработка.
3. Филтрира този списък по регион и време на постъпване на молбата.
4. Избира първата молба от списъка
5. Проверява адреса, посочен от клиента (използва се функция, която търси в базата на ЕКАТТЕ)
6. Проверява клиента за здравно осигуряване на базата на записаното ЕГН (използва се функция, изпращаща заявка към сайта на НОИ)
7. Проверява се за валидност номера на личната карта (заявка към сайта на МВР)
8. Проверява се за валидност посочения домашен телефон в база, предоставена от БТК
9. Записват се резултати от проведени разговори на посочените от клиента телефони
10. Проверява се за “съмнителни данни”- справка, която показва на кредитния инспектор дали не е нарушено някое от условията, разработени от компанията за оценка на нейните клиенти
11. Молбата се одобрява

Тъй като две са основните действия в скрипта – вход в системата и след това разглеждане и одобряване на молба, е необходимо те да бъдат отделени в две различни транзакции – loginApproval и fillApproval.

Вторият скрипт (Приложение 5) реализира бизнес процеса “Въвеждане на молба за продукт1, получена по факс”. При него се използва стандартната последователност от стъпки, които един служител изпълнява:

1. Потребител с права на “факс оператор” (позиция в бизнес организацията, имаща право да въвежда молби, пристигнали по факс) влиза в системата
2. От главното меню той избира попълване на молба за продукт1
3. Попълва последователно – код на молбата, търговски партньор и код на търговски обект
4. Попълва стъпка1 от молбата – избор на продукт
5. Попълва стъпка2 – избор на схема за плащане
6. Попълва стъпка3 – лични данни на клиента
7. Попълва стъпка4 – настоящ адрес и контакти
8. Попълва стъпка5 – образование и семейно положение
9. Попълва стъпка6 – месторабота
10. Попълва стъпка7 – доходи и разходи
11. Стъпка8 – отпечатване на необходимите документи
12. Стъпка9 – разглеждане на попълнените данни
13. Стъпка10 – записване на коментари и изпращане на молбата

За този скрипт отново има нужда от отделни транзакции за стъпка 1 и за стъпки от 3 до 13. Тези транзакции са записани съответно като – loginOnlineProd1 и fillOnlineProd1.

В Приложение 6 е посочен скриптът, който въвежда молби за продукт2, получени по факс. Той също спазва стандартната процедура, през която минават служителите на организацията:

1. Потребител с права на “факс оператор” (позиция в бизнес организацията, имаща право да въвежда молби, пристигнали по факс) влиза в системата
2. От главното меню, той избира попълване на молба за продукт2
3. Попълва последователно – код на молбата, регион и код на локалния офис
4. Попълва стъпка1 – избор на схема за плащане
5. Попълва стъпка2 – лични данни на клиента
6. Попълва стъпка3 – настоящ адрес и контакти
7. Попълва стъпка4 – образование и семейно положение
8. Попълва стъпка5 – месторабота
9. Попълва стъпка6 – доходи и разходи
10. Стъпка7 – отпечатване на необходимите документи
11. Стъпка8 – разглеждане на попълнените данни
12. Стъпка9 – записване на коментари и изпращане на молбата

За да се измери времето за вход в системата и въвеждане на молбата, се използват следните две транзакции – loginFaxProd2 и fillFaxProd2.

За попълването онлайн на молба за продукт1 се използва скриптът от Приложение 7. Той реализира следните стъпки:

1. Потребител с права на “ПОС оператор” (позиция в бизнес организацията, имаща право да въвежда онлайн молби) влиза в системата
2. От главното меню, той избира попълване на молба за продукт1
3. Попълва стъпка1 от молбата – избор на продукт
4. Попълва стъпка2 – избор на схема за плащане
5. Попълва стъпка3 – лични данни на клиента
6. Попълва стъпка4 – настоящ адрес и контакти
7. Попълва стъпка5 – образование и семейно положение
8. Попълва стъпка6 – месторабота
9. Попълва стъпка7 – доходи и разходи
10. Стъпка8 – отпечатване на необходимите документи
11. Стъпка9 – разглеждане на попълнените данни
12. Стъпка10 – записване на коментари и изпращане на молбата

При него също има две обособени транзакции – loginOnlineProd1 за отчитане на времето необходимо за вход в системата и fillOnlineProd1 за отчитане на времето необходимо за попълването на онлайн молба за продукт 1.

Последният скрипт (Приложение 8) осъществява попълването онлайн на молба за продукт2. При него последователността от действия е подобна на предходните молби:

1. Потребител с права на “ПОС оператор” (позиция в бизнес организацията, имаща право да въвежда онлайн молби) влиза в системата
2. От главното меню, той избира попълване на молба за продукт2
3. Попълва стъпка1 – избор на сума и схема за плащане
4. Попълва стъпка2 – лични данни на клиента
5. Попълва стъпка3 – настоящ адрес и контакти
6. Попълва стъпка4 – образование и семейно положение
7. Попълва стъпка5 – месторабота
8. Попълва стъпка6 – доходи и разходи
9. Попълва стъпка7 – сметка към която да се преведе сумата от стъпка1
10. Стъпка8 – отпечатване на необходимите документи
11. Стъпка9 – разглеждане на попълнените данни
12. Стъпка10 – записване на коментари и изпращане на молбата

Отново има две транзакции - loginOnlineProd2 за отчитане на времето необходимо за вход в системата и fillOnlineProd2 за отчитане на времето необходимо за попълването на онлайн молба за продукт 2.

При записването на всеки един от необходимите скриптове се следи и за времето, което служителят изразходва за мислене и попълване. Средната стойност, която е установена в практиката, е около 6 минути за всеки един от

горните сценарии. За да се отразят колебанията, дължащи се на човешкия фактор, при изпълнение на скриптовете се допуска времето да варира между 80% и 120% от записаното време за мислене.

4.3. Конфигурации на тестовия сценарий за натоварване

След записването на скриптовете, покриващи тестваната функционалност, е необходимо те да бъдат събрани в един сценарий, който максимално да се доближава до реалния бизнес процес. За тази цел се изследва процентното отношение на отделните бизнес процеси, които са отразени в скриптовете. Спрямо статистически извадки на получените молби от четирите типа за период от един месец, може да се пресметне, че онлайн молби за продукт1: онлайн молби за продукт2: факс молби за продукт1:факс молби за продукт2 = 60:25:10:5. От бизнес процеса “Одобрение на молба” се изисква максимално бързо да отговори на пристигащите молби, затова при изпълнението на тестовия сценарий ще бъде прието, че броят на обработващите е равен на броя на попълващите молби.

За да се добие по-добра представа за възможностите за производителност на системата е необходимо да бъдат избрани различни нива на натоварване. Тези нива могат да бъдат определени като за най-ниското се вземе средната стойност на натоварване, отчетена през празничен или промоционален период (200 молби/час). За следваща стойност може да се избере максималното натоварване отчитано до момента в работата на компанията (600 молби/час). Тъй като компанията предвижда разширяване на бизнеса ще бъдат избрани натоварвания по-големи от отчетеното до момента (1000 молби/час и 1400 молби/час).

При конфигурирането на сценария е необходимо да се зададе броят потребители, използващи едновременно системата. Като се има предвид, че за бизнеса натоварването се измерва в брой молби пристигнали за 1 астрономически час, а един служител попълва една молба средно за 6 минути, то за 1 час служителят ще попълни 10 молби. Нужният брой потребители може да се получи като се раздели броят на молбите за 1 час на 10.

Най-ниската избрана стойност за натоварване е 200 молби в час, следователно за този сценарий са необходими 20 виртуални потребителя, попълващи молби в съотношение 12:5:2:1 и още 20 виртуални потребителя за одобряване на пристигащите молби.

За втория сценарий е определено натоварване от 600 молби в час. Отново броят на попълващите молби е разпределен в същото процентно съотношение – 36:15:6:3, а броят на одобряващите е равен на общия брой попълващи – 60 виртуални потребителя.

При останалите стойности от 1000 и 1400 молби в час, съотношението на попълващите е съответно 60:25:10:5 и 84:35:14:7, а виртуалните потребители, одобряващи молби са 100 и 140.

И за четрите сценария е избрана еднаква продължителност от 60 минути, през които се очаква всеки виртуален потребител да изпълни по десет пъти скрипта си.

4.4. Основни параметри в измерванията

Оценката на производителността на системата се базира на стойностите на различни параметри, всеки от които отразява отделен аспект от работата на системата.

Броят заявки в секунда (hits per second) показва честотата, с която пристигат заявките към сървъра. До голяма степен обемът данни изпратени към сървъра зависи от този брой.

Обемът данни (throughput) изразява количеството информация, което се изпраща към сървъра. Той е ограничен по стойност, тъй като информацията пристига по канали с определен капацитет.

Времето за отговор (response time) е един от показателите, които пряко влияят на усещането на потребителите за системата. На базата на времето, което потребителите чакат, за да бъдат обслужени заявките им, те определят представянето на системата като бързо, достатъчно бързо, бавно или твърде бавно. Поради тази причина стойностите на този параметър са изключително важни за крайната оценка на производителността.

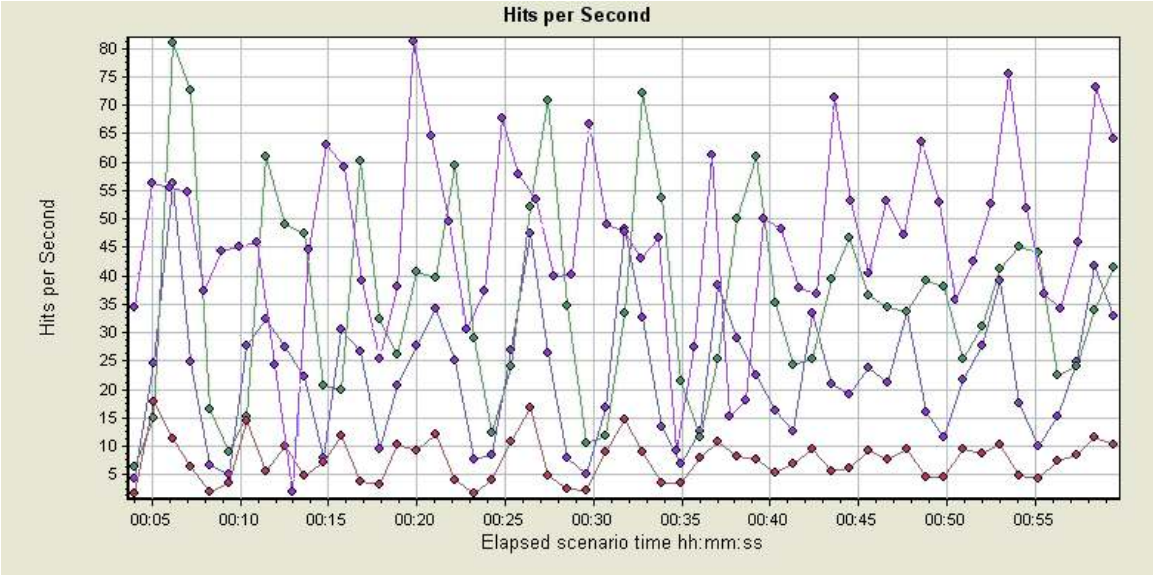
Друг параметър, който не винаги присъства в справките, е броя грешки регистрирани по време на теста. Те се дължат предимно на твърде голямо време за отговор или дори отказ на системата да отговори на подадената й заявка.

За да се анализира производителността на системата и да могат да се направят предвиждания за поведението й при увеличаване на натоварването е необходимо да се разгледат стойностите на изброените параметри като функции на броя потребители.

4.5. Анализ на получените резултати

За всеки един от изпълнените сценарии е необходимо да се сравнят графиките отговарящи на един и същ параметър. Отклоненията в техните показания носят информация за влиянието на броя потребители върху стойността на съответния параметър. Целта на анализа е освен да установи каква е производителността на разработваната система, също така и да намери точките, в които системата има най-лошо представяне – така наречените “стеснявания” (bottlenecks), които ограничават възможността на системата да поеме по-голямо натоварване [8].

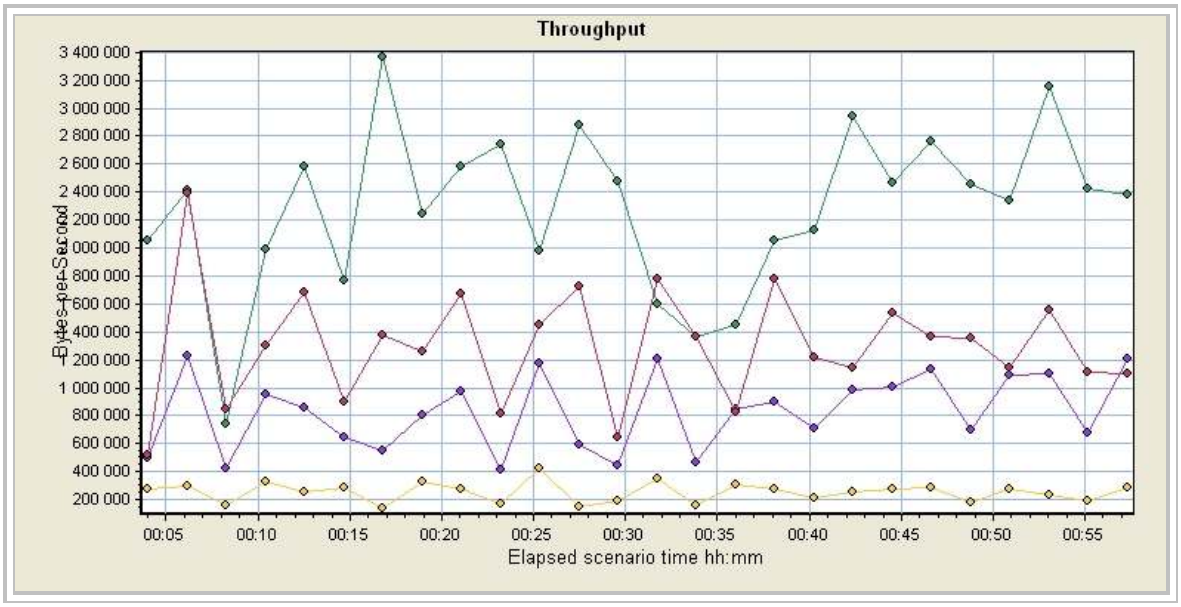
Параметърът „*брой заявки в секунда*” е пряко зависим от броя потребители, които ползват системата в даден момент. Интуитивен е фактът, че с нарастване на броя потребители, пропорционално ще расте и средния брой заявки в секунда. От направените измервания (Фиг. 6) също може да се види, че стойностите се променят равномерно.



Брой виртуални потребители	Среден брой заявки в секунда
40	7,639
120	22,738
200	36,387
280	46,158

Фигура 6: Средно ниво на броя заявки изпращани в секунда, в зависимост от броя виртуални потребители

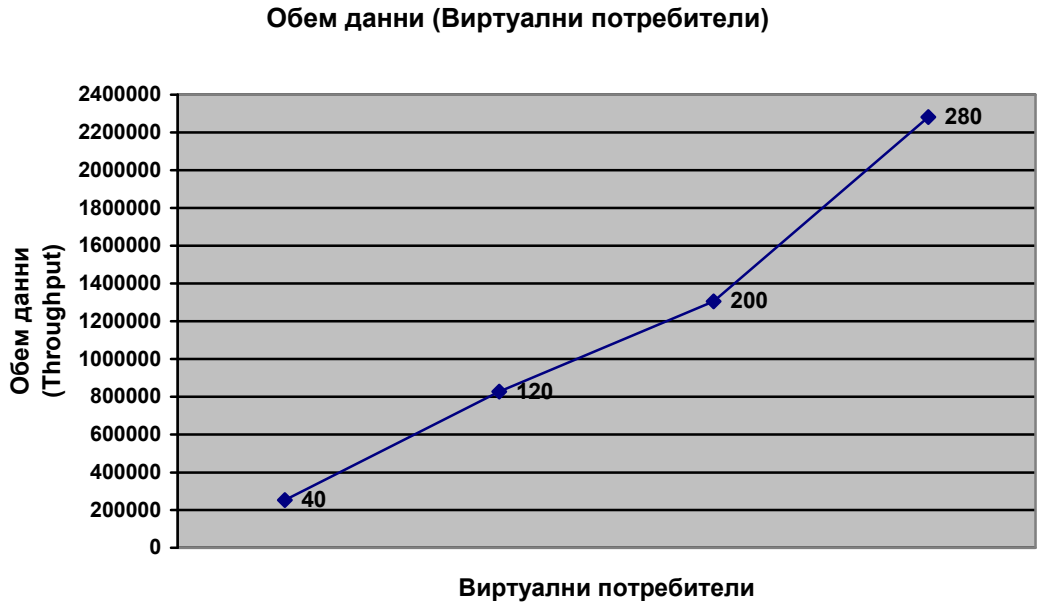
Резултатите за обемът данни, изпращани към сървъра, в зависимост от броя виртуални потребителя могат да се видят на Фигура 7:



Брой виртуални потребители	Минимум	Средна стойност	Максимум
40	138579.739	252058.37	428890.956
120	410471.173	826837.754	1228941.17
200	522924.783	1305336.478	2394101.398
280	737126.43	2281702.765	3365576.344

Фигура 7: Средно ниво на обемът данни измерен в байтове, спрямо броя виртуални потребители

Ако се вземе средната стойност от всеки сценарий, обемът данни може да бъде представен като функция на едновременно работещите виртуални потребители, което е показано на Фигура 8:



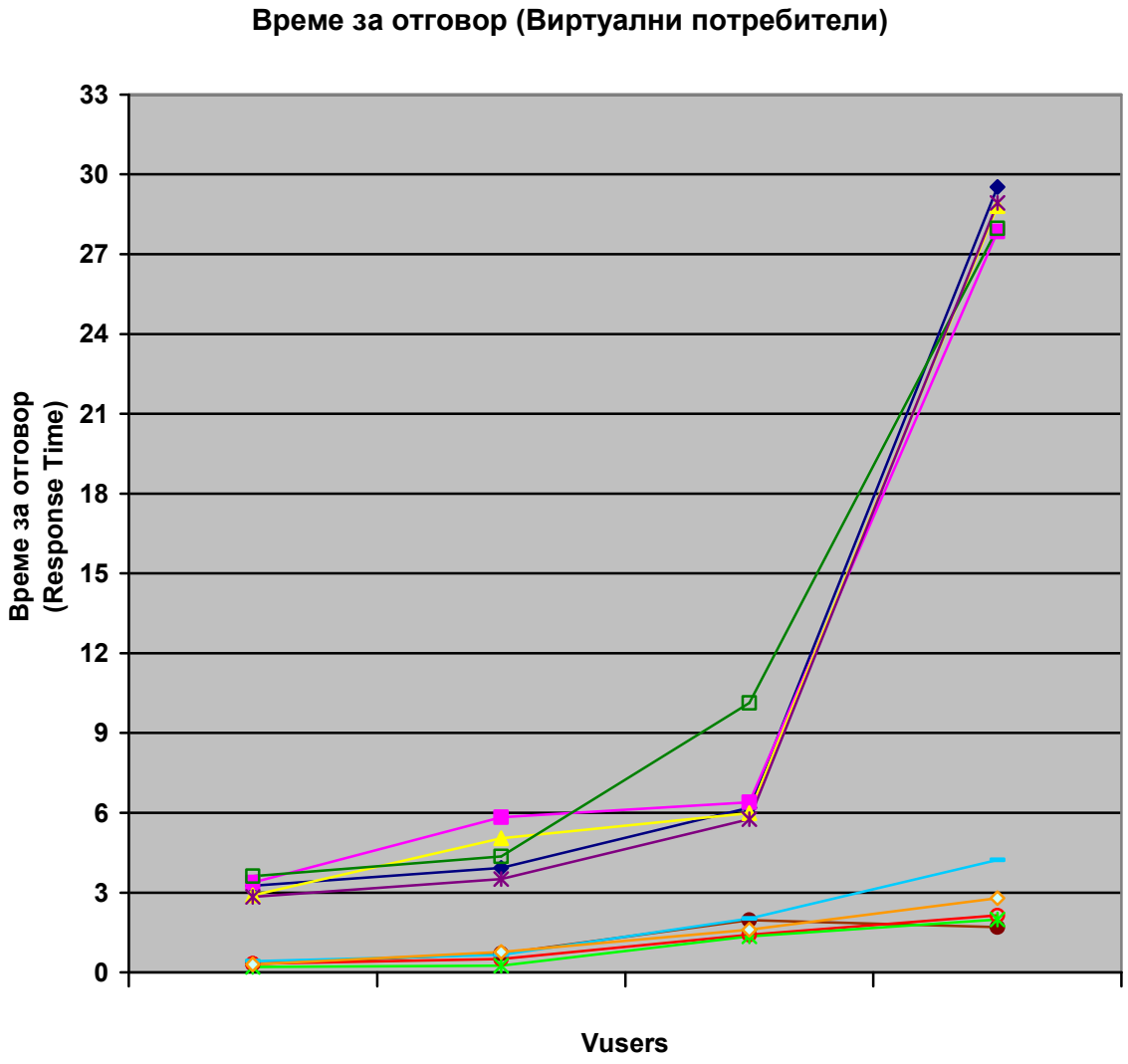
Фигура 8: Обем данни, измерен в байтове, като функция на броя едновременни потребители

Както се вижда от Фигура 8 с нарастването на броя потребители нараства и обемът данни, изпращани към сървъра. Изводът, който може да се направи на базата на тези данни е, че при нива на натоварване двукратно по-високи от максималните постигани до момента в реални условия, системата все още не е достигнала максимума на възможностите си.

Времето, което е необходимо на системата да отговори на дадена заявка също е зависимо от броя на виртуалните потребители. Отделните скриптове включват времето за мислене на истинския потребител. За да се види колко всъщност време отнема отговорът на системата е необходимо да бъдат разгледани резултати, от които се изключва това "допълнително"време.

Таблица 3: Средно време за извършването на всяка транзакция спрямо броя на виртуалните потребители

Транзакция / Брой потребители	40	120	200	280
Approval	3.26	3.931	6.192	29.525
fillFaxProd1	3.382	5.833	6.4	27.854
fillFaxProd2	2.908	5.039	5.985	28.804
fillOnlineProd1	3.619	4.358	10.139	27.972
fillOnlineProd2	2.849	3.512	5.764	28.926
loginApproval	0.306	0.706	1.963	1.696
loginFaxProd1	0.323	0.504	1.412	2.138
loginFaxProd2	0.198	0.256	1.347	1.982
loginOnlineProd1	0.419	0.66	2.018	4.237
loginOnlineProd2	0.297	0.767	1.605	2.794

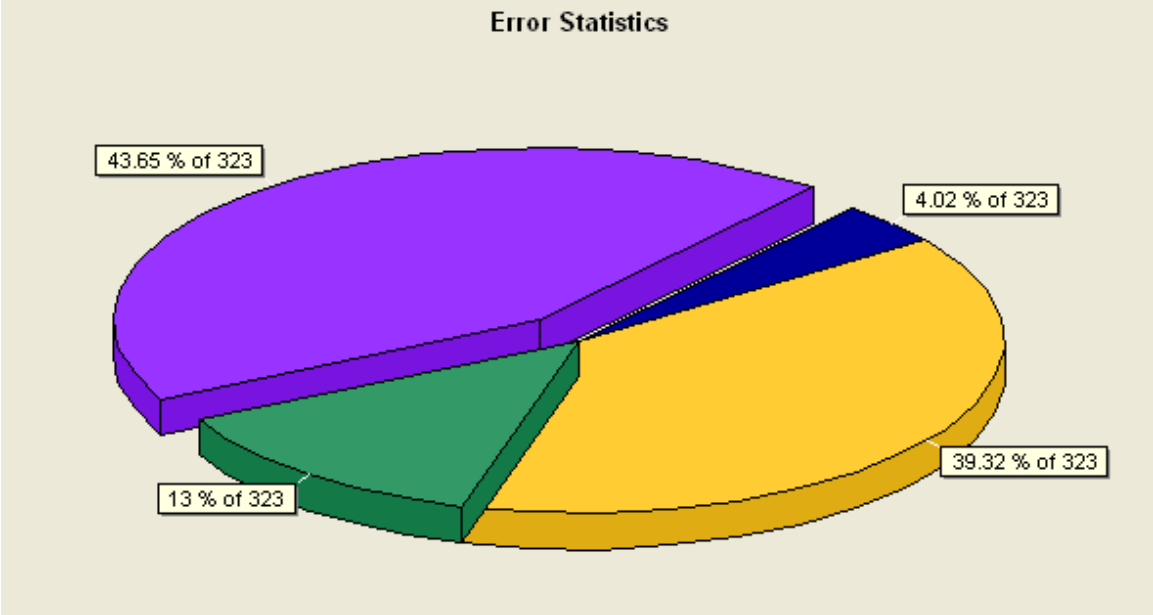


Фигура 9: Време за отговор, представено като функция на броя едновременни потребители

Противно на зависимостта между брой потребители и обем данни, при времето за отговор с увеличаването на количеството едновременно работещи със системата хора, стойностите нарастват все по-бързо. Както се вижда от Фигура 9, нивото на рязко покачване на времето за отговор не е достигнато при тези измервания.

Друг важен параметър за оценяването на производителността е броя грешки, дължащи се на невъзможността на системата да отговори в срок на исканията на потребителя. При направените измервания грешки се появяват едва при 280 виртуални потребителя, грешките са 323 към 8 839 успешни. Както се вижда от фигура 10 - най-голям процент от тях (43,65%) се дължат на грешка 27727 (изтичане на времето за сваляне на реурси), 39,32% се дължат на грешка 27791 (прекратяване на връзката към сървъра), 13% са от

грешка 27728 (изтичане на времето за сваляне на не-ресурси) и 4,02% са причинени от 27979 (търсената форма не може да бъде намерена).



Фигура 10: Статистика на грешките при 280 едновременно работещи потребителя

Необходимо е да се проверят максимумите, които са били измерени при този сценарий, и да се отбележат като цели в следващия етап на тестване на натоварването на системата за да може да се установи точката, от която започват проблемите.

4.6. Препоръки към системата и насоки в тестването ѝ на базата на анализираните резултати

При анализа на резултатите бе установено, че в рамките на направените измервания не е достигнат максимума на нито един от изследваните параметри. Въпреки, че стойностите използвани при измерванията напълно покриват нуждите на бизнеса, за да се постигне пълната картина на производителността е необходимо да се намери прага на възможностите на системата. Поради тази причина е трябва да се планират допълнителни сценарии, ориентирани към постигането на оптимално натоварване.

Друга насока, която да се изследва след като бъде установено, че системата покрива изискванията спрямо натоварването, е как се отразява на производителността промяната в конфигурацията на системната среда.

Една от следващите цели е да се предвидят възможностите на системата да поеме натоварване при бъдещо разширяване на бизнеса. Това може да се постигне чрез добавянето на нови хардуерни ресурси – процесори, памет дисково пространство, допълнителни машини.

Заключение

Тестването на софтуера е неизменна част от неговия жизнен цикъл. За успеха на крайния продукт осигуряването на качеството му е не по-малко важно от функционалността, която той предлага. Именно поради този факт изключително голямо внимание трябва да се обърне на методиката, която се използва в процеса на тестване. Независимо от конкретната реализация на софтуера, тестването трябва да премине през следните фази – изучаване на бизнес изискванията, определяне на целите, които продукта трябва да постигне, изготвяне на план, с който да се провери дали целите са постигнати, анализиране на получените резултати. Тази последователност е спазена от дипломната работа, като по-голямо внимание е обърнато на определянето на целите и анализа на резултатите.

Изборът на подходяща техника за планиране на тестовете осигурява пълнота и сигурност в постигнатите резултати. Поради тази причина за целите на дипломната работа се използва целево-ориентирана техника, която превръща бизнес целите в конкретни цели на тестването. Десетте стъпки описани в Глава 3 осигуряват основните точки, необходими за създаването на добре структуриран и последователен план за тестване на натоварването на системата.

Реализацията на плана за тестване на натоварването в Глава 4 и анализът на получените от него данни изграждат представа за производителността на системата в изследвания диапазон. При провеждането на експериментите и обработката на техните резултати се използва само един инструмент – продукта на фирмата Mercury – Mercury LoadRunner 8.0. Усилията и времето необходими за опитите са значително по-малки благодарение на факта, че този инструмент покрива всички нужди на процеса – от записването на скриптовете, през конфигурацията на отделните експерименти, до детайлното разглеждане на получените резултати.

Прилагането на методика за систематизиране на тестовия процес дава целева насоченост, пълнота и организираност, които гарантират постигането на резултати, отговарящи на действителните нужди на бизнеса.

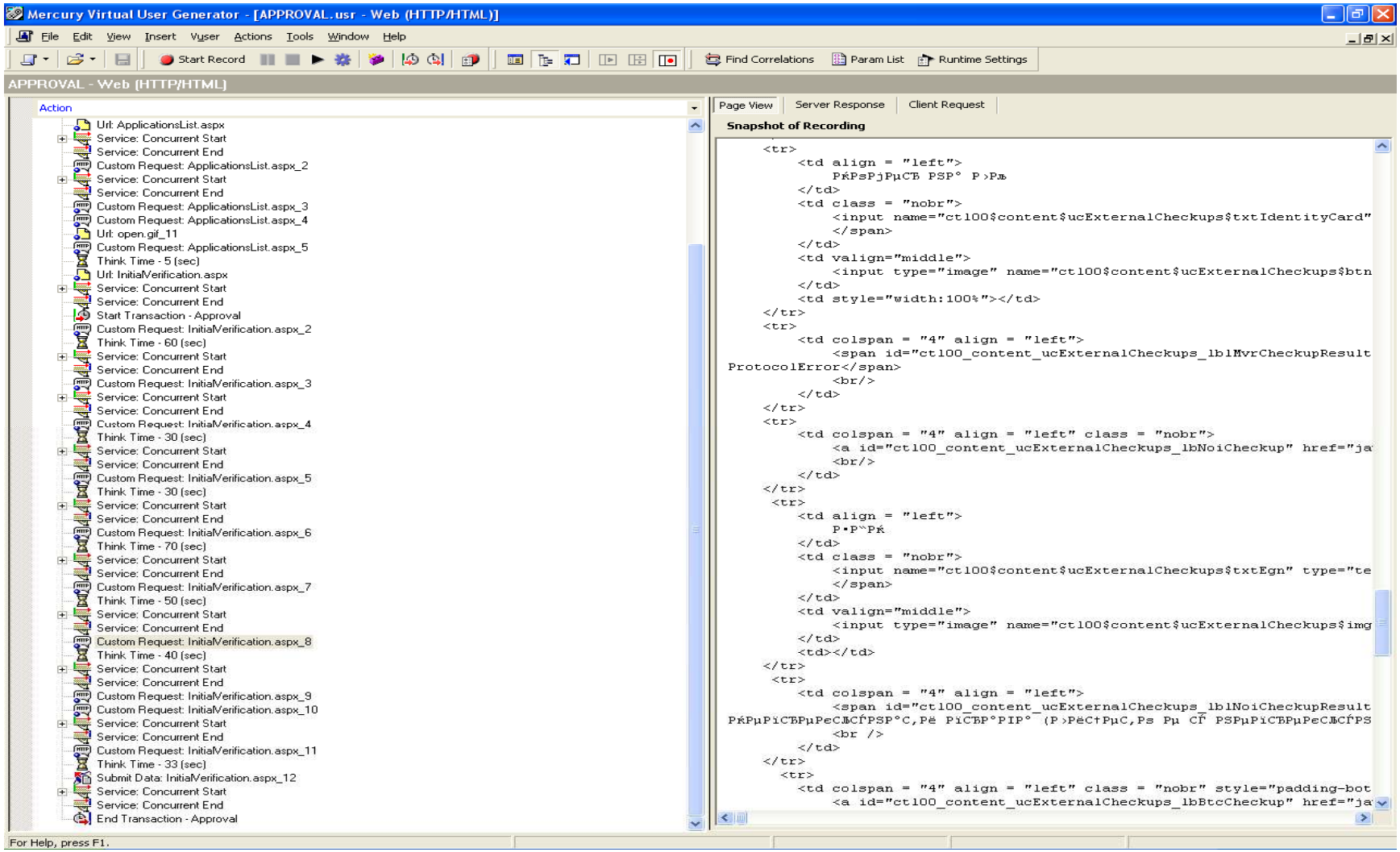
Приложения

Приложение 1: План за провеждане на тестването за натоварване

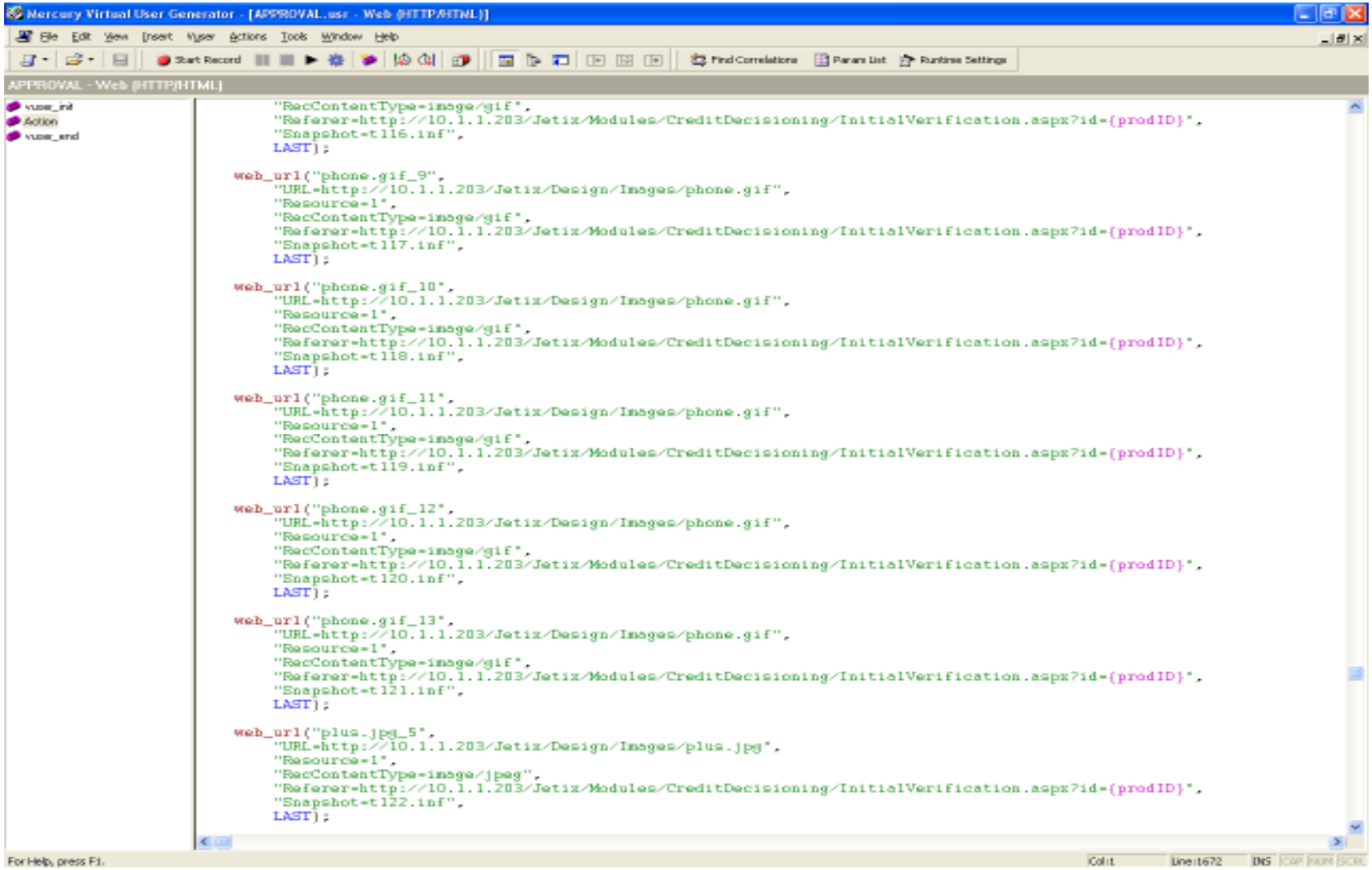
Използвани документи
• Спецификация на бизнес изискванията • Техническа спецификация • Статистика на продажбите за изминали периоди • Спецификация на нефункционалните изисквания
Цели
• Да се изследва бързината на отговора на системата в зависимост от количеството данни • Да се разгледа многократно увеличаване на едновременно работещи потребители • Да се определи кои метрики са важни за анализа • Да се определят значимите отношения между отделните метрики
Обхват
Изпращане и обработка на молби за кредит
Подход
Натоварване на системата със средния брой на очаквани молби (200 за час) Натоварване на системата с максималния постигнат брой молби (600 за час) Натоварване на системата с брой молби над максималния постиган някога (1000 за час) Натоварване на системата с брой молби над максималния постиган някога (1400 за час)
Тестове, които ще бъдат използвани и схеми за тяхното използване
• Тест1: Изпращане онлайн на молба за продукт1 • Тест2: Изпращане онлайн на молба за продукт2 • Тест3: Изпращане по локалната мрежа на получена по факса молба за продукт1 • Тест4: Изпращане по локалната мрежа на получена по факса молба за продукт2 • Тест5: Разглеждане на получена молба, извършване на проверки и одобряване на молбата • Всички скриптове стартират едновременно с пълния си брой виртуални потребители. Изпълнението на тестовете продължава 60 минути.
Имплементация на тестването на натоварването
• Записване на скриптовете • Параметризиране на скриптовете и времето за мислене на потребителите • Изпълнение на предвидените сценарии • Анализ на получените резултати • Описване на наблюдаваните проблеми • Справка за извършената дейност

Използвани инструменти
• Mercury Loadrunner 8.0
Описание на натоварването
• 40; 120; 200; 280 виртуални потребителя • 60 мин при максимален брой на потребителите; • Виртуалният потребител попълва данните за молба, след което прекратява сесията • Виртуалният потребител одобряващ молба, извършва всички проверки предоставени от интерфейса, след което редактира данните от молбата и я одобрява • Времето за мислене на потребителите е между 80% и 120% от записаното време • Всеки от бизнес сценариите (попълване на молба, одобрение на молба) е конфигуриран да трае между 5 и 6 минути • Спрямо статистически данни онлайн продукт1:онлайн продукт2: факс продукт1: факс продукт2 = 60:22:12:6. При изпълнение на тестовете за одобрение на молби се използва същия брой потребители, колкото е общия брой на попълващите молби.
Тестова среда
• Мрежови характеристики: LAN(T1) • Два сървъра, на които е инсталирано приложението и между които се поделя натоварването • База данни с над 1 000 000 записа • Браузър – IE6.0 или Mozilla • Тестовата среда се намира в локална мрежа и достъпът до нея не е позволен отвън
Изключения
• Няма да се тества за натоварване административната част • Няма да се тества системата при разпределяне на натоварването между повече от два сървъра • Не се включват функционални тестове
Продукти от тестването
• Тест план • Тест сценарии • Тест скриптове • Резултати от тестването • Справка за анализа на събраните данни • Справка за откритите проблеми

Приложение 2: Дървовиден изглед на скрипт в Mercury Loadrunner 8.0



Приложение 3: Изглед на скрипт в режим на редакция в Mercury LoadRunner 8.0



Приложение 4: Скрипт за изследване производителността на бизнес процеса „Одобрение на молби”, реализиран в средата на Mercury LoadRunner 8.0

APPROVAL.usr

```
--------------------------------------------------------------------------------
#include "web_api.h"


Action()
{
web_url( "Login.aspx",
"URL=http://10.1.1.203/Jetix/Login.aspx",
"Resource=0",
"RecContentType=text/html",
"Referer=",
"Snapshot=t31.inf",
"Mode=HTML",
LAST);
lr_think_time(5);

lr_start_transaction("loginApproval");

lr_think_time(2);

web_submit_form("Login.aspx_2",
"Snapshot=t32.inf",
ITEMDATA,
"Name=lgnJetix$UserName", "Value={user}", ENDITEM,
"Name=lgnJetix$Password", "Value={pass}", ENDITEM,
"Name=lgnJetix$LoginButton", "Value=Log In", ENDITEM,
LAST);

lr_end_transaction("loginApproval", LR_AUTO);
lr_think_time(3);

web_url("ApplicationsList.aspx",
"URL=http://10.1.1.203/Jetix/Modules/CreditDecisioning/ApplicationsList.aspx",
"Resource=0",
"RecContentType=text/html",
"Referer=http://10.1.1.203/Jetix/Default.aspx",
"Snapshot=t38.inf",
"Mode=HTML",
LAST);

web_custom_request("ApplicationsList.aspx_2",
"URL=http://10.1.1.203/Jetix/Modules/CreditDecisioning/ApplicationsList.aspx",
"Method=POST",
"Resource=0",
"RecContentType=text/plain",
"Referer=http://10.1.1.203/Jetix/Modules/CreditDecisioning/ApplicationsList.aspx",
"Snapshot=t44.inf",
"Mode=HTML",
"Body=ctl00$ScriptManager1=ctl00$content$upSettings|ctl00$content$ddlProduct&__EVENTTARGET=ctl00%24content%24ddlProduc”
“t&__ &ctl00$content$ddlProduct=0&",
LAST);

web_custom_request("ApplicationsList.aspx_3",
"URL=http://10.1.1.203/Jetix/Modules/CreditDecisioning/ApplicationsList.aspx",
"Method=POST",
"Resource=1",
"RecContentType=text/plain",
"Referer=http://10.1.1.203/Jetix/Modules/CreditDecisioning/ApplicationsList.aspx",
"Snapshot=t55.inf",
"Body=ctl00$ScriptManager1=ctl00$content$upSettings|ctl00$content$ddlMerchant&__&ctl00$content$ddlProduct=0&ctl00$content$dd”
“lMerchant=1365&ctl00$content$ddlMerchantPos=-1&ctl00$content$txtFrom=&ctl00$content$txtTo=&",
```

```
LAST);

web_custom_request("ApplicationsList.aspx_4",
"URL=http://10.1.1.203/Jetix/Modules/CreditDecisioning/ApplicationsList.aspx",
"Method=POST",
"Resource=1",
"RecContentType=text/plain",
"Referer=http://10.1.1.203/Jetix/Modules/CreditDecisioning/ApplicationsList.aspx",
"Snapshot=t56.inf",
"Body=ctl00$ScriptManager1=ctl00$content$upSettings|ctl00$content$lbtnSearch&__&ctl00$content$ddlProduct=0&ctl00$content$ddl"
"Merchant=1365&ctl00$content$ddlMerchantPos=1366&ctl00$content$txtFrom=&ctl00$content$txtTo=60&",
LAST);

web_url("open.gif_11",
"URL=http://10.1.1.203/Jetix/Design/Images/open.gif",
"Resource=1",
"RecContentType=image/gif",
"Referer=http://10.1.1.203/Jetix/Modules/CreditDecisioning/ApplicationsList.aspx",
"Snapshot=t57.inf",
LAST);

web_custom_request("ApplicationsList.aspx_5",
"URL=http://10.1.1.203/Jetix/Modules/CreditDecisioning/ApplicationsList.aspx",
"Method=POST",
"Resource=1",
"RecContentType=text/plain",
"Referer=http://10.1.1.203/Jetix/Modules/CreditDecisioning/ApplicationsList.aspx",
"Snapshot=t58.inf",
"Body=ctl00$ScriptManager1=ctl00$content$upCredits|ctl00$content$gvCreditsCred$ctl02$ibtnOpen&__ctl00$content$ddlProduct=0&"
"ctl00$content$ddlMerchant=1365&ctl00$content$ddlMerchantPos=1366&ctl00$content$txtFrom=&ctl00$content$txtTo=60&ctl00$cont"
"ent$gvCreditsCred$ctl02$ibtnOpen.x=9&ctl00$content$gvCreditsCred$ctl02$ibtnOpen.y=15",
LAST);

lr_think_time(5);

web_url("InitialVerification.aspx",
"URL=http://10.1.1.203/Jetix/Modules/CreditDecisioning/InitialVerification.aspx?id={prodID}",
"Resource=0",
"RecContentType=text/html",
"Referer=",
"Snapshot=t59.inf",
"Mode=HTML",
LAST);

lr_start_transaction("Approval");
//function sending request for verification of the client's address

lr_think_time(60);
//function sending request to MON to check client's health insurance on the basis of EGN
//function checking the validity of the client's ID card number
lr_think_time(30);
// function sending request to check for incompatibilities in the client details
lr_think_time(30);
//function sending request to check for the existence of client's address in the EKATE database
lr_think_time(70);
//function checking the home phone number of the client
lr_think_time(50);
//function recording the results of the phone calls made with the client
lr_think_time(40);
//function changing the chosen payment scheme of the application
lr_think_time(33);
//function sending all tha data included in the aplication and approving it
lr_end_transaction("Approval", LR_AUTO);

return 0;
}
```

Приложение 5: Скрипт за изследване производителността на бизнес процеса „Въвеждане на молби, получени по факс за продукт1”, реализиран в средата на Mercury LoadRunner 8.0

FAXPROD1.usr

```
------------------------------------------------
#include "web_api.h"

Action()
{
web_url( "Login.aspx",
"URL=http://10.1.1.203/Jetix/Login.aspx",
"Resource=0",
"RecContentType=text/html",
"Referer=",
"Snapshot=t31.inf",
"Mode=HTML",
LAST);
lr_think_time(5);

lr_start_transaction("loginFaxProd1");

web_submit_form("Login.aspx_2",
"Snapshot=t2.inf",
ITEMDATA,
"Name=lgnJetix$UserName", "Value={user}", ENDITEM,
"Name=lgnJetix$Password", "Value={pass}", ENDITEM,
"Name=lgnJetix$LoginButton", "Value=Log In", ENDITEM,
LAST);

lr_end_transaction("loginFaxProd1", LR_AUTO);

lr_think_time(7);

web_url("ApplicationProcessing.aspx",
"URL=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationProcessing.aspx",
"Resource=0",
"RecContentType=text/html",
"Referer=http://10.1.1.203/Jetix/Default.aspx",
"Snapshot=t8.inf",
"Mode=HTML",
LAST);

lr_think_time(4);

web_submit_data("ApplicationProcessing.aspx_2",
"Action=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationProcessing.aspx",
"Method=POST",
"RecContentType=text/html",
"Referer=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationProcessing.aspx",
"Snapshot=t14.inf",
"Mode=HTML",
ITEMDATA,
"Name=__LASTFOCUS", "Value=", ENDITEM,
"Name=__EVENTTARGET", "Value=ctl00$content$ddlProduct", ENDITEM,
"Name=__EVENTARGUMENT", "Value=", ENDITEM,
"Name=ctl00$content$ddlProduct", "Value=0", ENDITEM,
LAST);

lr_think_time(6);

web_submit_data("ApplicationProcessing.aspx_4",
"Action=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationProcessing.aspx",
```

```
"Method=POST",
"RecContentType=text/html",
"Referer=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationProcessing.aspx",
"Snapshot=t28.inf",
"Mode=HTML",
ITEMDATA,
"Name=__LASTFOCUS", "Value=", ENDITEM,
"Name=__EVENTTARGET", "Value=ctl00$content$btnGo", ENDITEM,
"Name=__EVENTARGUMENT", "Value=", ENDITEM,
"Name=ctl00$content$ddlProduct", "Value=0", ENDITEM,
"Name=ctl00$content$txtCreditCredCode", "Value={codeCRED}", ENDITEM,
"Name=ctl00$content$txtMerchantCode", "Value=00031", ENDITEM,
"Name=ctl00$content$txtPosCode", "Value=0001", ENDITEM,
"Name=ctl00$content$txtPosPerson", "Value=0001", ENDITEM,
LAST);

lr_start_transaction("fillFaxProd1");

// Function selecting goods from combo box and moving them to a list along with their quantity and price

lr_think_time(40);

// function selecting preferred payment type and scheme

lr_think_time(30);

// function filling personal details – EGN, title, three names, place of birth, number of ID card, validity of ID card, current address

lr_think_time(50);

// function filling personal details – permanent address, what kind of place the client lives in, whose property it is, how long is he/she living there, phone numbers for contact, additional contacts

lr_think_time(50);

// function filling educational degree, marital status, children under 18, details of spouse, details of car owned

lr_think_time(30);

// function filling employment details – name of company, address, contacts, occupation, work experience in the company and as a whole

lr_think_time(40);

// function filling payment details, monthly incomes and expenses, credits taken from banks

lr_think_time(20);

// function sending request for printing documents containing the data from the application

lr_think_time(30);

// function inserting additional notes leaved by the employee and sending the application for further process in the central office

lr_end_transaction("fillFaxProd1", LR_AUTO);

lr_think_time(7);

web_url("ApplicationProcessing.aspx_5",
"URL=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationProcessing.aspx",
"Resource=0",
"RecContentType=text/html",
"Referer=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationSteps.aspx",
"Snapshot=t261.inf",
"Mode=HTML",
LAST);
return 0;
}
```

Приложение 6: Скрипт за изследване производителността на бизнес процеса „Въвеждане на молби, получени по факс за продукт2”, реализиран в средата на Mercury LoadRunner 8.0

FAXPROD2.usr

```
#include "web_api.h"

Action()
{
web_url( "Login.aspx",
"URL=http://10.1.1.203/Jetix/Login.aspx",
"Resource=0",
"RecContentType=text/html",
"Referer=",
"Snapshot=t31.inf",
"Mode=HTML",
LAST);
lr_think_time(4);

lr_start_transaction("loginFaxProd2");

web_submit_form("Login.aspx_2",
"Snapshot=t2.inf",
ITEMDATA,
"Name=lgnJetix$UserName", "Value={user}", ENDITEM,
"Name=lgnJetix$Password", "Value={pass}", ENDITEM,
"Name=lgnJetix$LoginButton", "Value=Log In", ENDITEM,
LAST);

lr_end_transaction("loginFaxProd2", LR_AUTO);

lr_think_time(6);

web_url("ApplicationProcessing.aspx",
"URL=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationProcessing.aspx",
"Resource=0",
"RecContentType=text/html",
"Referer=http://10.1.1.203/Jetix/Default.aspx",
"Snapshot=t8.inf",
"Mode=HTML",
LAST);

web_submit_data("ApplicationProcessing.aspx_2",
"Action=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationProcessing.aspx",
"Method=POST",
"RecContentType=text/html",
"Referer=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationProcessing.aspx",
"Snapshot=t14.inf",
"Mode=HTML",
ITEMDATA,
"Name=__LASTFOCUS", "Value=", ENDITEM,
"Name=__EVENTTARGET", "Value=ctl00$content$ddlProduct", ENDITEM,
"Name=__EVENTARGUMENT", "Value=", ENDITEM,
"Name=ctl00$content$ddlProduct", "Value=1", ENDITEM,
LAST);

lr_think_time(29);

web_submit_data("ApplicationProcessing.aspx_3",
"Action=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationProcessing.aspx",
"Method=POST",
```

```
"RecContentType=text/html",
"Referer=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationProcessing.aspx",
"Snapshot=t23.inf",
"Mode=HTML",
ITEMDATA,
"Name=__LASTFOCUS", "Value=", ENDITEM,
"Name=__EVENTTARGET", "Value=ctl00$content$btnGo", ENDITEM,
"Name=__EVENTARGUMENT", "Value=", ENDITEM,
"Name=ctl00$content$ddlProduct", "Value=1", ENDITEM,
"Name=ctl00$content$txtCreditCashCode", "Value={nmbr}", ENDITEM,
"Name=ctl00$content$txtLocalOfficeCode", "Value=PVM_LO_001", ENDITEM,
"Name=ctl00$content$txtCashEmployeeCode", "Value=0001", ENDITEM,
"Name=ctl00$content$cCashSignDate$txtDate", "Value={date}", ENDITEM,
LAST);

lr_start_transaction("fillFaxProd2");

// function selecting preferred payment type and scheme

lr_think_time(55);

// function filling personal details – EGN, title, three names, place of birth, number of ID card, validity of ID card, current address

lr_think_time(45);

// function filling personal details – permanent address, what kind of place the client lives in, whose property it is, how long is he/she living there, phone numbers for contact, additional contacts

lr_think_time(60);

// function filling educational degree, marital status, children under 18, details of spouse, details of car owned

lr_think_time(35);

// function filling employment details – name of company, address, contacts, occupation, work experience in the company and as a whole

lr_think_time(30);

// function filling payment details, monthly incomes and expenses, credits taken from banks

lr_think_time(25);

// function sending request for printing documents containing the data from the application

lr_think_time(25);

// function inserting additional notes leaved by the employee and sending the application for further process in the central office

lr_end_transaction("fillFaxProd2", LR_AUTO);

lr_think_time(7);

web_url("ApplicationProcessing.aspx_4",
"URL=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationProcessing.aspx",
"Resource=0",
"RecContentType=text/html",
"Referer=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationSteps.aspx",
"Snapshot=t204.inf",
"Mode=HTML",
LAST);

return 0;
}
```

Приложение 7: Скрипт за изследване производителността на бизнес процеса „Въвеждане онлайн на молби за продукт1”, реализиран в средата на Mercury LoadRunner 8.0

ONLINEPROD1.usr

```
-------------------------------------------------------
#include "web_api.h"

Action()
{
web_url( ""Login.aspx",
"URL=http://10.1.1.203/Jetix/Login.aspx",
"Resource=0",
"RecContentType=text/html",
"Referer=",
"Snapshot=t1.inf",
"Mode=HTML",
LAST);

lr_think_time(3);
lr_start_transaction("loginOnlineProd1");
web_submit_form("Login.aspx_2",
"Snapshot=t2.inf",
ITEMDATA,
"Name=lgnJetix$UserName", "Value={user}", ENDITEM,
"Name=lgnJetix$Password", "Value={pass}", ENDITEM,
"Name=lgnJetix$LoginButton", "Value=Log In", ENDITEM,
LAST);
lr_end_transaction("loginOnlineProd1", LR_AUTO);

web_link("Нова молба",
"Text=Нова Молба",
"Snapshot=t5.inf",
LAST);
web_url("JetIX_logo.gif_2",
"URL=http://10.1.1.203/Jetix/Design/Images/JetIX_logo.gif",
"Resource=1",
"RecContentType=image/gif",
"Referer=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationProcessing.aspx",
"Snapshot=t6.inf",
LAST);

lr_think_time( 1 );

web_submit_data("ApplicationProcessing.aspx",
"Action=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationProcessing.aspx",
"Method=POST",
"RecContentType=text/html",
"Referer=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationProcessing.aspx",
"Snapshot=t7.inf",
"Mode=HTML",
ITEMDATA,
"Name=__LASTFOCUS", "Value=", ENDITEM,
"Name=__EVENTTARGET", "Value=ctl00$content$ddlProduct", ENDITEM,
"Name=__EVENTARGUMENT", "Value=", ENDITEM,
"Name=ctl00$content$ddlProduct", "Value=0", ENDITEM,
LAST);

web_url("JetIX_logo.gif_3",
"URL=http://10.1.1.203/Jetix/Design/Images/JetIX_logo.gif",
"Resource=1",
"RecContentType=image/gif",
"Referer=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationProcessing.aspx",
```

```
"Snapshot=t8.inf",
LAST);
lr_think_time( 5 );
web_submit_data("ApplicationProcessing.aspx_2",
"Action=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationProcessing.aspx",
"Method=POST",
"RecContentType=text/html",
"Referer=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationProcessing.aspx",
"Snapshot=t9.inf",
"Mode=HTML",
ITEMDATA,
"Name=__LASTFOCUS", "Value=", ENDITEM,
"Name=__EVENTTARGET", "Value=ctl00$content$btnGo", ENDITEM,
"Name=__EVENTARGUMENT", "Value=", ENDITEM,
"Name=ctl00$content$ddlProduct", "Value=0", ENDITEM,
LAST);

lr_start_transaction("fillOnlineProd1");

// Function selecting goods from combo box and moving them to a list along with their quantity and price

lr_think_time(40);

// function selecting preferred payment type and scheme

lr_think_time(30);

// function filling personal details – EGN, title, three names, place of birth, number of ID card, validity of ID card, current address

lr_think_time(50);

// function filling personal details – permanent address, what kind of place the client lives in, whose property it is, how long is he/she living there, phone numbers for contact, additional contacts

lr_think_time(50);

// function filling educational degree, marital status, children under 18, details of spouse, details of car owned

lr_think_time(30);

// function filling employment details – name of company, address, contacts, occupation, work experience in the company and as a whole

lr_think_time(40);

// function filling payment details, monthly incomes and expenses, credits taken from banks

lr_think_time(20);

// function sending request for printing documents containing the data from the application

lr_think_time(30);

// function inserting additional notes leaved by the employee and sending the application for further process in the central office

lr_end_transaction("fillOnlineProd1", LR_AUTO);
lr_think_time( 5 );
web_url("ApplicationProcessing.aspx_3",
"URL=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationProcessing.aspx",
"Resource=0",
"RecContentType=text/html",
"Referer=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationSteps.aspx",
"Snapshot=t277.inf",
"Mode=HTML",
LAST);
return 0;
}
```

Приложение 8: Скрипт за изследване производителността на бизнес процеса „Въвеждане онлайн на молби за продукт2”, реализиран в средата на Mercury LoadRunner 8.0

ONLINEPROD2.usr

```
------------------------------------------------------------
#include "web_api.h"

Action()
{
web_url( "Login.aspx",
"URL=http://10.1.1.203/Jetix/Login.aspx",
"Resource=0",
"RecContentType=text/html",
"Referer=",
"Snapshot=t31.inf",
"Mode=HTML",
LAST);
lr_think_time(1);

lr_start_transaction("loginOnlineProd2");

web_submit_form("Login.aspx_2",
"Snapshot=t2.inf",
ITEMDATA,
"Name=lgnJetix$UserName", "Value={user}", ENDITEM,
"Name=lgnJetix$Password", "Value={pass}", ENDITEM,
"Name=lgnJetix$LoginButton", "Value=Log In", ENDITEM,
LAST);

lr_end_transaction("loginOnlineProd2", LR_AUTO);

lr_think_time(12);


web_link("Нова молба",
"Text=Нова молба",
"Snapshot=t3.inf",
LAST);

web_url("JetIX_logo.gif_2",
"URL=http://10.1.1.203/Jetix/Design/Images/JetIX_logo.gif",
"Resource=1",
"RecContentType=image/gif",
"Referer=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationProcessing.aspx",
LAST);

lr_think_time(1);

web_submit_data("ApplicationProcessing.aspx",
"Action=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationProcessing.aspx",
"Method=POST",
"RecContentType=text/html",
"Referer=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationProcessing.aspx",
"Snapshot=t4.inf",
"Mode=HTML",
ITEMDATA,
"Name=__LASTFOCUS", "Value=", ENDITEM,
"Name=__EVENTTARGET", "Value=ctl00$content$ddlProduct", ENDITEM,
"Name=__EVENTARGUMENT", "Value=", ENDITEM,
"Name=ctl00$content$ddlProduct", "Value=0", ENDITEM,
LAST);

web_url("JetIX_logo.gif_3",
"URL=http://10.1.1.203/Jetix/Design/Images/JetIX_logo.gif",
```

```
"Resource=1",
"RecContentType=image/gif",
"Referer=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationProcessing.aspx",
LAST);


web_submit_data("ApplicationProcessing.aspx_2",
"Action=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationProcessing.aspx",
"Method=POST",
"RecContentType=text/html",
"Referer=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationProcessing.aspx",
"Snapshot=t5.inf",
"Mode=HTML",
ITEMDATA,
"Name=__LASTFOCUS", "Value=", ENDITEM,
"Name=__EVENTTARGET", "Value=ctl00$content$ddlProduct", ENDITEM,
"Name=__EVENTARGUMENT", "Value=", ENDITEM,
"Name=ctl00$content$ddlProduct", "Value=3", ENDITEM,
LAST);
web_url("JetIX_logo.gif_4",
"URL=http://10.1.1.203/Jetix/Design/Images/JetIX_logo.gif",
"Resource=1",
"RecContentType=image/gif",
"Referer=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationProcessing.aspx",
LAST);

web_submit_data("ApplicationProcessing.aspx_3",
"Action=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationProcessing.aspx",
"Method=POST",
"RecContentType=text/html",
"Referer=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationProcessing.aspx",
"Snapshot=t6.inf",
"Mode=HTML",
ITEMDATA,
"Name=__LASTFOCUS", "Value=", ENDITEM,
"Name=__EVENTTARGET", "Value=ctl00$content$btnGo", ENDITEM,
"Name=__EVENTARGUMENT", "Value=", ENDITEM,
"Name=ctl00$content$ddlProduct", "Value=3", ENDITEM,
LAST);

lr_start_transaction("fillOnlineProd2");

// function selecting preferred payment type and scheme

lr_think_time(55);

// function filling personal details – EGN, title, three names, place of birth, number of ID card, validity of ID card, current address

lr_think_time(45);

// function filling personal details – permanent address, what kind of place the client lives in, whose property it is, how long is he/she living there, phone numbers for contact, additional contacts

lr_think_time(60);

// function filling educational degree, marital status, children under 18, details of spouse, details of car owned

lr_think_time(35);

// function filling employment details – name of company, address, contacts, occupation, work experience in the company and as a whole

lr_think_time(30);

// function filling payment details, monthly incomes and expenses, credits taken from banks

lr_think_time(25);

// function selecting where the client wants his money to be transferred – personal account, account created by the
```

```
creditor

lr_think_time(30);

// function sending request for printing documents containing the data from the application

lr_think_time(25);

// function inserting additional notes leaved by the employee and sending the application for further process in the central
office
lr_think_time(20);
lr_end_transaction("fillFaxProd2", LR_AUTO);

web_url("ApplicationProcessing.aspx_4",
"URL=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationProcessing.aspx",
"Resource=0",
"RecContentType=text/html",
"Referer=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationSteps.aspx",
"Snapshot=t16.inf",
"Mode=HTML",
LAST);

web_url("JetIX_logo.gif_6",
"URL=http://10.1.1.203/Jetix/Design/Images/JetIX_logo.gif",
"Resource=1",
"RecContentType=image/gif",
"Referer=http://10.1.1.203/Jetix/Modules/ApplicationProcessing/ApplicationProcessing.aspx",
LAST);

return 0;
```

Приложение 9: Резултати при натоварване 200 молби в час

Analysis Summary

Period: 17.09.2007 16:57:35 - 17.09.2007 18:03:42

Scenario Name: Scenario1
Results in Session: D:\diploma\Results to be used\result40\result40\result40.lrr
Duration: 1 hour, 6 minutes and 7 seconds.

Statistics Summary

Maximum Running Vusers:	40	
Total Throughput (bytes):	909 091 248	
Average Throughput (bytes/second):	251 965	
Total Hits:	27 562	
Average Hits per Second:	7.639	View HTTP Responses Summary

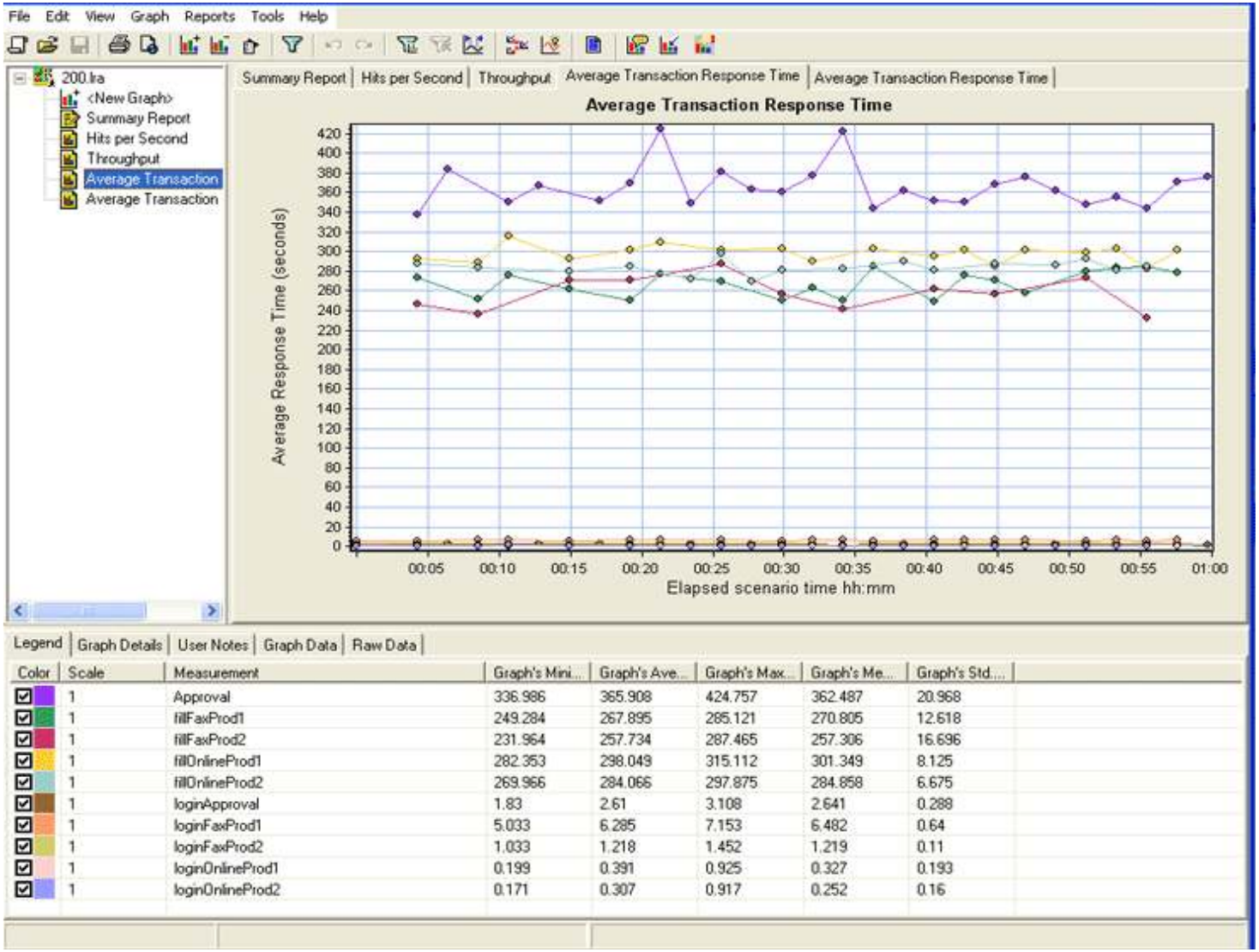
Transaction Summary

Transactions: Total Passed: 837 Total Failed: 0 Total Stopped: 0 Average Response Time

Transaction Name	Minimum	Average	Maximum	Std. Deviation	90 Percent	Pass	Fail	Stop
Approval	293.038	362.872	431.938	26.435	397.9	179	0	0
fillFaxProd1	249.284	267.97	285.121	12.263	283.255	22	0	0
fillFaxProd2	231.964	257.734	287.465	16.696	273.979	11	0	0
fillOnlineProd1	275.755	298.51	321.671	10.347	313.724	132	0	0
fillOnlineProd2	264.093	284.46	311.096	11.353	298.856	55	0	0
loginApproval	1.797	2.646	3.739	0.445	3.203	198	0	0
loginFaxProd1	5.033	6.224	7.153	0.636	7.072	24	0	0
loginFaxProd2	1.033	1.218	1.452	0.11	1.342	12	0	0
loginOnlineProd1	0.148	0.419	1.525	0.3	0.947	144	0	0
loginOnlineProd2	0.147	0.297	1.335	0.176	0.494	60	0	0

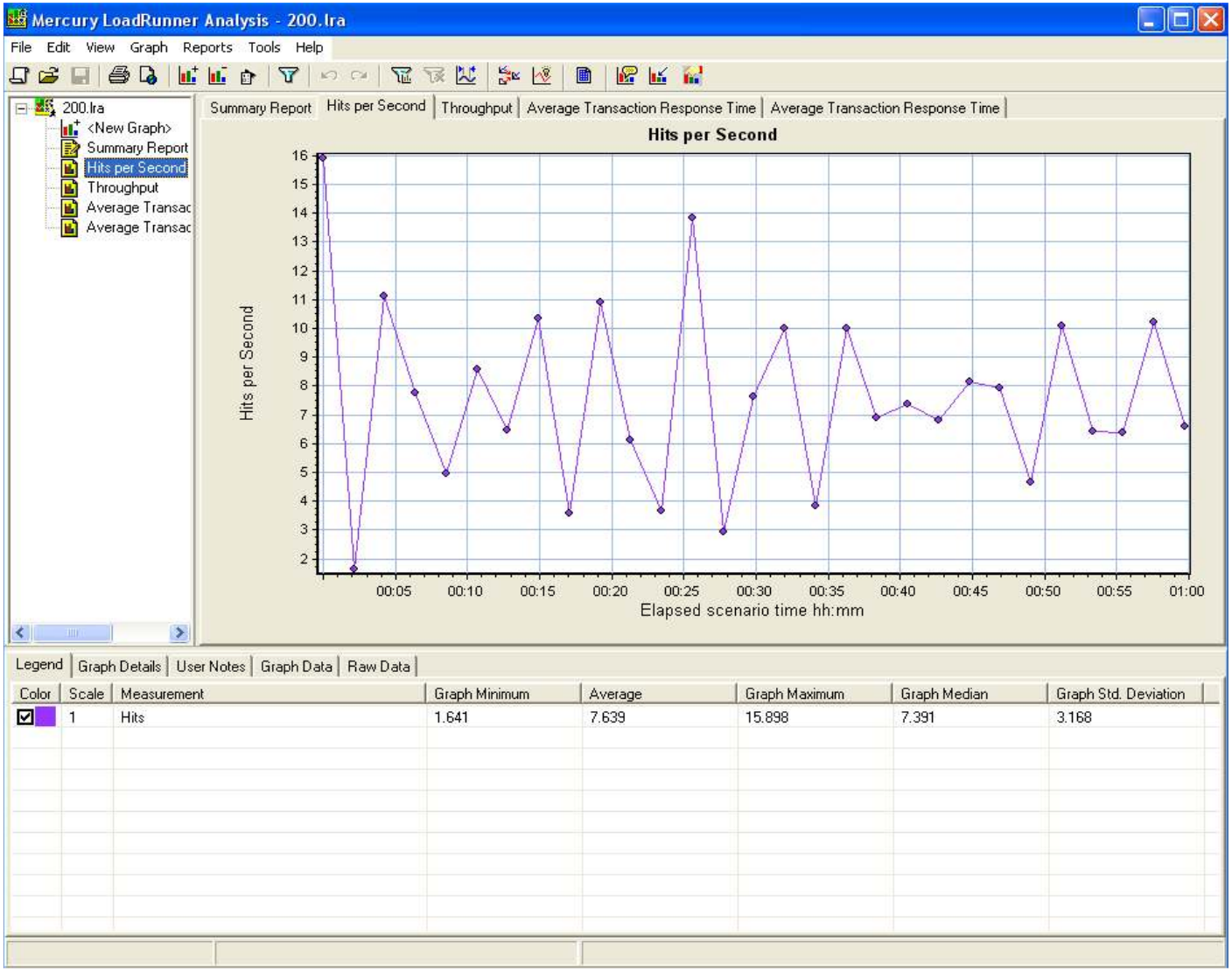
HTTP Responses Summary

HTTP Responses	Total	Per second
HTTP_200	26 248	7.275
HTTP_302	1 314	0.364

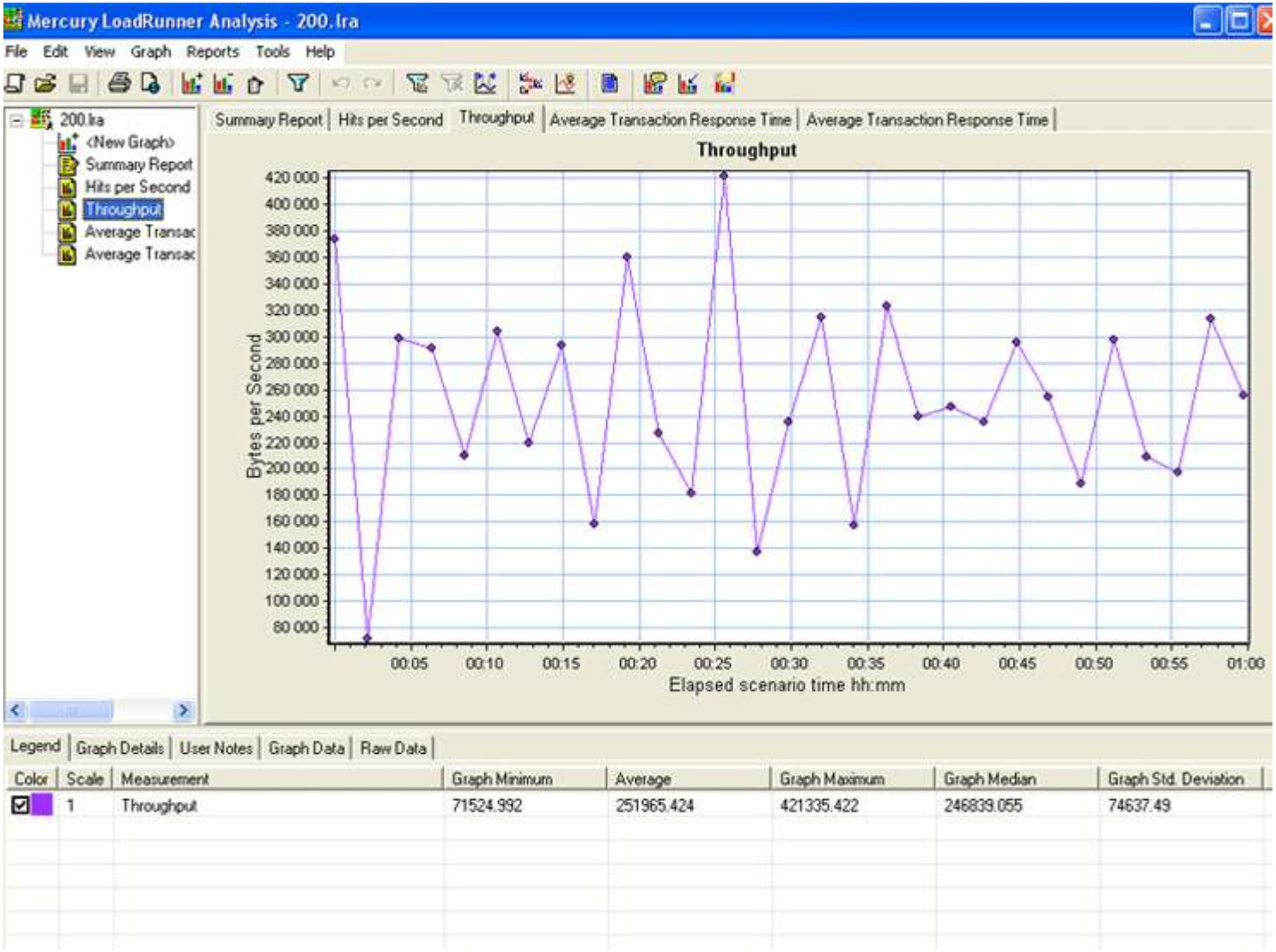


Color	Scale	Measurement	Graph's Mini...	Graph's Ave...	Graph's Max...	Graph's Me...	Graph's Std....
	1	Approval	336.986	365.908	424.757	362.487	20.968
	1	fillFaxProd1	249.284	267.895	285.121	270.805	12.618
	1	fillFaxProd2	231.964	257.734	287.465	257.306	16.696
	1	fillOnlineProd1	282.353	298.049	315.112	301.349	8.125
	1	fillOnlineProd2	269.966	284.066	297.875	284.858	6.675
	1	loginApproval	1.83	2.61	3.108	2.641	0.288
	1	loginFaxProd1	5.033	6.285	7.153	6.482	0.64
	1	loginFaxProd2	1.033	1.218	1.452	1.219	0.11
	1	loginOnlineProd1	0.199	0.391	0.925	0.327	0.193
	1	loginOnlineProd2	0.171	0.307	0.917	0.252	0.16

Фигура 11: Средно време за отговор на различните транзакции през цялата продължителност на сценария



Фигура 12: Брой заявки към сървъра в секунда за цялата продължителност на сценария



Фигура 13: Обем на данните изпратени към сървъра за цялата продължителност на сценария

Приложение 10: Резултати при натоварване 600 молби в час

Analysis Summary

Period: 17.09.2007 14:05:30 - 17.09.2007 15:12:23

Scenario Name: Scenario1
Results in Session: D:\diploma\result17-11-2007\result17-11-2007.lrr
Duration: 1 hour, 6 minutes and 53 seconds.

Statistics Summary

Maximum Running Vusers:	120	
Total Throughput (bytes):	3 048 243 954	
Average Throughput (bytes/second):	834 221	
Total Hits:	83 085	
Average Hits per Second:	22.738	View HTTP Responses Summary

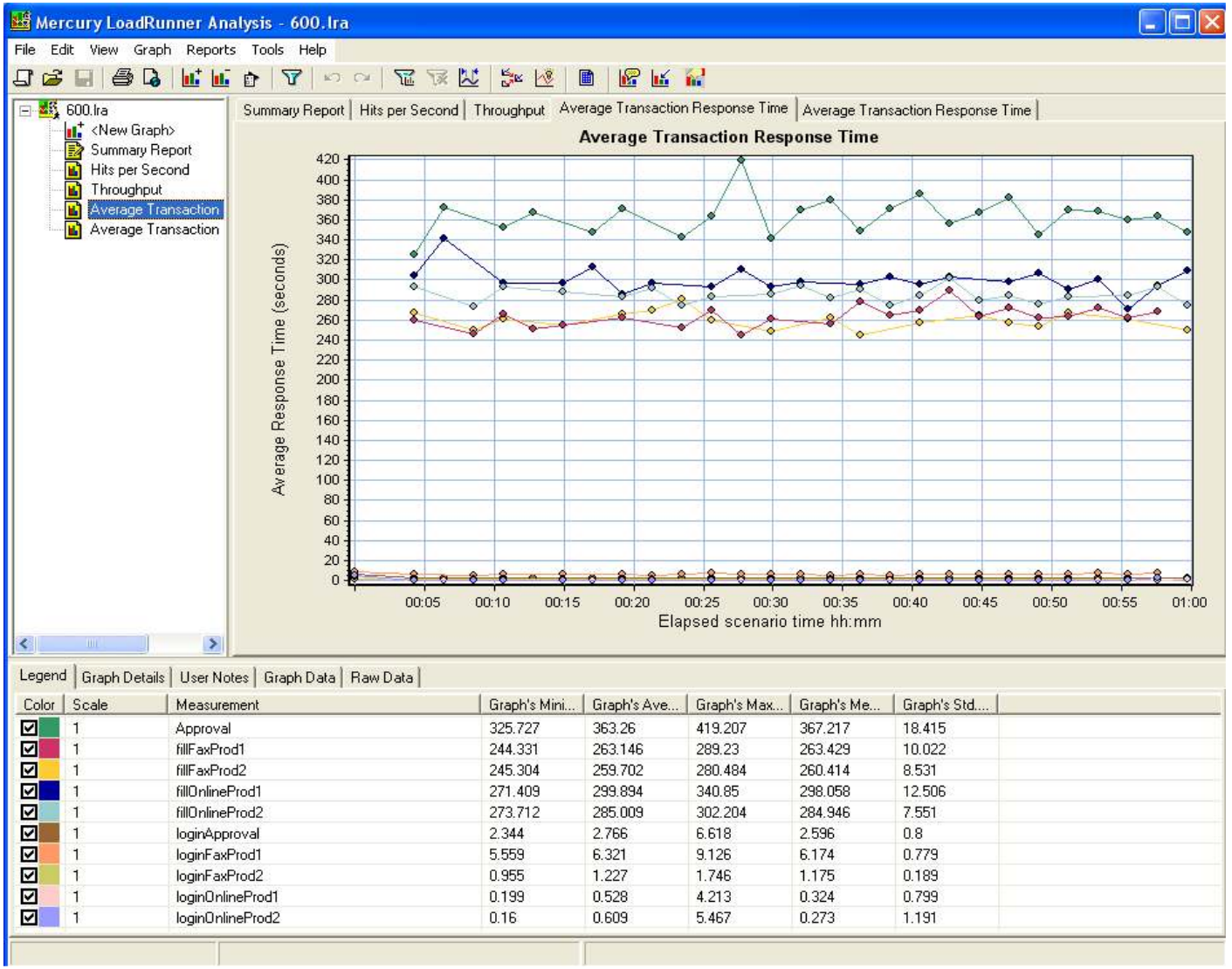
Transaction Summary

Transactions: Total Passed: 2 513 Total Failed: 0 Total Stopped: 0 Average Response Time

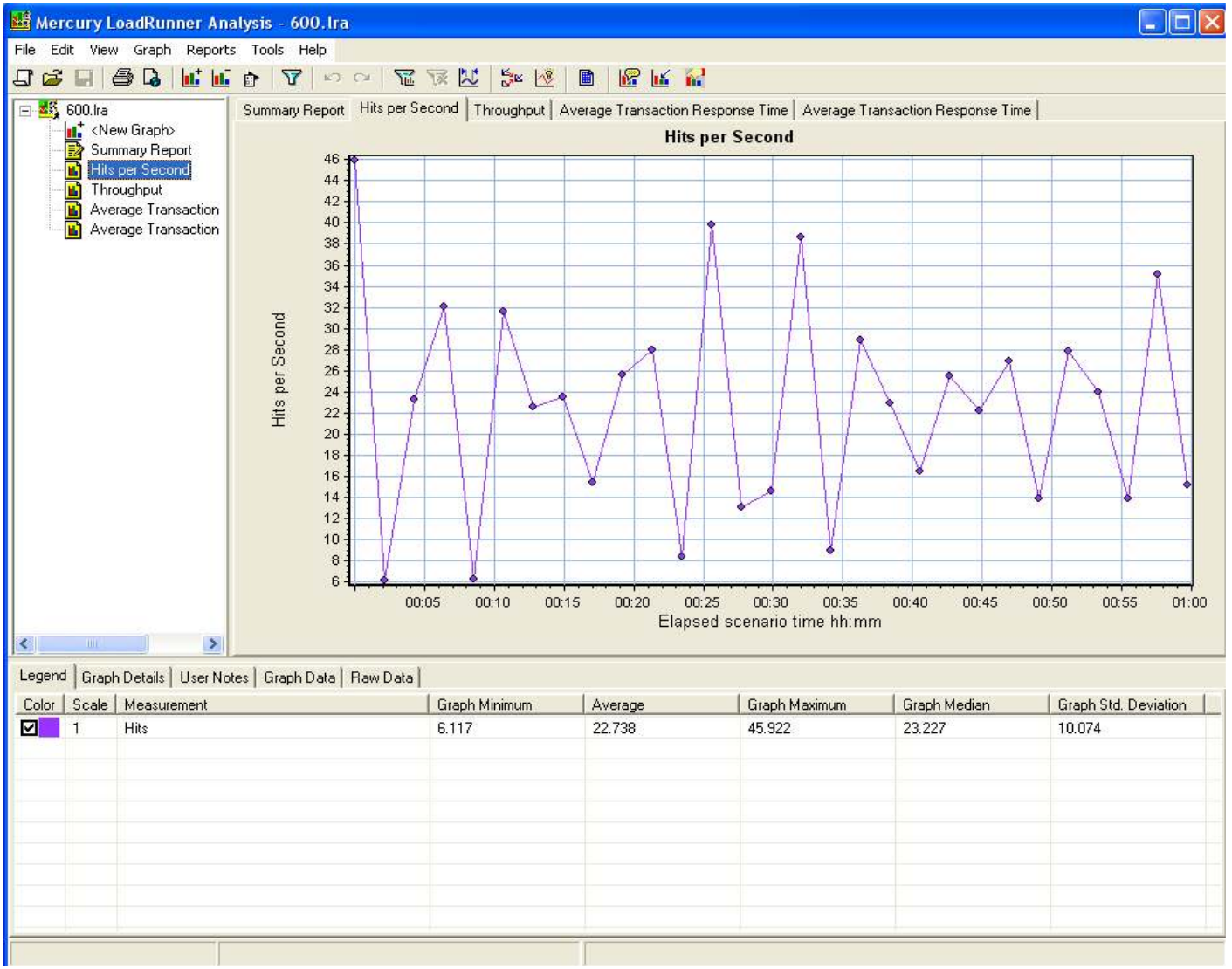
Transaction Name	Minimum	Average	Maximum	Std. Deviation	90 Percent	Pass	Fail	Stop
Approval	291.398	364.467	428.289	24.189	398.698	539	0	0
fillFaxProd1	233.5	262.98	290.673	13.768	279.062	66	0	0
fillFaxProd2	230.756	259.711	293.515	13.167	279.555	34	0	0
fillOnlineProd1	264.517	297.397	340.85	12.857	313.931	396	0	0
fillOnlineProd2	256.441	286.459	318.876	12.073	302.14	166	0	0
loginApproval	1.746	3.031	12.517	1.448	3.617	594	0	0
loginFaxProd1	4.93	6.433	12.249	1.301	7.501	72	0	0
loginFaxProd2	0.91	1.247	2.376	0.33	1.514	36	0	0
loginOnlineProd1	0.143	0.66	5.501	1.129	1.018	430	0	0
loginOnlineProd2	0.156	0.767	11.365	1.704	0.675	180	0	0

HTTP Responses Summary

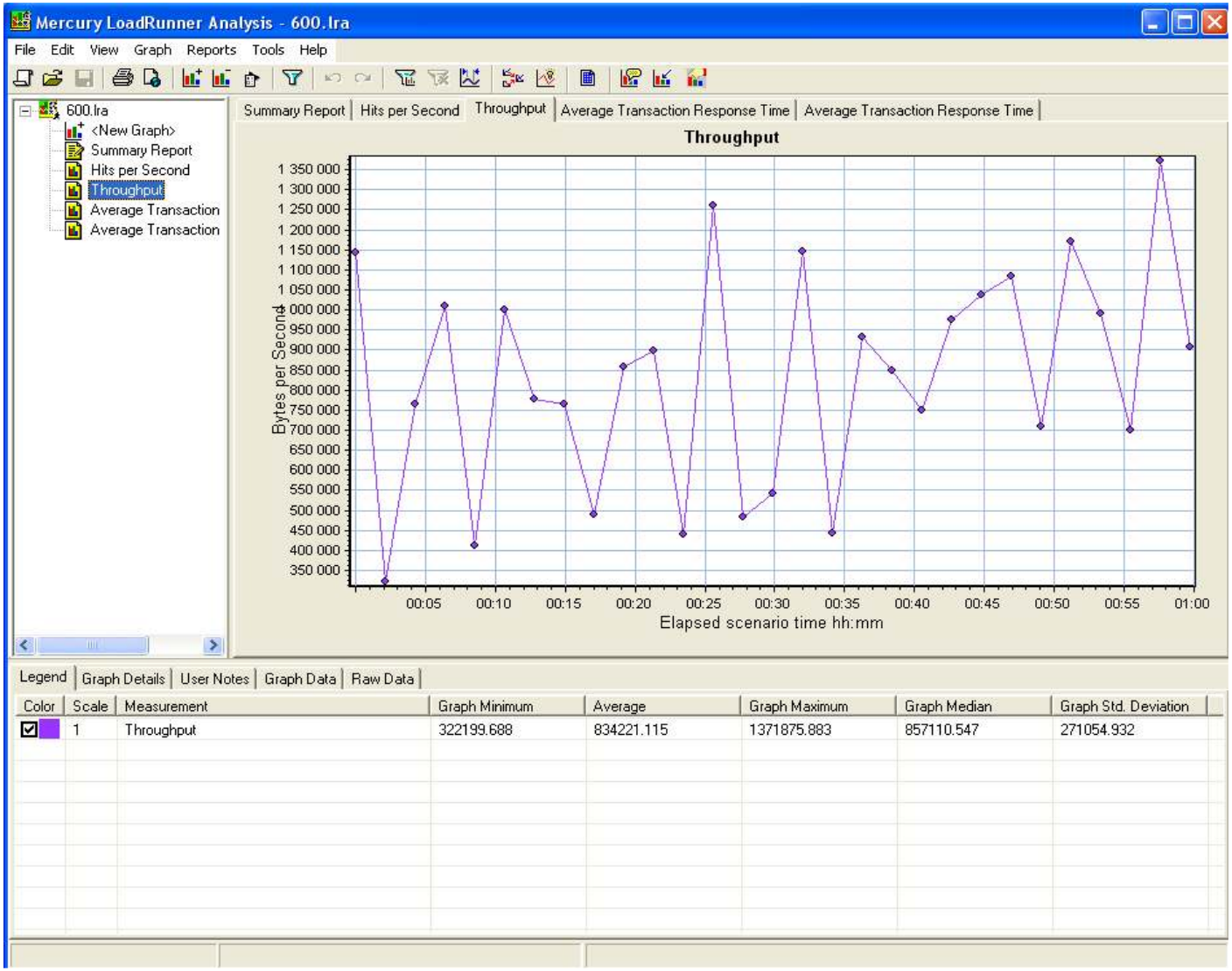
HTTP Responses	Total	Per second
HTTP_200	79 108	21.65
HTTP_302	3 977	1.088



Фигура 14: Средно време за отговор на различните транзакции през цялата продължителност на сценария



Фигура 15: Брой заявки към сървъра в секунда за цялата продължителност на сценария



Фигура 16: Обем на данните изпратени към сървъра за цялата продължителност на сценария

Приложение 11: Резултати при натоварване 1000 молби в час

Analysis Summary

Period: 17.09.2007 15:46:53 - 17.09.2007 16:53:46

Scenario Name: Scenario1
Results in Session: D:\diploma\result200\result200.lrr
Duration: 1 hour, 6 minutes and 53 seconds.

Statistics Summary

Maximum Running Vusers:	200
Total Throughput (bytes):	4 785 928 913
Average Throughput (bytes/second):	1 309 778
Total Hits:	132 959
Average Hits per Second:	36.387

View HTTP Responses Summary

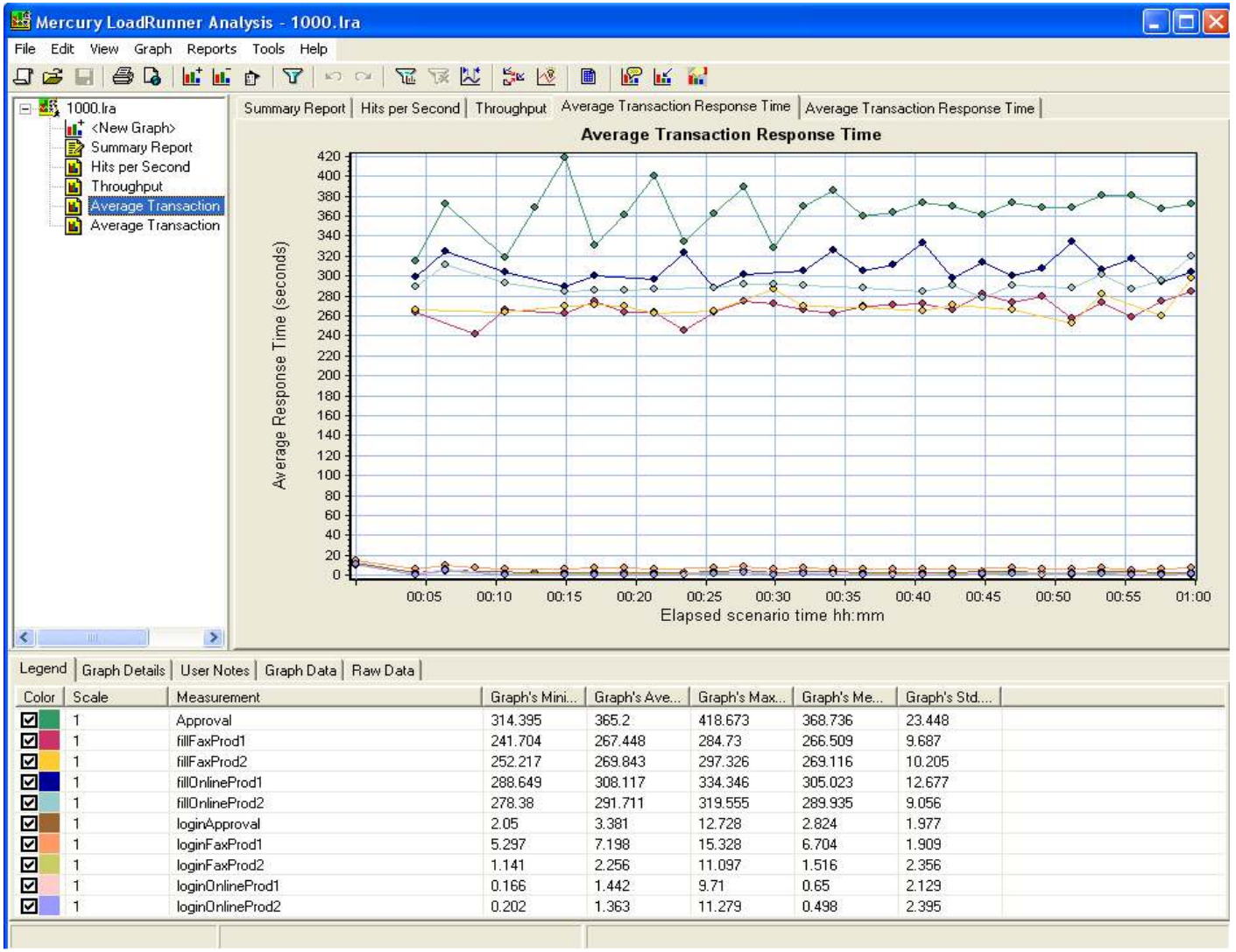
Transaction Summary

Transactions: Total Passed: 4 057 Total Failed: 0 Total Stopped: 0 Average Response Time

Transaction Name	Minimum	Average	Maximum	Std. Deviation	90 Percent	Pass	Fail	Stop
Approval	286.473	368.03	434.815	25.824	400.747	881	0	0
fillFaxProd1	239.37	267.509	295.252	11.935	282.298	110	0	0
fillFaxProd2	240.825	267.751	297.326	11.429	284.637	55	0	0
fillOnlineProd1	257.068	303.668	343.927	13.943	322.209	638	0	0
fillOnlineProd2	255.335	289.972	325.11	12.207	304.884	275	0	0
loginApproval	1.78	4.269	15.093	3.123	11.286	955	0	0
loginFaxProd1	5.004	7.491	16.244	2.595	10.002	117	0	0
loginFaxProd2	0.905	2.355	11.93	2.705	3.358	60	0	0
loginOnlineProd1	0.145	2.018	11.154	3.015	8.842	667	0	0
loginOnlineProd2	0.159	1.605	11.981	3.099	3.028	299	0	0

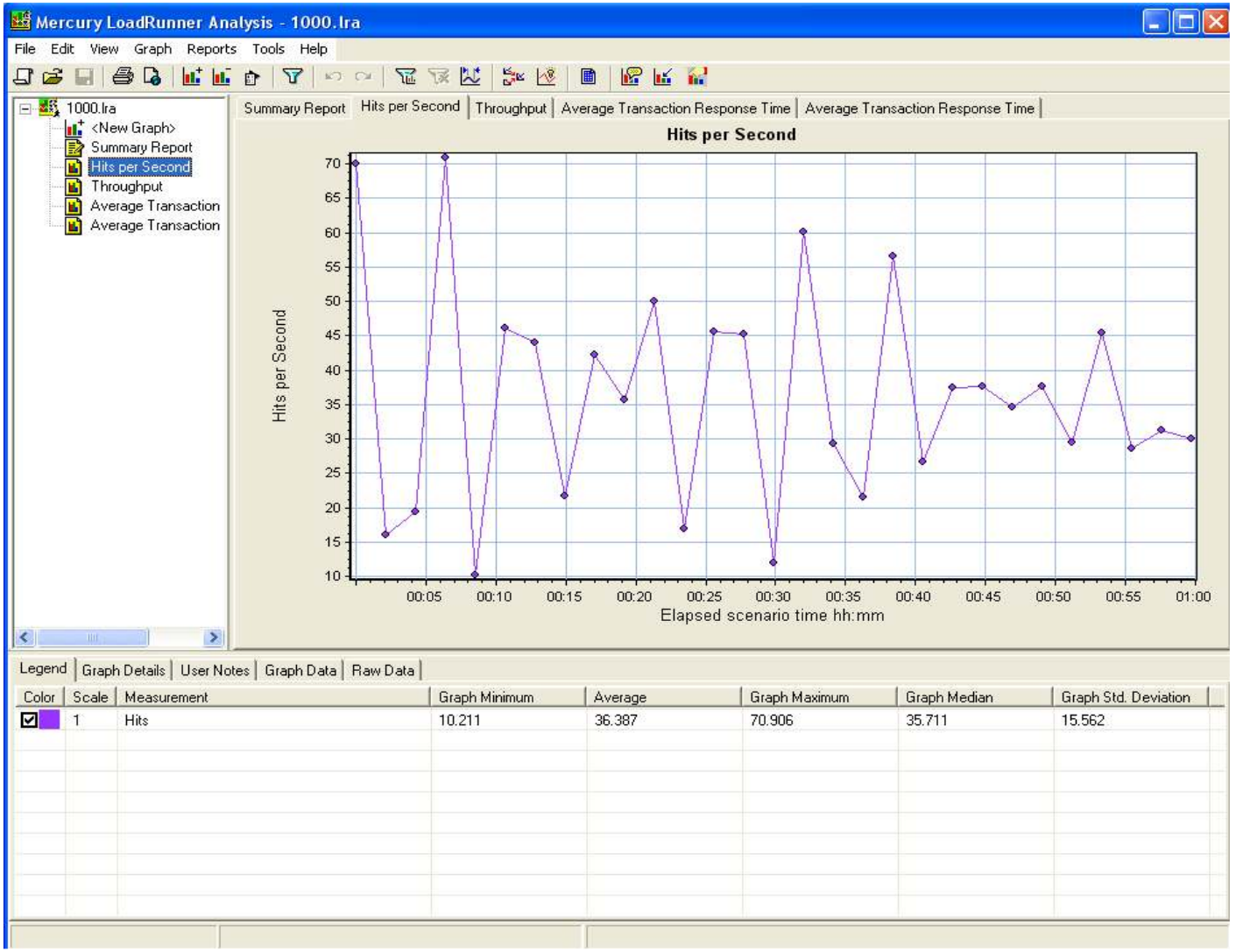
HTTP Responses Summary

HTTP Responses	Total	Per second
HTTP_200	126 602	34.648
HTTP_302	6 357	1.74

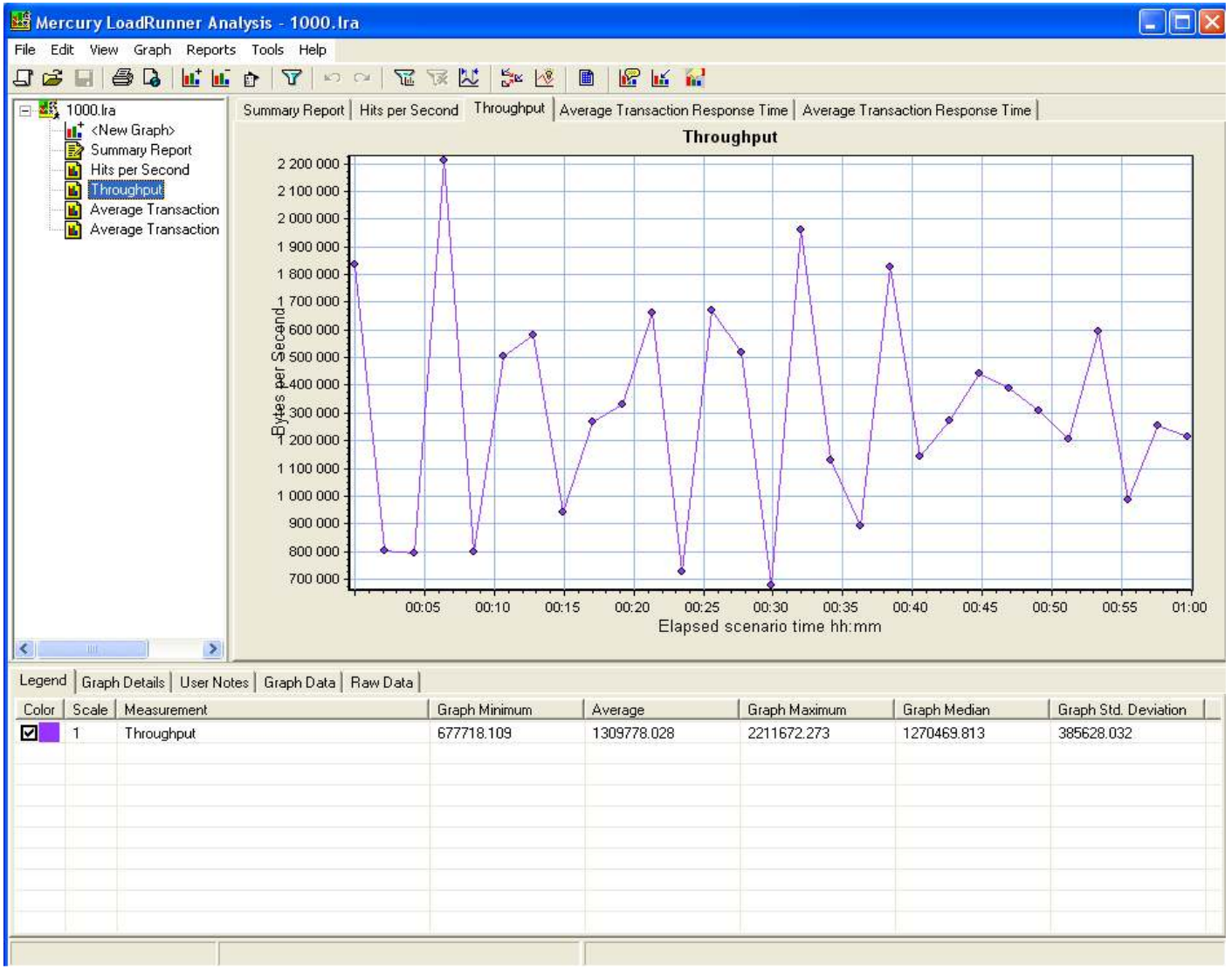


Color	Scale	Measurement	Graph's Mini...	Graph's Ave...	Graph's Max...	Graph's Me...	Graph's Std....
	1	Approval	314.395	365.2	418.673	368.736	23.448
	1	fillFaxProd1	241.704	267.448	284.73	266.509	9.687
	1	fillFaxProd2	252.217	269.843	297.326	269.116	10.205
	1	fillOnlineProd1	288.649	308.117	334.346	305.023	12.677
	1	fillOnlineProd2	278.38	291.711	319.555	289.935	9.056
	1	loginApproval	2.05	3.381	12.728	2.824	1.977
	1	loginFaxProd1	5.297	7.198	15.328	6.704	1.909
	1	loginFaxProd2	1.141	2.256	11.097	1.516	2.356
	1	loginOnlineProd1	0.166	1.442	9.71	0.65	2.129
	1	loginOnlineProd2	0.202	1.363	11.279	0.498	2.395

Фигура 17: Средно време за отговор на различните транзакции през цялата продължителност на сценария



Фигура 18: Брой заявки към сървъра в секунда за цялата продължителност на сценария



Фигура 19: Обем на данните изпратени към сървъра за цялата продължителност на сценария

Приложение 12: Резултати при 1400 молби в час

Analysis Summary

Period: 13.09.2007 17:10:38 - 13.09.2007 18:19:24

Scenario Name: Scenario1
Results in Session: D:\diploma\20070914-3\20070914-3\res\res.lrr
Duration: 1 hour, 8 minutes and 46 seconds.

Statistics Summary

Maximum Running Vusers:	280
Total Throughput (bytes):	8 390 111 185
Average Throughput (bytes/second):	2 300 551
Total Hits:	168 338
Average Hits per Second:	46.158

View HTTP Responses Summary

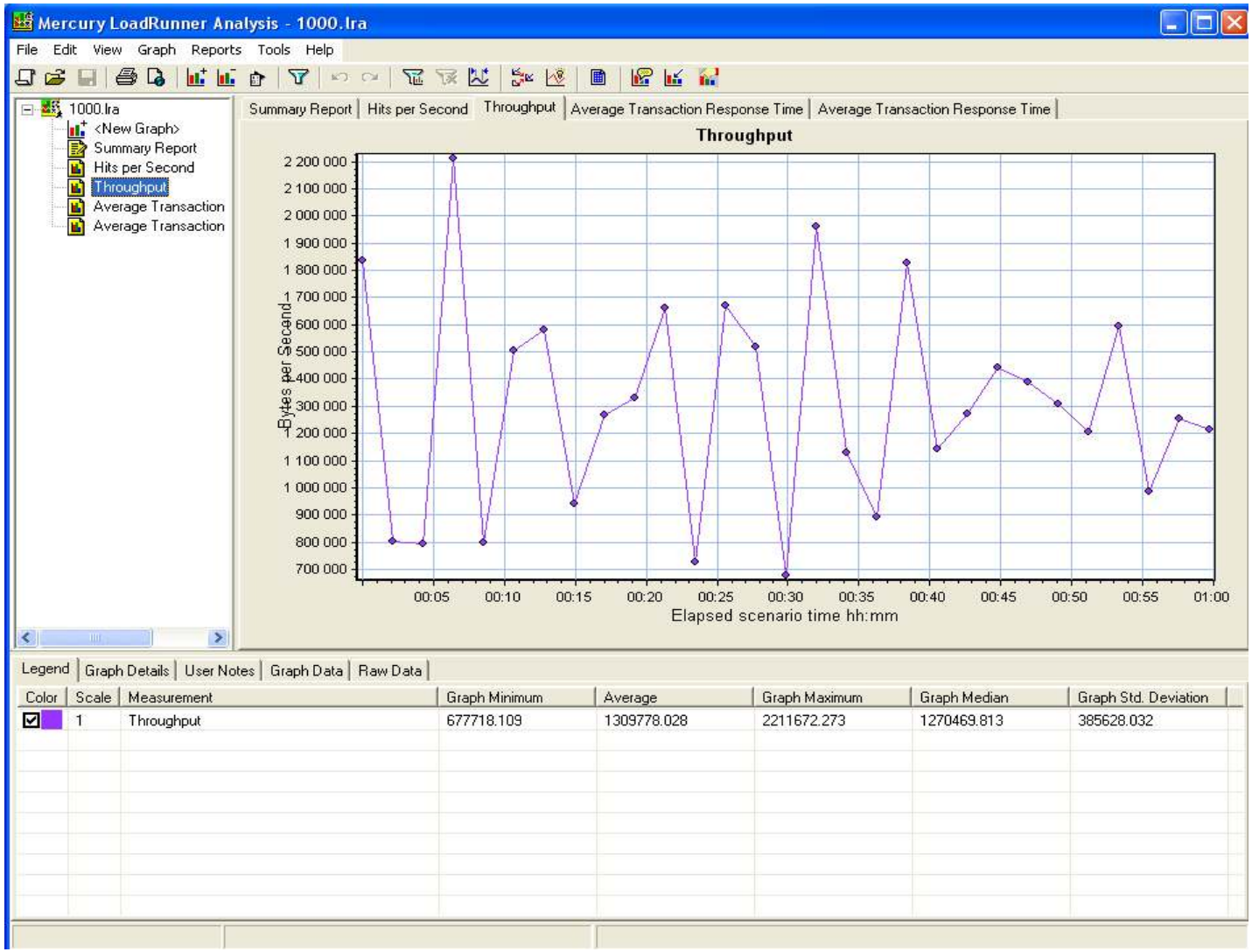
Transaction Summary

Transactions: Total Passed: 5 028 Total Failed: 180 Total Stopped: 0 Average Response Time

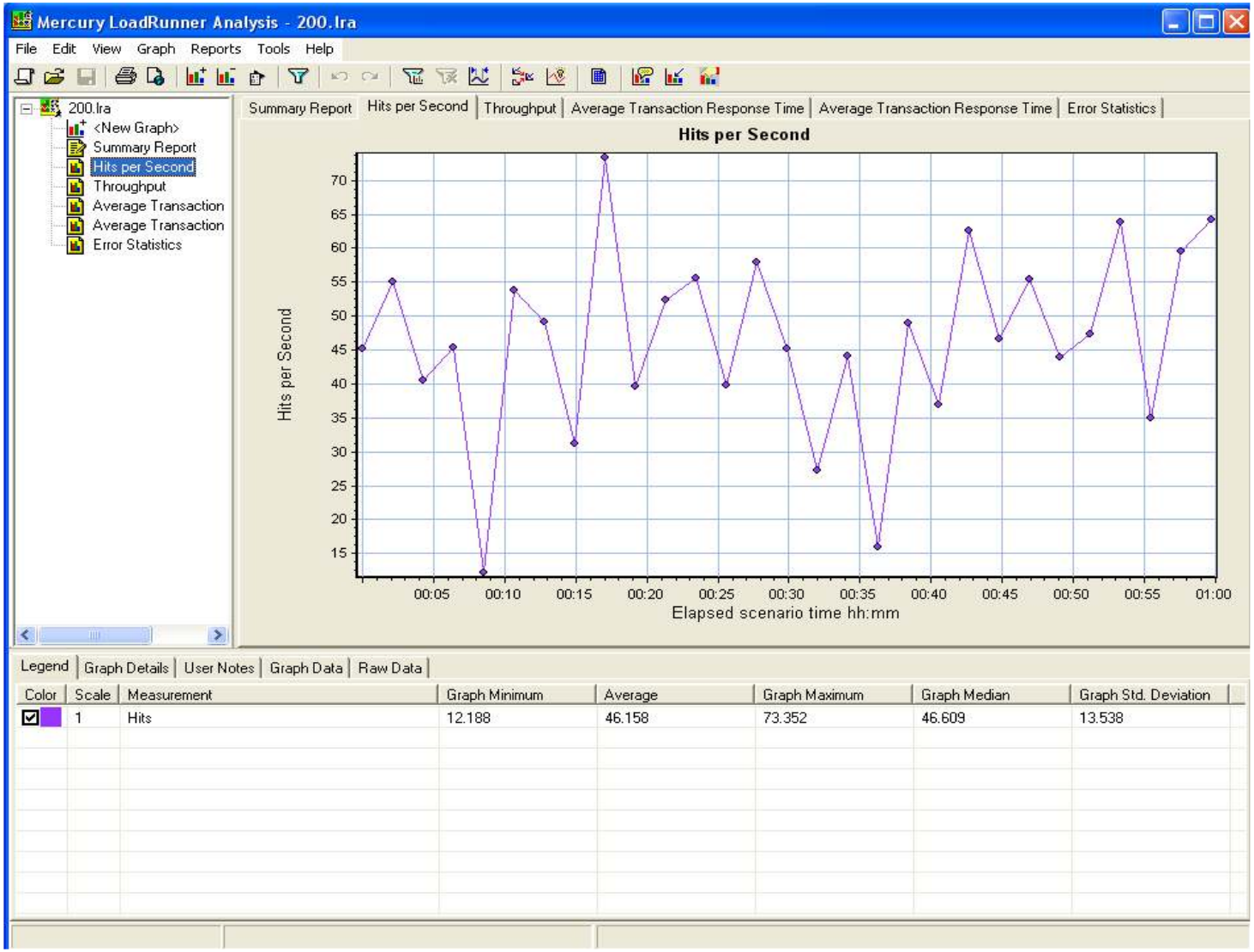
Transaction Name	Minimum	Average	Maximum	Std. Deviation	90 Percent	Pass	Fail	Stop
Approval	291.634	388.919	938.657	58.324	476.687	1 031	55	0
fillFaxProd1	241.365	287.501	475.298	47.629	374.518	136	4	0
fillFaxProd2	239.033	290.31	419.208	46.379	362.802	67	3	0
fillOnlineProd1	260.211	320.722	606.435	47.91	395.021	753	55	0
fillOnlineProd2	263.452	314.423	558.727	50.305	405.223	328	24	0
loginApproval	1.699	3.999	96.794	6.837	4.585	1 195	25	0
loginFaxProd1	4.926	8.233	50.09	5.842	10.189	152	1	0
loginFaxProd2	0.896	3.01	45.279	5.544	4.941	77	0	0
loginOnlineProd1	0.126	4.237	96.748	12.5	6.113	902	13	0
loginOnlineProd2	0.137	2.794	95.402	9.386	4.157	387	0	0

HTTP Responses Summary

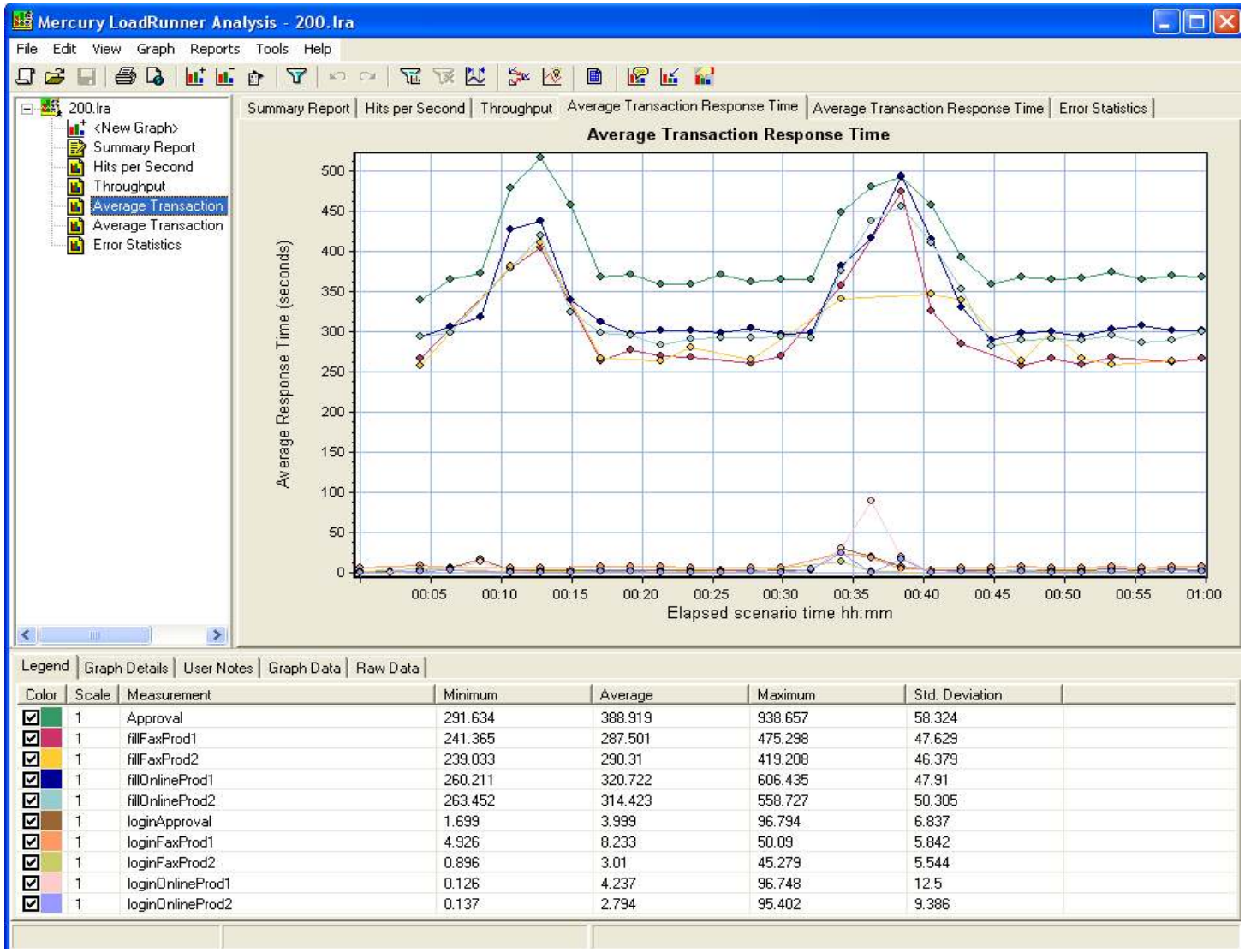
HTTP Responses	Total	Per second
HTTP_200	159 865	43.835
HTTP_302	8 470	2.322
HTTP_500	3	0.001



Фигура 20: Обем на данните изпратени към сървъра за цялата продължителност на сценария



Фигура 21: Брой заявки към сървъра в секунда за цялата продължителност на сценария



Color	Scale	Measurement	Minimum	Average	Maximum	Std. Deviation
	1	Approval	291.634	388.919	938.657	58.324
	1	fillFaxProd1	241.365	287.501	475.298	47.629
	1	fillFaxProd2	239.033	290.31	419.208	46.379
	1	fillOnlineProd1	260.211	320.722	606.435	47.91
	1	fillOnlineProd2	263.452	314.423	558.727	50.305
	1	loginApproval	1.699	3.999	96.794	6.837
	1	loginFaxProd1	4.926	8.233	50.09	5.842
	1	loginFaxProd2	0.896	3.01	45.279	5.544
	1	loginOnlineProd1	0.126	4.237	96.748	12.5
	1	loginOnlineProd2	0.137	2.794	95.402	9.386

Фигура 22: Средно време за отговор на различните транзакции през цялата продължителност на сценария

Термини и съкращения

автентикация – потвърждаване идентичността на потребител

авторизация – потвърждаване на правото на потребител до дадена функционалност

акаунт – съвкупност от данни (потребителско име, парола, права), които осигуряват достъп на потребителя до дадена система

атрибут – характеристика или притежание на даден обект (размер, цвят,...)

бъг (bug) – проблем или пропуск във функционалността на софтуера.

валидация – потвърждаване, че софтуерният продукт отговаря на нуждите на потребителите и техните изисквания.

верификация – потвърждаване, че софтуерът спазва изискванията и функционира коректно

виртуален потребител – виртуалните потребители се използват за симулация на реални потребители от LoadRunner и останалите инструменти за натоварване. По време на изпълнение на тестов сценарий, виртуалните потребители следват действията, които човекът е извършил по време на записа на скрипта. Един сценарий може да съдържа десетки, стотици и дори хиляди виртуални потребители, а да използва само една работна станция.

драйвер (test driver) - софтуер, изпълняващ друг софтуер, който е обект на тестване. Дава възможност за въвеждане на определени входни данни, изпълнение на функционалността от компонента и връщане на изходни данни.

ЕКАТТЕ - Единния класификатор на административно-териториалните и териториалните единици

жизнен цикъл на продукта – Стадиите, през които продуктът минава.

индикация - един или повече резултата от измерване, които помагат за определянето на необходимите измервания за даден

интеграция – процесът на комбиниране/сливане на компоненти

интерфейс – връзка между потребителя и функционалността на програмата

итерация – едно изпълнение на цикъл

клиент-сървър - компютърна архитектура, базирана на идеята, че приложението може да се раздели на клиентска част, която се използва за достъпване на информацията, която от своя страна се съхранява нас сървърната част.

лиценз – документ/ключ/код, даващ официално право за ползване на един продукт за определено време, от определен брой потребители и т.н.

метрика – стандарт за измерване. Софтуерните метрики са статистики, описващи структурата или съдържанието на програма. Всяка метрика трябва да описва реално измерване, като например брой бъгове открити за ред код.

многослойна архитектура – клиент-сървър архитектура, в която приложението се разделя на няколко модула

негативни тестове – тестове, имащи за цел да проверят нещата, които системата не трябва да прави

прототип – модел или копие на система, използвано за оценяване на възможностите й

ресурси – хора, софтуер, хардуер, инструменти – всяка единица, притежаваща качества, които могат да се използват за реализиране на дадени цели.

сесия – продължителността от време от момента на свързване на потребителя със сървъра до момента на прекратяването на тази връзка..

симулация – подражаване на част от поведението или цялото поведение на една система, като се използва друга различна от нея система.

скрипт – скриптът описва последователността от действия, които виртуалния потребител трябва да изпълни по време на теста. Скриптът включва функции, които измерват и записват производителността на отделните компоненти от приложението.

спецификация – Описания на изискванията и подхода за разработка на приложението. Всяка една функционалност трябва да е описана в този документ, заедно с очакваното й поведение.

стесняване (bottleneck) – най-бавната част от системата, която ограничава капацитета й

стъб (stub) – фиктивен компонент, който замества единици от по-ниските нива в йерархията, с цел да улесни тестването.

същност –обект (човек, място, концепция), за който се събират данни

тест план – план, определящ подхода за тестването на базата на функционалната област, която обхваща.

тестови сценарии – индивидуален тест, обхващащ част от функционална област.

транзакция - транзакцията в термините на LoadRunner представлява действие или набор от действия и се използва за измерване на производителността на сървъра. Чрез нея може да бъде засечено времето на всяка от отделните стъпки съставящи скрипта.

функционално тестване (black-box testing) – тестовият обект се разглежда като черна кутия – подават му се входни данни и се следи какви изходни данни връща. Не е необходимо да се познава конкретната реализация и структура на обекта.

холуей тестване (hallway testing) – техника за тестване на използваемостта на софтуера. Името й идва от английската дума за коридор, намеквайки за случайността на извадката от хора, която се ползва при самите тестове.

цел-въпрос-метрика (goal-question-metric) - парадигма, която е разработена от Базили и Ромбах. Тя е полезна не само с факта, че определя точните метрики, които са необходими, но и причините за тяхната необходимост.

PDCA (plan-do-check-act) – итеративен четири стъпков модел, който се използва при Контрола на качеството

Six Sigma – група практики, използвани за подобряване на процеса, премахвайки дефектите в него

GQM-MEDEA (Goal-Question-Metric – MEasure DEfinition Approach) – метод за дефиниране на продуктови метрики, на базата на експериментални цели, поставени пред измерването.

Индекс на използваните фигури

Индекс на използваните таблици

Използвана литература

[1] - Beck K., (2002). *Test Driven Development: By Example*
[2] - Beizer B.,(1990). *Software Testing Techniques 2E*
[3] - Bereza-Jarociński B., (2004). *Non-Functional Software Testing in Practice*
[4] - Dustin E., (2003). *Effective Software Testing – 50 Specific Ways to Improve Your Testing*
[5] - Fewster M., Graham D., (1994).*Software Test Automation. Effective use of test execution tools*
[6] - Information Processing Ltd., (1997). *Software Testing and Software Development Lifecycles*
[7] - Marik B. *How Many Bugs Do Regression Tests Find?*
[8] - Menasce D., (2002). *Load Testing of Web Sites*
[9] - Norm Brown, (1998).*Little Book of Testing. Volume 1.Overview and Best Practices*
[10] - Park R., Goethert W., Florac W., (1996) *Goal driven software measurement*
[11] - Richard Lippincott, *Methods for Documentation Testing in Technical Publications Quality Assurance*
[12] – Sommerville, (2006). *Software Engineering*
[13] - (2004).*Mercury LoadRunnerTM Tutorial Version 8.0*
[14] - Задължителен курс Управление на качеството за магистърска програма Софтуерни технологии, лектор Илина Манова.
[15] - http://en.wikipedia.org/wiki/Remote_procedure_call
[16] - http://en.wikipedia.org/wiki/REST
[17] - http://en.wikipedia.org/wiki/Service-oriented_architecture
[18] - http://en.wikipedia.org/wiki/System_testing
[19] - http://en.wikipedia.org/wiki/Web_service
[20] - http://en.wikipedia.org/wiki/Windows_service
[21] - http://fox.wikis.com/wc.dll?Wiki~InstallationTesting~SoftwareEng
[22] - http://support.borland.com/entry.jspa?externalID=5489&categoryID=67
[23] - http://www.developer.com/net/csharp/article.php/2173801
[24] - http://www.locatemodelcities.org/library/ESIF33.pdf
[25] - http://www.mactech.com/articles/mactech/Vol.13/13.10/SoftwareTestAutomation/
[26] - http://www.mercury.com/us/products/performance-center/loadrunner
[27] - http://www.software-risk.co.uk/cont457.htm
[28] - http://www.softwaretesting.no/testing/volumetest.pdf
[29] - http://www.softwaretestinghelp.com/choosing-automation-tool-for-your-organization/
[30] - http://www.wrox.com/WileyCDA/Section/id-291048.html