СОФИЙСКИ УНИВЕРСИТЕТ

“СВ. КЛИМЕНТ ОХРИДСКИ”

Факултет по математика и информатика

Катедра: Информационни технологии

ДИПЛОМНА РАБОТА

на тема:

Внедряване на базирана на данни методология за автоматизирано функционално тестване на интернет приложения

Дипломант: Ива Любенова Кръстева

Специалност: Разпределени системи и мобилни технологии, фак. № 21298

Научен ръководител: доц. д-р Силвия Илиева

Консултант: Илина Манова

София, януари 2005 г.

Съдържание

1. Увод

1.1 Въведение в областта

В последните двайсет години се наблюдава мащабно развитие на софтуерните технологии и разпространението на софтуер. Закономерно на това търсенето и предлагането на софтуерни продукти се увеличава многократно. Пред съвременното софтуерното инженерство е поставено ново предизвикателство - не просто производство на софтуер, а производство на софтуер с високо качество. Акцентът в разработката на софтуерна система е реализиране на изискванията и очакванията на клиента за системата в максимална степен.

Осигуряването на качеството на софтуерния продукт е неизменна и важна част от процеса на разработка на софтуер. Тестовият процес проверява до каква степен са покрити изискванията към системата от различни гледни точки - функционалност, бързодействие, сигурност и други. Какво ще бъде участието на различните видове тестване в тестовия процес зависи както от типа на системата и областта на приложение така и от конкретната система. Най-често използвано в това отношение е функционалното тестване. То има за цел да провери дали разработения продукт реализира бизнес процесите на клиента и дали предоставя необходимата функционалност.

Автоматизираното тестване подпомага процеса на тестване и намира приложение за различни видове тестове. В действителност изпълнението на някои от видовете тестове не би било възможно без тяхното автоматизиране. Такива са например тестовете за измерване на производителността, бързодействието и максималното натоварване на системата. Продуктите и средствата за автоматизирано функционално тестване дават възможност за бързо и лесно многократно изпълнение на идентифицираните тестови сценарии, покриващи определени бизнес процеси. Най - разпространен и поддържан от почти всички инструменти за автоматизирано тестване е моделът на записване и изпълнение на скриптове. При него на първата стъпка автоматичното се генерира скрипт, който описва последователност от потребителски взаимодействия със системата. При изпълнението на скрипта се симулират тези взаимодействия и се проверява поведението на системата

Съвременните софтуерни процеси са итеративни и инкрементални в същността си. От една страна разработката на интернет системи предразполага към намаляване на продължителността на итерациите и времето за тестване. От друга страна проектът преминава през множество версии и промени на интерфейсите. Моделът на записване и изпълнение на скриптове е неефективен за прилагане в такава среда, защото адаптирането на скриптовете към новата версия е много трудоемък процес. Нова базирана на данни методология решава недостатъците на стария метод и предоставя допълнителни предимства за използването й. Методологията позволява да се осигури надежден тестов процес в среда на бързо променящи се изисквания и намалено време за тестване. Дипломната работа разглежда и предлага решение за използването на методологията, базирана на данни, (data - driven methodology) за автоматизирано функционално тестване на интернет базирано .Net приложение. При внедряването на методологията се използват:

- продуктът на фирмата IBM за автоматизирано функционално тестване - Rational Robot;
- тестова платформа, която надгражда инструмента и реализира методологията за тестване, базирано на данни - Rational Robot Automation Framework Support (RRAFS). Тестовата платформа е разработена като проект с отворен код на фирмата SourceForge;
- специално разработен инструмент, който осигурява допълнителни функции.

1.2 Цел и задачи

Целта на дипломната работа е да предложи решение за внедряване на методология, базирана на данни, за автоматизирано функционално тестване в итеративен процес на създаване на софтуер и в частност на създаване на интернет базирано .Net приложение.

Задачи, произтичащи от целта:

- Представяне и анализиране на предимствата на разглежданата базирана на данни методология;
- Разработване на инструмент, който ще използва Rational Robot и Rational Robot Automation Framework за съхранение и управление на скриптовете и сценариите;
- Описание на процеса на разработване на инструмента и неговите основни функции;
- Описание на тестовото приложение и идентифициране на основните бизнес процеси, и съответните тестови сценарии.

При разработването на дипломната работа са наложени следните ограничаващи условия:

- Инструментът да бъде разработен чрез .Net технология.
- За тестово приложение да бъде използвана конкретна интернет базирана .Net система
- За внедряване на методологията да се използват продукта на IBM Rational Robot и платформата на SourceForge Rational Robot Automation Framework Support (RRAFS)

1.3 Полза

Предлаганото решение за внедряване на методология, базирана на данни, е насочено към софтуерните инженери, които осигуряват качеството на софтуерния продукт. Използването на тестов процес, реализиращ разглежданата методология, при разработването на интернет базирани системи значително ще повиши качеството на приложението. Няколко са основните предпоставки, които определят ефективността на прилагане на методологията.

Първата предпоставка е лесната поддръжка на тестовете. При разработка на софтуер на кратки итерации и чести промени на интерфейсите на системата адаптирането на тестовете за новата версия трябва да се извършва бързо и лесно.

Втората предпоставка е възможността за използване на едни и същи тестове за автоматизирано и ръчно тестване. Това позволява тестовете, създадени за ръчно тестване да се използват и за автоматизиране, което води до скъсяване на времето за създаване на автоматизираните тестове. По този начин не се дублират усилията, които

са влагат при разработването на тестовете.

Третата предпоставка, която определя ефективността на новата методология, е използването на неквалифициран персонал при разработването на тестовете за автоматизирано тестване. Софтуерните инженери, които осигуряват качеството на продукта, не е необходимо да са запознати с езика на инструмента, който се използва за създаване на автоматизирани тестове. Тестовете се създават във формат, който е достъпен и разбираем от хората, и времето за обучение е значително по - малко.

Четвъртата предпоставка е покриването от тестове на голяма част от приложението. Намаляване на времето за разработване и изпълнение на тестовете позволява да бъдат разработвани повече тестове, които покриват по - голяма част от приложението.

1.4 Структура

Изложението на предложеното в дипломната работа решение е разгледано в следващите пет части.

Във втората част именувана “Използвани технологии” са описани технологиите, използвани за реализиране на решението. В тази част са разгледани предимствата и характеристиките на .Net технологията. Тази технология се използва за създаване на приложение, което подпомага процеса на внедряване на методологията, базирана на данни. Също така се използва .Net интернет базирана система за демонстриране на предлаганото решение. Rational продуктите, разработвани от фирмата IBM, също са обект на внимание в тази част. Акцентът на разглеждането се поставя върху продукта за автоматизирано тестване Rational Robot. Тестовата платформа Rational Robot Automation Framework Support надгражда инструмента Rational Robot и предоставя възможност за използване на методологията, базирана на данни, при автоматизирано функционално тестване на софтуерни системи.

Третата част е озаглавена “Обзор на проблемната област и теоретична обосновка на предлаганото решение”. В нея се представя необходимата теоретична основа за представяне на решението. Описва се тестовия процес и мястото му в цялостния процес на разработка на софтуер. Представени са най-известните методологии за автоматизирано функционално тестване. Накрая са изтъкнати предимствата на методологията, базирана на данни, и подробно е разгледан моделът на тестовата платформата, реализираща методологията.

Четвъртата част със заглавие ”Спецификация на решението за внедряване на базирана на данни методология” е посветена на обобщаване и систематизиране на принципите, към които се придържа процесът на внедряване.

Петата част е именувана “Реализация на инструмента за автоматизирано създаване на тестове”. В нея е разгледано проектирането и конструирането на инструмента, който реализира част принципите, към които се придържа процесът на внедряване.

Последната шеста част от изложението носи заглавието “Внедряване на методологията”. В нея се представя цялостния тестов процес, основан на базираната на данни методология за автоматизирано функционално тестване. Описва се прилагането на реализираното решение за тестването на един от модулите на интернет базирано .Net приложение.

Следващата седма част представя заключителната част на дипломната работа. В нея е направено обобщение на изложеното решение за внедряване на методология за автоматизирано функционално тестване на интернет приложения. Предложени са

насоки за усъвършенстване и развитие на решението.

Последните три части от дипломната работа описват речника на използваните абревиатури и термини, литературните източници и приложенията.

2. Използвани технологии

В тази част са разгледани използваните в дипломната работа технологии. Представена е технологията .Net, на която е реализирано десктоп приложение. Приложението подпомага процеса на внедряване на разглежданата методология за автоматизирано функционално тестване. Също така се разглежда интернет базирана .Net система като тестово приложение за демонстриране на решението, предложено в дипломната работа. При внедряването на методологията, базирана на данни, се прилага продукта Rational Robot като основа на платформата за автоматизирано тестване. Предложеното в дипломната работа решение използва тестовата платформа Rational Robot Automated Framework Support (RRAFS).

2.1 Net.

2.1.1 Характеристики

.Net обединява множество технологии, продукти и услуги за осигуряване на среда за създаване на решения, в която отделни устройства и услуги работят заедно. Основните характеристики, които определят широкото разпространение на технологията са:

- **Многоезикова разработка** - .Net позволява интегрирането на различни езици при създаване на приложения чрез използване на единен междинен език, наречен MSIL - Microsoft Intermediate Language. Всеки език, за който е дефиниран компилатор за преобразуване до междинния език, може да бъде използван. В момента Microsoft и други фирми предоставят компилатори за езиците C++, C#, Jscript, Visual Basic, COBOL, Eiffle, Fortran, Perl, Python, Scheme и други.
- **Независимост от платформи и процесори** - Компилирането на приложенията до междинния език, който е независим от платформи и процесори, осигурява преносимостта на приложенията между различните платформи. Единственото ограничение е наличието на среда за изпълнение на .Net приложенията (Common Language Runtime) за съответната платформа.
- **Управление на паметта** - .Net предоставя автоматизирано управление на паметта като представя решения за следене на използваната от обектите памет, нейното ефективно преразпределяне и навременно освобождаване.
- **Управление на версиите** - Единиците за внедряване на компонентите на приложението съдържат информация за версията си, която се използва при извикване на компонента. По този начин приложението ще използва единствено компонентите, с които е тествано и инсталирано.
- **Поддръжка на отворени стандарти** - обмена на съобщения в .Net среда е основано на отворените текстови стандарти за обмен на съобщения в интернет - XML - Extensible Markup Language, SOAP - Simple Object Access Protocol, UDDI - Universal Description, Discovery, and Integration, WSDL - Web Services Description Language.

- **Лесно инсталиране** - Поради използването на мета данни в .Net компонентите не се инсталират в системния регистър, което значително облекчава процеса на инсталиране на компонентите.
- **Разпределени приложения** - акцентът при създаването на .Net технологията е в осигуряването на цялостни решения за изграждането на разпределени интернет базирани приложения, които използват интернет услугите и протоколите за достъп, откриване и дефиниране на интернет услуги.
- **Интеграция с библиотечни файлове и COM / Component Object Model/ компоненти** - .Net предоставя възможност за използване на библиотечни файлове и COM компоненти от .Net компоненти. Също така COM компоненти могат да се обръщат към .Net компоненти.
- **Сигурност** - .Net осигурява на сигурност на различни нива - от клас и метод, до единиците за внедряване (assembly), машина и интернет зони. .Net поддържа работа с роли, алгоритми за криптиране и автентикация на потребители.
- **Производителност и мащабируемост (scalability) -** .Net предоставя подобрени решения за повишаване на производителността и мащабируемостта на приложенията, реализирани както в библиотечните класове, така и в сървъра за бази данни.

2.1.2 Инфраструктура

Основният компонент на .Net е .Net инфраструктурата, която се състои от .Net Платформата (.Net Framework), интегрираната среда за разработка Microsoft Visual Studio.NET и фамилията от сървъри .NET Enterprise Servers. Инфраструктурата обхваща всички технологии за изграждане и изпълнение на мощни и мащабируеми разпределени приложения.

Microsoft Visual Studio.NET е ”интегрирана и унифицирана визуална среда” [4] за разработване на приложения. Тя увеличава продуктивността на програмиста и опростява процеса на разработка като предоставя средства за допълване на кода, поддържа информация за елементите на класовете - IntelliSense и възможност за работа със съществуващи приложения. Средата организира всички компоненти на приложението като намалява времето за компилиране и инсталиране. Visual Studio.NET съдържа компилатори за C++, C#, Jscript, Visual Basic за многоезично осигуряване на разработваното решение. Освен това средата предоставя достъп до широката функционалност на библиотечни класове и е напълно интегрирана със сървъра за бази данни - MS SQL Server 2000. За програмиране на интернет базирани приложения средата поддържа работата с интернет услуги, опростява програмирането на бизнес логиката и разширява възможностите за представяне на информацията на потребителя.

Фамилията от продукти .NET Enterprise Servers обхваща единайсет сървъра за предоставяне на бърз и лесен достъп да данни и функционалност. Интегрирането на сървърите с платформата става чрез XML, който опростява манипулирането и достъпа до данни в разпределените .Net приложения.

.Net платформата е компонентът на инфраструктурата, който е най-тясно обвързан със създаването на приложения. В следващата част ще обърнем внимание на елементите, които изграждат платформата.

2.1.3 Елементи на .Net платформата

.Net Платформата има 6 основни елемента, предоставящи широк набор от функционалности за изграждане на приложения [4]. Тези елементи са средата за изпълнение (Common Language Runtime), библиотеките от класове (.NET Framework Classes), технологията ASP.NET, интернет услугите (Web services), .NET Remoting и формите за създаване на десктоп приложения (Windows Forms).

Среда за изпълнение

Средата за изпълнение е ядрото на платформата, чиято функция е изпълнението на приложенията. Тя е отговорна за интегрирането на различните програмни езици, сигурен достъп до кода и управление на паметта, процесите и нишките. Средата се грижи да бъде извикан и изпълнен правилния код като използва мета данните, записани във файловете. Средата за изпълнение поддържа многоезикова разработка и предоставя компилатор, който преобразува междинния език в машинен код. Този компилатор е оптимизиран за работа със съответната платформа, на която ши бъде изпълнено приложението. Пълната интеграцията на различните езици е възможна, защото компилаторите използват обща система от типове - Common Type System, която дефинира правилата за създаване и използване на нови типове. Всеки език за който е написан компилатор, отговарящ на дефинирани правила за преобразуване до междинния език, може да бъде използван. Тези правила са описани в спецификацията за интегриране на различни програмни езици - Common Language Specification. В .Net средата е възможно наследяване, извикване на методи и предаване на параметри през рамките на езика за програмиране. Средата за изпълнение също така предоставя и възможност за работа с компоненти и библиотеки, които не са създадени със средата за изпълнение на .Net.

Библиотеки от класове

.Net Платформата осигурява класове, интерфейси и типове, които оптимизират и улесняват разработването на приложенията. Тези библиотечни типове отговарят на правилата, дефинирани в общата система от типове, и могат да бъдат използвани в контекста на различни програмни езици. .Net Платформата предлага над 4500 класове, които предоставят функционалност за достъп до данни, входно - изходни операции, осигуряване на сигурност и създаване на разнообразни форми и потребителски интерфейси. Също така предоставя възможности за работа с XML, SMTP, транзакции, работа със съобщения, следене на производителността и много други.

ASP.NET

ASP.NET е “революционна програмна платформа” [4] за създаване на интернет базирани приложения. ASP.NET опростява създаването на страници, използването на интернет услуги и предлага нови функционалности. Технологията поддържа повече от 25 езика за програмиране. Производителността е подобрена чрез динамично компилиране на страниците в класове, които се управляват от средата за изпълнение. Microsoft е провел тестове, според които ASP.NET приложенията ускоряват изпълнението на заявки с два до три пъти спрямо класическите ASP приложения.

ASP.NET позволява съвместното използване на ASP.NET и ASP страници при изграждането на интернет приложение.

Интернет услуги

Интернет услугите са основното звено за изграждане на разпределени интернет базирани системи. Те са обединение от функции, които предоставят данни и услуги на други приложения. Интернет услугите се публикуват и са достъпни от приложенията чрез стандартни протоколи HTTP, XML и SOAP.

.NET Remoting

.NET Remoting е технология, която предоставя мощни и високопроизводителни средства за свързване на отдалечени обекти. Комуникацията между обектите се извършва посредством канали за обмен на съобщения. .NET Remoting поддържа HTTP и TCP протоколи за транспортиране на съобщенията. Съобщенията се форматират в подходящ формат за транспортиране. Технологията използва двоично и текстово форматира като за последното се използва протоколът SOAP. .NET Remoting е лесна за използване и осигурява прозрачно взаимодействие между отдалечени обекти.

Форми за създаване на десктоп приложения

Формите за създаване на десктоп приложения - Windows Forms надграждат традиционните форми в две направления. Изпълнението на формите от средата за изпълнение повишава сигурността. Другото основно предимство е, че използването на формите в интернет страници намалява времето за зареждане и визуализиране на страницата в сравнение с традиционните ActiveX контроли. Това се дължи на факта, че зареждането на контролите се извършва при необходимост, а не първоначално със зареждане на страницата. Този модел позволява добавянето на функционалност без да е необходимо приложението да се компилира наново.

2.1.4 C#

C# е един от езиците, който се поддържа от .Net и участва в изграждането на многоезични приложения. C# е сравнително нов обектно - ориентиран език. Основните предимства от използването му при създаването на .Net приложения са следните:

- Повишена производителност и надеждност - Ефективно управление на паметта от модул на средата. Езикът поддържа сигурност на типовете и автоматично инициализиране на променливите.
- Поддържа използване на версии
- Взаимодействие с други компоненти - C# позволява използването на COM компоненти и функции на операционната система при построяването на приложения
- Компонентите създадени със C# лесно могат да бъдат преобразувани в интернет услуги

2.2 RationalSoftware продукти за цялостния процес на разработка

RationalSoftware - едно от основните направления от софтуерни решения от фирмата IBM, е фамилия от софтуерни продукти, които намират приложение в процеса на разработка на софтуер. IBM определя [3] следните основни категории в зависимост от мястото им в софтуерния процес:

- Управление на процеса и портфолиото на проекта (Process and portfolio management)
- Анализ на изискванията (Requirements and analysis)
- Дизайн и разработка (Design and construction)
- Управление на конфигурациите (Software configuration management)
- Осигуряване и управление на качеството (Software quality)

2.2.1 Управление на процеса и портфолиото на проекта

Категорията обхваща продуктите за управления на софтуерния процес и портфейла на проекта от ръководителите на проекти и финансовите директори. Те предоставят възможност за планиране, разпределяне, наблюдение и оценка на ресурсите и финансовите средства на проекта през целия му жизнен цикъл. IBM предоставя гъвкава платформа за поддържане на софтуерния процес, изграден на основата на унифицирания процес за разработка на софтуер на Rational – Rational Unified Process (RUP). Унифицираният процес на разработка е инкрементален и итеративен в същността си процес, който използва най-добрите практики за осигуряване на успешен проект. Rational Suite обединява няколко продукта, които осигуряват цялостно решение за процеса на разработка - от събиране на изискванията до внедряването. В него са включени инструменти за анализ, разработка и тестване на продукта, както и интегрирана платформа за съвместна работа на екипа.

2.2.2 Анализ на изискванията

Продуктите в тази категория предоставят решения за определяне на проблемната област, за събиране и следене на промените в изискванията, за моделиране на взаимодействията и определяне архитектурата на базата данни. Моделите на процеси, архитектури и взаимодействия са основния начин за връзка с клиента и задоволяване на изискванията му в най - голяма степен. IMB Rational Rose е инструмент за моделиране, който използва езика за моделиране UML - Unified Modeling Language. Той предоставя възможности за моделиране на процеси, взаимодействия, състояния, архитектури и данни през всички фази на разработката на софтуерната система.

2.2.3 Дизайн и разработка

В тази категория са продуктите за моделиране на архитектурата и дизайна на системата, както и за нейното конструиране. В тази категория попадат продуктите, които поддържат модела на разработка, основана на модели /model driven development/ и модела на бързото създаване на приложения / rapid application development/.

2.2.4 Управление на конфигурациите

Продуктите в тази категория подпомагат паралелната разработка, достъпа до кода и определяне на привилегиите при конкурентно ползване. Те поддържат и управление на версиите на продукта, грешките и промените на софтуера. Продуктите могат да се използват с различни интегрирани среди за разработка (IDEs) като IBM Websphere Studio, Eclipse и Microsoft .NET.

2.2.5 Осигуряване и управление на качеството

Фирмата IMB предоставя продукти за осигуряване на качеството на софтуерната система за подпомагане както на процеса на тестване така и за процеса на разработка. Инструментите за автоматизирано тестване обхващат разнообразие от приложения - интернет базирани системи, вградени системи (embedded systems), системи за работа в реално време (real-time systems) и др., езици и платформи. Продуктите предоставят различен набор от видове тестове, които могат да бъдат изпълнени - за проверка на функционалността, на производителността или на ефективната работа на системата. Rational Test Manager ни помага в управлението на тестовия процеса - планиране, изпълнение и анализиране на тестовите активности. Инструментът позволява изпълнението на разнообразни тестове, предоставя различни графични рапорти и има пълна интеграция с други Rational продукти за определяне на изискванията, следене на грешките и версиите.

2.3 Rational Robot

2.3.1 Характеристики

Rational Robot е ”пълен комплект от компоненти за автоматизиране на тестването на Windows клиент-сървър и интернет базирани приложения” [6]. Инструментът осигурява поддръжка за:

- операционни системи - Windows 2000, Windows 98, Windows Me, Windows NT, Windows XP.
- програмни езици и технологии - .Net, Visual Basic, Oracle Forms, Web-based Oracle Forms, Delphi, Java и други. /За пълния списък поддържани технологии - http://www-306.ibm.com/software/awdtools/tester/robot/sysreq/index.html /
- браузъри - Microsoft Internet Explorer 4.x, 5.x и 6.0 и Netscape
- Интернет поддръжка - работи със статични и динамични интернет страници, генерирани от HTTP сървъри

2.3.2 Създаване на функционални тестове с Rational Robot

Rational Robot е инструмент за автоматизирано записване и изпълнение на

тестове. Инструментът предоставя възможност за изпълнение на функционални тестове и тестове за производителност. Функционалните тестове се създават за проверка на реализираните от приложението бизнес процеси. Тестовете за производителност се използват за да се проследи поведението на системата при различно натоварване с потребители, времето за отговор и за изпълнение на основните операции. Ще се спрем по-подробно на видовете тестване и в следващата част. За изпълнението на функционалните тестове Rational Robot работи с автоматично записани или ръчно програмирани скриптове на езика за програмиране на инструмента - SQA Basic. Тестовете за производителност се създават на друг скриптов език – VU Language. Обект на разглеждане в настоящата дипломна работа са възможностите, които предоставя Rational Robot за автоматизирано функционално тестване.

2.3.3 SQA Basic

SQA Basic е програмен език за създаване на скриптове за автоматизирани функционални тестове в Rational Robot. SQA Basic е подобен на Visual Basic като добавя множество от команди, които са специфични за работата с тестове. SQA Basic не поддържа работата с обекти за създаване на различни елементи на потребителския интерфейс. В програмния език на Rational Robot са добавени команди за работа с обектите на приложението и техните свойства, за работа с координати, за моделиране на взаимодействията на потребителя с приложението и други.

2.3.4 Технология за записване Object - Oriented Recording

В процеса на записване на скриптовете за функционално тестване инструментът предоставя възможност за откриване на обектите, използвани в приложението. Това се извършва благодарение на специална технология за работа с обекти - Object - Oriented Recording. Rational Robot използва технологията за да идентифицира обектите чрез техните вътрешни имена, а не чрез координатите на екрана. Инструментът генерира идентификационен низове за всеки обект на приложението. Идентификационните низове на Rational Robot съдържат йерархията от компоненти до този компонент, типа на компонента и някой от атрибутите на компонента, осигуряващ му уникалност. Реда на използване на атрибутите на компонента може да бъде конфигуриран от потребителя. Първото предимство, които произтича от използването на технологията за записване е, че обектите могат да променят местоположението си на екрана без това да попречи на изпълнението на теста. Второто предимство е, че могат да бъдат тествани обекти, които не са видими на потребителския интерфейс.

2.3.5 Технология за тестване на състоянието на обекти Object Testing

На различни места в тестовете може да бъде направена проверка на състоянието на обектите на приложението. Технологията Object Testing дава възможност да бъдат тествани обектите на приложението както и техните свойства и данни. При записването на скрипта се задава очакваната стойност на дадено свойство или данни. При изпълнението на тестовете състоянието на обектите се сравнява с тази стойност. Инструментът използва лог файл, в който записва резултатите от изпълнението на тестовете. Лог файлът се използва за анализиране на резултатите от теста. На възможностите за анализ на резултатите ще се спрем при разглеждането на компонента

за управление на тестовете - Rational TestManager.

2.3.6 Пулове от данни (datapools)

Пуловете от данни са източници на тестови данни, които скриптовете използват по време на изпълнение на тестовете. Данните, които се предават на скриптовете, могат да бъдат различни и дори уникални при всяко изпълнение на тестовете. Пуловете от данни се могат да бъдат дефинирани със случайно генерирани стойности. На данните могат да бъдат зададени често използвани предефинирани типове като градове, телефони, имена или да използват потребителски типове. Пуловете от данни се създават чрез инструмента Rational TestManager и се използват в скриптовете, създадени с Rational Robot.

2.3.7 Поддръжка на .Net и HTML

Rational Robot осигурява изчерпателното тестване на .Net приложения, създадени с Visual Basic, C++ или C#. Инструментът разпознава обектите на формите, използва имената им, позволява проследяване на свойствата на обекти и техните данни. За целите на дипломната ще се спрем по-подробно на поддръжката, която предоставя Rational Robot за автоматизираното тестване на интернет форми създадени с ASP .Net. Инструментът подпомага тестването на интернет приложения, включително и тези, създадени с ASP .Net. Rational Robot използва разширението за тестване на HTML страници за генерираните от ASP .Net страници. Дефинирани са типове обекти, на които се съпоставят HTML елементи. Например на типа CheckBox съпоставя HTML елементите, които имат вида <INPUT type=Checkbox>, а на типа EditBox - HTML елементите, които имат вида <INPUT type=Text> или <INPUT type=TextArea>. Пълния списък от асоциации е представен в Приложение 1. За различните видове типове инструментът поддържа възможности за тестването на данните и атрибутите на обектите. Това дава възможност да се изследва текстът на елементите както и всички атрибути на елементите, включително и тези, които нямат визуално представяне и не могат да бъдат тествани ръчно. При генериране на идентификационните низове за HTML елементите Rational Robot винаги използва следната последователност от атрибути в идентификационния низ -атрибута htmlid, атрибута name, атрибута text. Ако тази поредица от атрибути не гарантира уникалност на низа накрая слага индекс. Rational Robot предоставя възможности за синхронизиране на страниците и изчакване на страниците да се заредят напълно преди да бъдат изследвани техните обекти.

2.3.8 Интеграция с други продукти

За подпомагане на процеса на автоматизирано тестване Rational Robot е интегриран със следните Rational продукти:

- **Rational TestManager** - Инструмент за управление на тестовите активности - планиране, дизайн на тестовете, създаване на тестове за ръчна и автоматизирана обработка, изпълнение на тестовете и анализ на резултатите чрез лог файлове и различни документи за анализиране на изпълнението на тестовете.. TestManager се използва и за изпълнение на тестове за производителност. Сценариите, които ще се тестват, се създават чрез Rational Robot. Rational TestManager дава възможност да се определи натоварването на системата с виртуални

потребители, което да се симулира при изпълнението на тестовете. Инструментът предоставя възможност за преглед на резултатите от изпълнението на теста и състоянието на обектите на зададените места. Той дава информация как е преминало изпълнението на теста като визуализира лог файла. Също така сравнява актуалното и очакваното състояние на даден обект. Сравняването може да се извърши на четири нива - на атрибути, на данни, на картинки и на таблици.

- **Rational TestFactory** - Автоматично създава тестови скриптове за цялото приложение на базата на готови скриптове и структурата на приложението. Дава информация в каква степен е покрито приложението от тестовете.
- **Rational ClearQuest** - Инструмент за управление на промените на спецификацията и грешките по време на цялата фаза на разработка на приложението. Тази интеграция дава възможност грешките да се отразяват директно, да се следят промените и да се анализира прогреса на проекта.
- **Rational Purify** - Инструмент за осигуряване на надеждност на кода - проверява управлението на паметта и грешки по време на изпълнението.
- **Rational Quantify** - Инструмент за анализиране на производителността на системата.
- **Rational PureCoverage** - Инструмент за анализиране на степента на покритие на кода от извършените тестове.
- **Rational RequisitePro** - Инструмент за управление на изискванията към приложението през целия процес на разработване на приложението. Степента на задоволяване на изискванията на клиента е в пряка зависимост с качеството на продукта.

2.4 Платформа за автоматизирано тестване RRAFS

SAFS - Software Automation Framework Support е проект с отворен код на SourceForge за реализиране на платформи за автоматизирано функционално тестване като се използва методологията, базирана на данни. Идеята на проекта е да бъдат създадени съвместими имплементации за различни инструменти и платформи и „в близко бъдеще да бъде създаден един многоплатформен набор от програми” [5]. По този начин автоматизираните тестове ще бъдат независими от тестовата платформа и инструмента, на който са създадени, и ще бъдат преносими на различни платформи и инструменти. До момента са създадени имплементации за два от най-големите инструмента за автоматизирано функционално тестване - Rational Robot на фирмата IBM и продукта WinRunner на фирмата Mercury Interactive. Има и реализация за инструмента от фирмата IBM за тестване на интернет и Java приложения - Rational XDE Tester.

Разработката на тестовата платформата за автоматизирано тестване RRAFS (Rational Robot Automation Framework Support) стартира през 1999 година. Последната версия е публикувана на 16 декември 2004 година. RRAFS осигурява поддръжка за:

- Платформи - Windows
- Типове приложения - интернет, .Net, Oracle, PeopleSoft
- Програмни езици - Java, VB, C, C++, C# , Delphi

Използването на RRAFS като платформата за автоматизирано функционално тестване има множество предимства, които произхождат от същността на методологията, базирана на данни. Тях ще разгледаме при представянето на

методологията в следващата част.

Други инструменти, които реализират методологията, базирана на данни, са:

- **EMOS Framework** - EMOS Framework [14] е платформа с отворен код за инструмента за автоматизирано тестване на фирмата Mercury Interactive - WinRunner. Платформата е създадена от група програмисти от фирмата EMOS Computer Consulting GmbH. Основната цел при създаването на платформата е определена по следния начин - „Създаване на колкото се може повече тестове с цената на по - малко усилия" [14]. Според авторите на платформата целта може да бъде постигната като се даде възможност автоматизираните тестове да се създават от хора, които не са програмисти.

- **FIT/FitNesse** - FIT/FitNesse [15] е проект с отворен код за управление на автоматизирани тестове, който се състои от три компонента - олекотена платформа за описание на тестовете, сървър и база данни. Платформата подпомага създаването, поддръжката, изпълнението и представянето на резултатите от тестването на системата от потребителя при доставянето на системата (Acceptance testing).

- **Certify** – Certify[16] е инструмент, разработен от фирмата Worksoft, който предоставя възможност за организиране, планиране, дефиниране, изпълнение и анализиране на автоматизирани тестовете. Инструментът предоставя пълно решение за автоматизирано тестване като осигурява поддръжка на множество технологии и платформи. Фирмата производител обещава намаляване на времето и усилията за автоматизиране на тестовете с шейсет процента [16].

- **Unified TestPro** - Unified TestPro [17] е множество инструменти за автоматизирано тестване, разработени от фирмата SDT - Software Development Technologies. Фамилията продукти предоставя решения за цялостния процес на автоматизирано тестване в най-различни технологични направления като телекомуникации и вградени системи.

- **TestFrame** – TestFrame [18] предоставя цялостно решение за внедряване и използване на методология за автоматизирано тестване на фирмата LogicaCMG. Фирмата разчита на опита и експертните знания в различни бизнес области за идентифициране на най - добрите практики за тестването като цяло и в частност автоматизираното тестване.

- **TestArchitect** – TestArchitect [19] предоставя завършено решение, реализирано от фирмата LogiGear. Платформата предоставя и множество допълнителни функции като поддръжка на разпределени екипи, контрол на версиите, интеграция с инструменти за автоматизирано тестване както и възможности за анализ на резултатите от тестовете.

3. Обзор на проблемната област и теоретична обосновка на предлаганото решение

3.1 Процес на разработка на софтуер

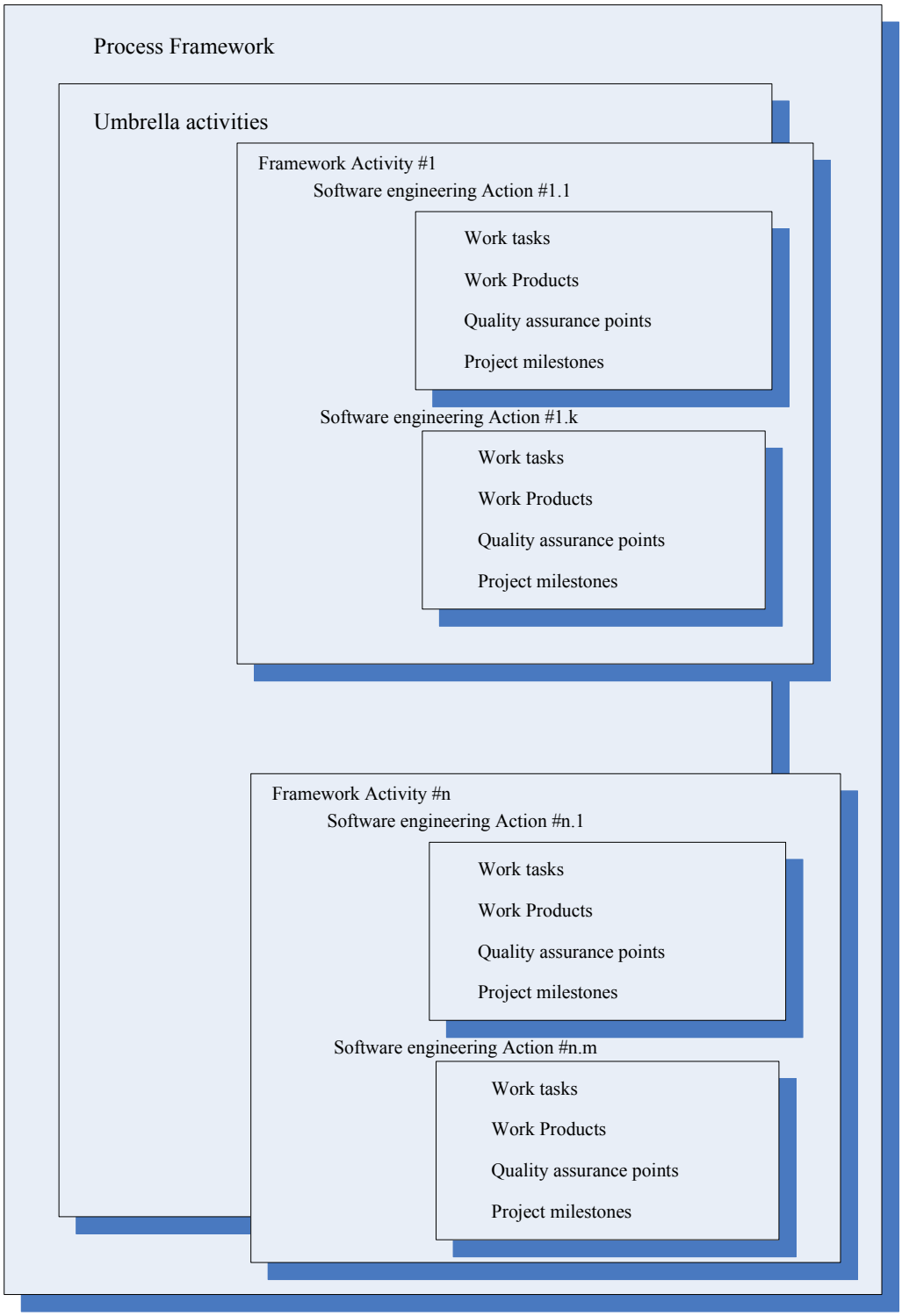
Софтуерният процес дефинира последователност от дейности, водещи до създаване на софтуерно решение или до развитие на съществуващо такова. Използването на предварително дефиниран процес осигурява стабилност, контрол и организация на дейностите. По този начин се гарантира разработването на качествен продукт в определените времеви рамки. За да бъде ефективен и приложим един софтуерен процес подходите за дефинирането му трябва да са гъвкави и съобразени със средата, в която ще се прилага и с типа на разработваното софтуерно решение.

В книгата си Software Engineering - A Practitioner'a Approach [2] Робърт Пресмън дефинира софтуерния процес като рамка (framework) за всички дейности, които са необходими за създаването на софтуер с високо качество. Тази рамка е основа на всеки софтуерен процес. Тя идентифицира няколко активности, които са приложими за всеки софтуерен проект независимо от неговата големина и сложност. Също така тази рамка включва и множество от глобални активности, които намират своето приложение през целия процес. На Фигура 1 е представена схема на софтуерния процес.

От Фигура 1 се вижда, че всяка ***активност*** е съставена от множество ***действия*** (software engineering actions), всяко от които е изградено от различни ***дейности*** (tasks). ***Дейностите*** извършват част от работата, която трябва да бъде изпълнена, в рамките на едно действие. В резултат на изпълнението на дадено ***действие*** се създава определена основна част от софтуерния продукт. Като казваме 'част от софтуерния продукт' изхождаме от представената в книгата [2] дефиниция на софтуерен продукт като обединение на програма/и, данни и документация. Компютърната програма задава инструкциите, чието изпълнение осигурява желаната функционалност. Структурите от данни, използвани от програмата, позволяват правилната обработка и представяне на информацията. Документите, съпровождащи процеса на разработка на продукта, предоставят описание на системата и начин на употреба. В рамките на едно действие, освен дейностите, които трябва да бъдат извършени, се определят и свързаните с тяхната работа продукти, дейностите за осигуряване на качеството и времевите интервали, определени в плана на проекта.

Основните ***активности***, които са определени и които са приложими към мнозинството софтуерни проекти са:

- **Комуникация** - Тази активност определя дейностите свързани с идентифициране на изискванията на клиента и изисква постоянна връзка и сътрудничество с клиента.
- **Планиране** - Планирането е основна активност, която е свързана с планиране на дейностите в активностите, които предстоят. Освен това се описват продуктите, които ще бъдат изготвени в процеса на работа, необходимите ресурси и вероятните рискове.
- **Моделиране** - Тази активност е свързана с изработване на различни модели на системата, които позволяват по-пълно разбиране на изискванията на системата, както от клиента така и от програмиста.



Фигура 1. Софтуерен процес

- **Конструиране** - Конструирането е активност, която включва в себе си създаването на кода за програмата и тестването на програмата.
- **Внедряване** - Тази активност включва доставянето на системата на клиента и неговата оценка за разработения продукт.

Тези пет основни активности са приложими в различни по тематика и сложност проекти. Конкретното реализиране на действията е различно в зависимост от характеристиките на проекта.

Рамката на софтуерния процес дефинира и множество от ***глобални активности***, които намират своето приложение в целия процес. Някои такива активности са:

- **Контрол на проекта** - Тази активност позволява да се следи прогреса на проекта и времето за изпълнението на всяка дейност;
- **Управление на риска** (Risk Management) - Управлението на риска е свързано с превантивни мерки за намаляване на рисковете на проекта;
- **Осигуряване на качеството** (Quality Assurance) - Тази активност задава дейности за гарантиране на качеството на софтуерния продукт;
- **Оценка** (Measurement) - Тази активност може да бъде използвана заедно с други активности. Тя задава метриките, които се използват за оценка на процеса, проекта и продукта;
- **Управление на конфигурациите –** Управлението на конфигурациите е активност, която проследява промените и версиите на всички части на софтуерния продукт.

Моделите на процеси представят различни реализации на рамката на софтуерния процес. Прилагането на един софтуерен модел трябва да е съобразено с проблема, проекта, екипа и организационната структура. Моделите се различават в степента на дефиниране и начина на прилагане на отделните компоненти и на взаимодействията между тях, както и определяне на ролите и организацията на участниците в процеса. Два основни типа модели [2] са намерили приложение в софтуерното инженерство – описателните (prescriptive) и гъвкавите (agile) модели на процеси.

Появата на описателните модели датира от преди трийсет години. Те наблягат на изчерпателност при дефинирането, и идентифицирането на активностите. Всеки описателен модел определя и начина на взаимодействие и последователността на прилагане на отделните елементи на софтуерния процес. Описателните модели могат да бъдат разделени в няколко групи - линейни, инкрементални, еволюционни и специализирани модели. Линейните модели представят последователното преминаване на процеса през активностите, определени от софтуерната рамка. Най-известния представител на линейните модели е Моделът на Водопада (Waterfall Model). Инкременталните модели задават създаване на софтуер на итерации като по този начин се намалява времето за разработка и се осъществява по-ранна връзка с клиента. При големи и сложни системи изискванията към системата и нейната функционалност не могат да бъдат определени изцяло в началото на процеса на разработка на софтуер. Еволюционните модели са итеративни модели, като на всяка следваща итерация системата се надгражда с нова функционалност. Представител на еволюционните модели е Моделът на Спиралата (Spiral Model), който представя цикличното преминаване през основните активности като постепенно се разширява обхвата им и се намалява риска на проекта. Специализираните модели са тясно свързани с конкретната област на прилагане. Например такива модели се прилагат при разработването на

софтуерни компоненти и на системи с висока сигурност.

Много по-късно - в началото на нашия век, се появяват гъвкавите модели на процеси, които наблягат на възможността за промяна и адаптация на процеса. Гъвкавите процеси предлагат алтернатива на строго формалните описателни модели. През 2001 година е публикуван Манифеста за Гъвкаво Разработване на Софтуер (Manifesto for Agile Software Development). В него [7] са дефинирани четерите важни точки, на които се основава новата методология:

- **Човешката индивидуалност** и взаимоотношения пред дефинирания процес и инструменти
- **Работещият продукт** пред изчерпателната документация
- **Съвместна работа с клиента** пред договори
- **Реагиране на промените** пред стриктното спазване на плана

Манифестът се основава на дванайсет основни принципа, към които се придържат гъвкавите модели. До момента са познати над десет гъвкави модела на процеси като най - разпространения от тях е Екстремното Програмиране (Extreme Programing).

3.2 Тестов процес

Осигуряване на качеството е основна активност във всеки софтуерен процес, която се прилага през целия процес на създаване на софтуер. Тя определя дейности за предотвратяване и коригиране на дефекти в процеса на създаване на софтуер. Тестването дефинира множеството от дейности, гарантиращи качеството на разработвания продукт. Качество на софтуерния продукт може да бъде определено като оценка на продукт, по отношение изпълнението на потребителските очаквания за него, базирани на предварително зададени изисквания. Формалната дефиниция на тестовия процес е следната: [Cem Kaner, Jack Falk, Hung Quoc Nguyen, 1999, Testing Computer Software, John Wiiley & Sons,Inc] *Процес на изследване на софтуерната система, за да се докаже, че тя удовлетворява изискванията, за които е предназначена и че покрива критериите за качество към нея.* Най-важното условие за една система е да удовлетвори очакванията на своите клиенти и потребители.

Тестовият процес е неразделна част от процеса на създаване на софтуерен продукт. Определянето на тестовите дейности, какъв да бъде техния обхват, кога да бъдат извършени и как е в основата на тестовия процес. Тестването се прилага на няколко нива, определящи тестовите цикли (test cycles):

- **Тестване на програмни единици** (Unit testing) - клас, процедура. Този вид тестване обикновено се извършва от програмиста.
- **Тестване на програмни модули** (Module testing) - прилага се при големи и сложни продукти, съставени от няколко модула
- **Интеграционно тестване** (Integration testing) - тестване на системата след интеграцията на различни модули
- **Системно тестване** (System Testing) - тестване на цялата система. На това ниво се извършват тестове на функционалността, на сигурността и на производителността на системата.
- **Тестване при доставяне на системата** (Acceptance testing) - На това ниво на тестване се проверява как системата се държи в реална среда. Тестовете се извършват от потребителите на системата.

Тестването на функционалността на системата проверява дали системата

коректно реализирани бизнес процесите на клиента или потребителя. Тестовите сценарии проверяват поведението на системата за всеки бизнес процес. Този вид тестване не се интересува от начина на конкретно реализиране. Приложението се разглежда като черна кутия (black box), която при зададени входни данни трябва да генерира определен поток от изходни данни. Тестовите сценарии се дефинират за множество от входни данни и очакван резултат, който трябва да бъде постигнат след изпълнението на програмата. Добра практика е функционалните тестови сценарии да се извършват както за множество от валидни и така и за множество от невалидни входни данни. Тестовете, които се извършват с валидни входни данни се наричат положителни тестове. Негативните тестове проверяват поведението на системата при въвеждане на невалидни входни данни.

В процеса на итеративно създаване на софтуер значително се увеличава нуждата от изпълнението на тестове многократно. След всяка итерация трябва да се провери, че приложението е променено успешно. Също така е необходимо да се види дали промяната не е довела до непредвидени грешки в приложението. **Регресионното тестване** се използва да се провери, че приложението покрива тестовите сценарии, които не са засегнати от промяната. **Автоматизираното тестване** е приложимо за тестове, които се изпълняват многократно. Това е така, защото създаването на тестовете за автоматизирана обработка е по-дълъг и трудоемък процес. Веднъж създадени всяко следващо изпълнение увеличава ефективността на автоматизирания процес на тестване. Друго предимство, е че автоматизираното тестване дава възможност за често тестване, покриващо по-голяма част от приложението.

Инструментите за автоматизирано тестване дават различни възможности за автоматизиране на различни видове тестове. В дипломната работа ще разгледаме автоматизирането на функционалните тестове.

Основните компоненти, които участват в създаването на функционален тест за автоматизирано изпълнение, най - често са следните:

- **Тестов сценарий (Test case)** - множество от входни данни, предпоставки и очаквани резултати, които да верифицират определено изискване.
- **Тестова сюита (Test suite)** - подредено множество от тестови сценарии. Тестовите сюити въвеждат йерархична структура на създаване на тестове.
- **Скрипт (Test Script)** - програма, написана на езика на инструмента за автоматизирано тестване, която автоматизира изпълнението на тестов сценарий или сюита
- **Тестови данни (Input data)** - входни данни за тестовия сценарии.
- **Очаквани резултати (Expected Results)** - състояние на приложението в даден момент от изпълнението на тестовия сценарии.

Автоматизираното изпълнение на тестове най - общо има 3 функции:

- Стартиране на тестовия скрипт, описващ какви действия ще изпълни тестовото приложението;
- Сравняване на резултатите от изпълнението на действията с очакваните;
- Запазване на резултатите от изпълнението.

Нека да разгледаме съществуващите методологии за автоматизирано функционално тестване.

3.3 Методологии за автоматизирано тестване

3.3.1 Принципи на внедряване

Платформа за автоматизирано тестване се дефинира като множество от предположения, концепции и практики, които осигуряват поддръжка за автоматизирано тестване.

При внедряване на методология за автоматизирано тестване трябва да се имат предвид следните принципи:

- **Тестовата платформа е независима от приложението** - платформата трябва да може да работи с компоненти, които са общи за всички приложения като по този начин ще позволи използването и при различни приложения. Потребителския интерфейс на едно приложение е изграден от определен набор от компоненти, за които могат да бъдат дефинирани определени функции. Тези компоненти и функции са общи за различните приложения;
- **Тестовата платформа е разширяема и лесна за поддръжка;**
- **Тестовата платформа не трябва да оказва влияние на дизайна на тестовете** - речникът, който се използва при създаването на тестовете не трябва да зависи от тестовата платформа. Това позволява едни и същ речник да бъде използван при различни тестови платформи и приложения. Не трябва да оказва влияние както на начина, по който специфицираме взаимодействието на потребителя с приложението през неговия интерфейс така и на цялостния дизайн на създаване на тестови сценарии;
- **Едни и същи тестови скриптове за ръчно и автоматизирано тестване** - При наличие на преносимост на тестовете между платформите следващата стъпка е да се осигури възможност на тестовете, съставени ръчно, да бъдат използвани за автоматизиране и обратното. Използване на тестовете от ръчното тестване за автоматизирано е стъпка за намаляване на времето и усилията за тестване, защото в итеративен процес на създаване на софтуер почти всички сценарии се използват по няколко пъти. Автоматизирането на тестовете има смисъл при многократно изпълнени на тестовите сценарии;
- **Тестовата платформа трябва да осигурява допълнително ниво на абстрактност за работа с тестовата среда** – Тъй като речникът, който се използва, е независим от платформата и тестовете, които се създават ръчно се използват за автоматизирано тестване не е необходимо тестерите, които се занимават със създаване на тестови сценарии, да са запознати с платформата в детайли.

3.3.2 Записване и Изпълнение на скриптове

Методологията Записване и Изпълнение на скриптове (Record/Playback) се основава на автоматизираното генериране на изпълним скрипт, който описва последователност от действия на потребителя на тестваното приложение. При изпълнение на създадените скриптове се пресъздават записаните действия.

Характеристиките на този метод, които го определят като най - неефективен са следните:

- Скъпи инструменти;
- Кодиране на данните в скриптовете - скриптовете съдържат в себе си кодирани входните и изходните данни, разпознаването на елементите на интерфейсите, навигацията и бизнес логиката;
- Трудна поддръжка - всяка промяна на някой от горните елементи води до презаписване или ръчна промяна на кода на скриптовете;
- Труден процес на записване:
    - Ако при изпълнение на скриптовете се появи съобщение, което не е било предвидено при записването, теста трябва да се повтори
    - Ако се направи грешка докато се записва теста, то той трябва да се презапише
    - Ако има грешки в приложението, то теста не може да се запише докато те не се оправи. Така процеса на откриване на грешки се извършва докато се записва теста, а не при изпълнението му.

Всички тези причини водят до изпълнение на записаните скриптове не повече от няколко пъти и отделяне на много време за поддръжката им. Поради това тази методология не може да бъде използвана в дългосрочен план.

3.3.3 Параметризиране на входни и изходни данни

При прилагането на методологията Параметризиране на входни и изходни данни (Data- driven Scripts [9]) скриптовете се записват или реализират на езика на продукта за автоматизирано тестване, който се използва. След това те се модифицират да могат да работят с променливи входни и изходни данни. Данните могат да се съхраняват във файлове или бази данни, от където се извличат и се ползват в тестовете. Тази методология се характеризира с:

- Намаляване броя на скриптовете, необходими за автоматизирано тестване - използването на едни и същи скриптове за тестването на ситуации с различни входни и изходни данни
- Запазва останалите недостатъци, коментирани при разглеждането на предишната методология

3.3.4 Функционална декомпозиция

При метода на Функционалната декомпозиция (Functional Decomposition) се използва йерархична структура и модулен дизайн. Всеки тест се разделя на части, които се реализират в различни скриптове. Тези части се определят в зависимост от това дали са част от навигацията, дали реализират определена бизнес функция или участват при верификацията на данните. Така всеки тест се реализира като последователност от скриптове, които могат да участват в други тестови сценарии. Например верификация на състоянието на банкова сметка на потребителя може да се прави и при теглене и при внасяне на пари в сметката му. Автентикацията на потребител и регистрирането на потребител в системата може да се използва на няколко места в един тестов сценарии или да се използва в няколко като не е необходимо всеки път да се прави верификация на потребителя. Предимствата на

разглеждания метод са:

- Намаляване на времето и усилията за създаване на автоматизирани тестове чрез повторно използване на скриптове
- Ефективна поддръжка на скриптовете - при промяна се променя само необходимия скрипт.

3.3.5 Методология, базирана на данни

Методологията, базирана на данни, се характеризира с параметризиране на входните данни, изходните данни, и компонентите на интерфейсите. Методологията запазва предимствата на представените в точки 3.3.2, 3.3.3 и 3.3.4 методи и елиминира недостатъците. Освен това е единствената от тези методологии, която отговаря на разгледаните в точка 3.3.1 принципи за успешна тестова стратегия.

Тестовете се разработват под формата на таблици, в които се описват стъпките за изпълнението на сценария. Действията на потребителя при взаимодействие с приложението и необходимите проверки се описват в таблиците чрез ключови думи. В таблицата се описват и компонента на интерфейса, чрез който се извършва взаимодействието или който ще бъде верифициран, и стойностите, които се предават.

Тестовата платформа интерпретира по независим от приложението начин тестовете. Освен това при наличие на стандарт, който да определя речника от ключови думи и структурата на таблиците, тестовете биха били преносими между различните имплементации на тестови платформи, поддържащи методологията. Ограниченията на дизайнерите на тестове е да използват речника на ключовите думи и структурата на тестовите таблици. При наличие на стандарт те ще имат един шаблон за описване на тестовите сценарии. Създадените по този шаблон тестови таблици могат да се използват за автоматизирано тестване между различните имплементации на тестовата платформа и напълно независимо от инструмента за автоматизирано тестване.

Тестовата платформа осигурява функции за работа с общи за приложенията компоненти. Тези функции приемат параметри, зададени в контекста на определено приложение. Те могат да бъдат използвани с всяко приложение, което сме избрали да тестваме. При интерпретирането на тестовите таблици платформата избира необходимата функция за зададения компонент, изпълняващ действието, което трябва да бъде предприето.

Таблица 1. Примерен ред от тестова таблица

Прозорец	**Компонент**	**Действие/ Ключова дума**	**Параметри**
Страница за автентикация	Потребителско име	Верифициране-на-стойността	“Име”

Разчитането на този реда от Таблица 1 е следното: На страницата за автентикация провери, че полето Потребителско име има стойност Име.

След като се запознахме накратко с методологията нека да разгледаме **предимствата на методологията, базирана на данни**:

- Предлага предимствата на автоматизираното тестване за повишаване на качеството на кода и намаляване на времето за разработка;
- Таблиците са лесни за създаване и поддръжка, защото са лесни за интерпретиране от човека;

- Тестовете могат да бъдат създадени преди да е завършен продукта, достатъчно е да са изяснени изискванията и дизайна на приложението. Единственото, което е необходимо да се знае, е какъв ще е интерфейса и да е изяснена спецификацията на бизнес процесите. По този начин тестовете могат да започнат да бъдат изпълнявани веднага след завършване на процеса на имплементиране на определен модул или цялата система. Това намалява значително времето за завършване на процеса и предоставя възможност да бъдат реализирани повече тестови сценарии;
- Не е необходимо обучение за използване на платформата за автоматизирано тестване на тестерите. За изпълнението на тестовете е необходим един човек от екипа тестери, който е запознат с инструмента за автоматизирано тестване;
- Възможността за повторно използване на тестовете се осигурява от допълнително ниво на абстрактност между компонентите на приложението в таблиците и идентифицирането им в конкретния интерфейс както и от принципа на модулност и йерархичност при създаване на сценариите. На това предимство ще се спрем при по - подробното разглеждане на модела на платформата.

В различните литературни източници [8, 9, 10, 11] се разглеждат методологии, които са обединение на някои от посочените - например функционалната декомпозиция и параметризиране на входни и изходни данни. Също така се използва понятието за базирани на данни като обединение на няколко от методите или за именуване на методологията, базирана на параметризиране на входни и изходни данни. Разбира се рязко разделение не може да бъде направено, защото всяка от тях се възползва от предимствата на другата и я включва в себе си. Така например метода на функционалната декомпозиция не би имал смисъл, ако нямаме възможност да предаваме различни данни на обособените бизнес скриптовете. Последната методология обхваща предишните две, тъй като записаните в таблиците входни и изходни стойности предоставят лесен начин за промяна, защото не са кодирани в скриптове, а таблиците са организирани в йерархична структура и се предоставя възможност за повторно използване. В разглежданията на дипломната работа, когато казваме базирана на данни, ще я отъждествяваме с третия метод. Също така като еквивалентни определяния ще използваме “методология, определена от ключови думи” или “методология, определена от команди”. В литературата се използват следните синоними:

- Data - driven - [6, 8, 12]
- Keyword-driven – [8, 11, 12]
- Table Driven - [9, 11]
- Test Plan Driven Testing Framework – [8]

3.4 Модел на тестовата платформа

В дипломната работа ще разгледаме внедряването на базирана на данни методология, като използваме имплементацията на тестова платформа RRAFS. Нека да разгледаме основните принципи, които са спазвани при имплементирането на платформата, представени [9] от един от инициаторите и създателите на платформата Карл Нейгъл (Carl Nagle):

- Възможност за създаване и осъществяване на логични интуитивни тестове едновременно ръчно и автоматизирано;
- В един ред от входните таблици ще се съдържат действието, което трябва да се извърши, входните и изходните данни;
- Възможност за извикване и изпълнение на скриптове създадени на езика на автоматизирания инструмент. Използването на такива скриптове е подходящо в случаи когато логиката на теста се описва трудно в таблици и когато няма да има нужда от допълнителна поддръжка при промяна на приложението;
- Платформата ще е напълно независима от приложенията като ще позволява тестването на различни приложения след внедряването;
- Платформата ще бъде добре документирана и оповестена;
- Платформата ще бъде свободно достъпна за ползване и разработване.

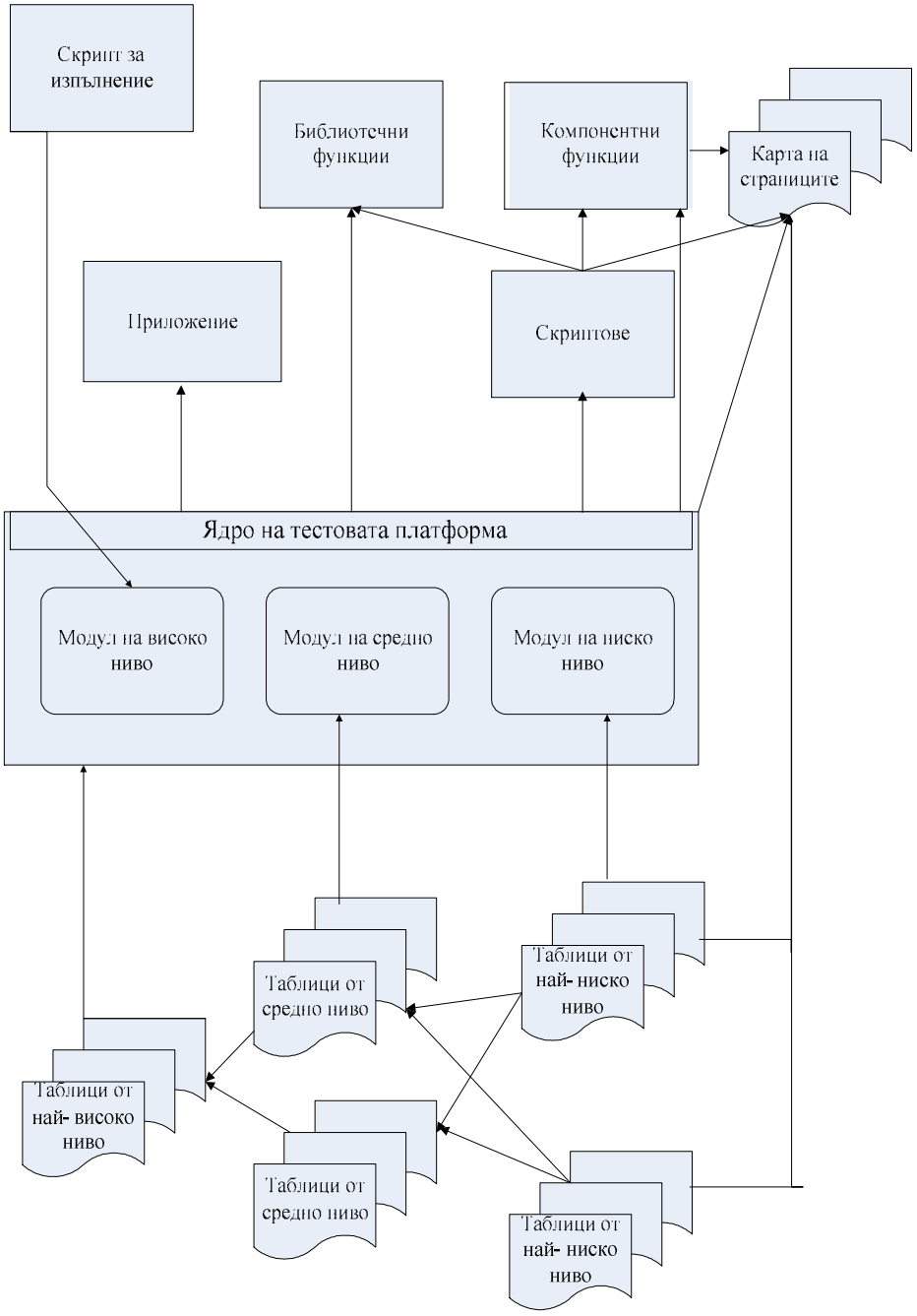
На Фигура 2 е представен модела на тестовата платформа. Най-общо платформата се дефинира от:

- **Ядро (Core Data Driven Engine),** което се състои от имплементацията на модулите за интерпретиране на тестовите таблици по йерархия;
- **Компонентните функции (Component Functions) -** това са функциите които реализират всички взаимодействия на потребителя с елементите на интерфейса на приложението;
- **Библиотеки от функции (Support Libraries) -** реализират общи функции докато библиотеките от функции са използваеми дори извън контекста на конкретната платформа, другите две компонента са силно зависими един от друг както и от библиотеките от функции.

Изпълнението на тест започва със скрипта за изпълнение. Имаме дефинирана йерархия от тестови таблици. Скриптът за изпълнение извиква Ядрото като подава таблица от най - високо ниво на модула за интерпретиране на тези таблици. Този модул извиква модула за интерпретиране на таблиците на средно ниво при всяко срещане на таблица от това ниво. Модулът на средно ниво извиква модула на най - ниско ниво за всяка таблица, която се среща в таблиците на средното ниво. Когато модулът на най - ниско ниво срещне ключова дума или команда, той определя компонента, за който се отнася, и извиква съответната компонентна функция.

Картата на страниците (Application Map) съдържа информацията за съпоставянето на имената на компонентите, използвани в тестовите таблици, и тези в конкретното приложение. Ядрото, компонентните функции и потребителските скриптове използват тази информация. Потребителските скриптове са програми, написани на езика на автоматизирания инструмента за тестване, в които може да е кодирана част от логиката на тестовете Тези скриптове могат да се описват в таблиците и да се извикват при срещането им от ядрото.

Нека да разгледаме подробно всеки елемент.



Фигура 2. Модел на тестовата платформа

3.4.1 Карта на страниците

Както споменахме при разглеждането на предимствата на методологията, базирана на данни и ключови думи, допълнително ниво на абстракция между тестовите таблици и компонентите на интерфейса на приложението дава възможност тестовите таблици да се разработват без да има работещо приложение. Това допълнително ниво се осигурява от картата на страниците. Чрез нея се съпоставят имена на обекти, които са разбираеми за хората, с формата на данните, който се използва от автоматизирания инструмент. Може да се създаде именуваща конвенция за имената на компонентите и прозорците на приложението, които да се използват при създаването на тестовите таблиците. След това в картата на страниците асоцииране всяко име с метода на идентифициране на компонентите, който се използва от инструмента за автоматизирано тестване, за да открие правилния обект в прозореца. Методите за идентифициране на компонентите са дълги символни низове, които съдържат различна информация в зависимост от използвания метод. Различните методи се определят от това в какъв ред ще участват атрибутите на компонента. Методът на идентификация съдържа и йерархията от бащини елементи. Идентификационните низове може да са много дълги, сложни и трудни за разбиране от човек. Затова картите на страниците могат да се изготвят от тестер, който е запознат с инструмента за автоматизирано тестване. Инструментите за автоматизирано тестване предлагат и възможности за разпознаване на компоненти. Тестерите, които изготвят тестовете не е необходимо да познават методите за разпознаване на обектите на определения инструмент. Подредбата на елементите в карта на страниците не е задължително да следва реда на появата им в страницата. Картата на страниците ни дава възможност по лесен начин да отразяваме промените на интерфейса без да е необходимо да променяме тестовите таблици. Таблица 2 представя как изглежда "картата на страниците" на страницата за автентикация на потребителите.

Таблица 2. "Карта на страниците" на страницата за автентикация на потребителите

[LoginPage]		
LoginPage	"Type=Window;Caption={A I S*}"	Window
UserName_Enter	"Type=EditBox;Name=txtLogin"	EditBox
Password_Enter	"Type=EditBox;Name=txtPassword"	EditBox
Login_Action	"Type=PushButton;Name=btnLogin"	PushButton

Картата на страниците може да реализирана чрез текстови файлове, във вид на електронна таблица на MS Excel или таблица във база данни. Платформата осигурява подходящи библиотечни функции за извличане на информацията от картата на страниците в зависимост от използвания формат.

3.4.2 Компонентни функции

Компонентните функции реализират взаимодействието с компонентите. За всеки компонент има различен набор от функции за работа с него. Компонентните функции са дефинирани независимо от приложението и се използват от всяко

приложение, което тестваме. Те използват специфичните за приложението данни от картата на страниците и от тестовите таблици.

Компонентните функции определят ключовите думи, които могат да се използват за всеки компонент. По този начин те дефинират речника, който използваме при създаването на тестовите таблици от най - ниско ниво. Така например за работа с текстови полета част от действията, които можем да използваме в тестовите таблици са :

- SetTextCharacters - за въвеждане на стойност на полето;
- VerifyValue - за верифициране на стойността на полето;

Почти за всички компоненти имаме дефинирани ключови думи за избор на компонента, за действия с мишката както и за верифициране на стойности на характерните за компонента свойства (property). Действията може да изискват допълнителни аргументи. Например при въвеждане на стойност в текстовото поле е необходимо да се зададе допълнителен аргумент със стойността на входните данни. При верифициране на определено свойство е необходимо да се зададе кое е свойство и каква е стойността, която очакваме.

Ключовите думи и техните аргументи, дефинирани от компонентните функции, определят речника, който използваме при създаване на тестовите таблици от най - ниско ниво. Наличие на този речник и именуваща конвенция за имената на компонентите са необходимите предпоставки за създаване на тестовите таблици за проекта.

3.4.3 Тестови таблици

Тестовите сценарии, които се описват от тестовите таблици, са организирани в йерархична структура на 3 нива.

На най-ниско ниво са ***стъпковите таблици***, които съдържат детайлните инструкции за изпълнението на тестовете. Чрез речника на ключовите думи и имената на компонентите и страниците в таблиците е описано в коя страница, за кой компонент от нея какво действие трябва да се извърши и какви са аргументите. Модулът на най-ниско ниво от ядрото на тестовата платформа интерпретира всяка инструкция, зададена с ключова дума. Таблица 3 , Таблица 4 и Таблица 5 представят стъпкови таблици.

Таблица 3. Стъпкова таблица LoginUser за регистриране на потребител на системата

C	SetApplicationMap	ApplicationMap.MAP		
C	StartWebBrowser	http://www.system.com		
T	LoginPage	LoginPage	Maximize	
T	LoginPage	UserName_Enter	SetTextCharacters	“user”
T	LoginPage	Password_Enter	SetTextCharacters	“password”
T	LoginPage	Login_Action	Click	

Таблица 4. Стъпкова таблица VerifySuccessfulLogin за проверка дали регистрирането е успешно

C	SetApplicationMap	ApplicationMap.MAP			

T	HomePage	CurrentUserLogin_TableCell	VerifyPropertyContains	InnerText	“user”

Таблица 5. Стъпкова таблица LogOutUser за излизане от системата на регистриран потребител

C	SetApplicationMap	ApplicationMap.MAP	
T	HomePage	Logout_Link	Click

В представените примери се вижда, че има два вида записи. Редовете от таблиците, които започват с ‘C’ са указание към обработващия модул, че на този ред е зададена команда (системна ключова дума), която не е свързана с определен компонент, а е инструкция към платформата. Така например инструкция към платформата е стартирането на браузър. Редовете на таблиците имат различна дължина и значения на колоните. Така например командите не се отнасят за определен компонент, а ключовата дума може да се отнася както за страница, така и за компонент. Всеки ред се интерпретира отделно и данните се събират докато се появи празна колона (или табулация).

На средно ниво са таблиците, които комбинират няколко стъпкови таблици в тестови сюити. Една и съща таблица от най - ниско ниво може да бъде използвана в различни ***таблици от средно ниво***. Модулът на средно ниво от ядрото, който се грижи за изпълнението на сюитите при срещане на стъпкова таблица я предава на модула за обработване на таблици от най - ниско ниво.

Таблица 6. Таблица на средно ниво LoginRegisteredUserSuite за получаване на достъп до системата на регистриран потребител

T	LoginUser		
T	VerifySuccessfulLogin		
T	LogOutUser		

Таблица 6 представя съдържанието на таблица от средно ниво, която съдържа 3 таблици от най-ниско ниво. Таблицата представя тестова сюита за получаване на достъп до системата на регистриран потребител. Тестовата сюита е съставена от тестов сценарий за регистриране на потребителя, проверка дали е валиден потребител и излизане от системата.

Таблиците от най - високо ниво комбинират няколко сюити. Сюитите могат да бъдат комбинирани по различен начин в зависимост от типа тестване което искаме да извършим - дали ще е на цялата система, на модул, или ще извършваме само част от тестването на системата при предаването и на клиента (acceptance testing). Както и при създаването на таблиците на средно ниво една сюита може да участва в различни таблици на високо ниво. Таблиците на високо ниво се обработват от модула на високо ниво, който предава всяка срещната в таблицата сюита на модула от средно ниво.

Таблица 7. Таблица на най - високо ниво

T	LoginRegisteredUserSuite

Таблица 7 е пример за таблица на най-високо ниво, която съдържа една единствена сюита LoginRegisteredUserSuite.

3.4.4 Ядро на тестовата платформа

Ядрото на тестовата платформа се състои от 3 модула:

- **Модулът на високо ниво** обработва таблиците от високо ниво, които съдържат списък от сюити от тестове за изпълнение. Модулът чете всеки ред на таблицата от високо ниво и изпраща на модула от средно ниво всяка таблица от средно ниво, която намери по време на този процес.
- **Модулът на средно ниво** обработва таблиците от средно ниво, които съдържат списък от стъпкови таблици за изпълнение. Модулът чете всеки ред на таблицата от средно ниво и изпраща на модула от ниско ниво всяка стъпкова таблица, която намери по време на този процес.
- **Модулът на ниско ниво** обработва стъпковите таблици, които съдържат записи от инструкции с ключови думи, дефинирани от компонентите функции. Модулът извършва синхронизации и проверява, че страницата или компонента, който ще обработва, съществуват и са активни. След това обработващия модул изпраща инструкцията на компонентната функция, която ще извърши желаното действие.

Нека да разгледаме псевдокода, който е необходим за обработването на записа :

Прозорец	**Компонент**	**Действие/ Ключова дума**	**Параметри**
Страница за автентикация	Потребителско име	Верифициране на стойността	“Име”

Модул на ниско ниво:

- Провери дали съществува страницата " Страница за автентикация "
- Направи активна страницата " Страница за автентикация ".
- Провери дали съществува компонента " Потребителско име "
- Намери типа на компонента " Потребителско име ".
- Извикай модула, който управлява всички компоненти от този тип компонентите от тип текстово поле

Модул за обработка на компонентни функции на текстово поле:

- Валидиране на ключовата дума „ Верифициране на стойността”
- Извиква функцията Textbox.VerifyValue

Функцията Textbox.VerifyValue:

- Извлечи стойността на текстовото поле " Потребителско име "
- Сравни тази стойност с " Име "
- Запиши дали сравнението е успешно или не

Всички модули притежават множество от команди или системни ключови думи, които може да обработва. Командите са инструкции към тестовата платформа и не са свързани с приложението. Едни от най - често използваните команди, които се използват и в трите модула са:

- UseApplicationMap - определя коя карта на страниците да се използва за идентифициране на компонентите
- LaunchApplication - стартира приложение
- LaunchBrowser - стартира приложение чрез зададено url
- CallScript - извиква скрипт,създаден чрез програмния език на инструмента за автоматизирано тестване
- WaitForWindow - изчаква да се появи определен прозорец или страница

- WaitForWindowGone - изчаква определен прозорец или страница да изчезне
- Pause (Sleep) – спира изпълнението за определен период от време
- LogMessage - записва съобщение в рапорта за изпълнението на тестовете

3.4.5 Библиотеки

Библиотеките осигуряват общи функции за работа с файлове, низове, буфери, променливи, работа с бази данни, лог файлове, създаване на карти на страниците и др. Библиотечните функции се използват от всички компоненти на платформата. Библиотечните функции осигуряват достъп до специфичните компонентите на платформата и позволяват използването им в скриптове.

Процесът на внедряване и използване на тестовата платформа ще разгледаме по-подробно в следващата част.

4. Спецификация на решението за внедряване на базирана на данни методология

4.1 Основни принципи на процеса на внедряване

Интернет базираните приложения предполагат множество итерации, защото доставянето на продукта на клиента изисква еднократна инсталация и предоставя възможност да се осъществява ранна и постоянна връзка с клиента. Това е едно от условията за успешен и качествен проект. Версии на програмата могат да се доставят с много голяма честота. Намаляването на итерациите води до намаляване и на времето за тестване, което не трябва да оказва влияние на степента на покриване на изискванията и намаляване на риска. Това води до необходимост от автоматизиране на голяма част от тестовете на приложението. Често за да се промени или добави една функционалност се правят промени в интерфейсите, които могат да засегнат друга функционалност и да доведат до провал на записаните автоматизирани скриптове. Това е промяна не в логиката на теста, а в интерфейсите и невъзможността да бъде разпознат даден обект от интерфейса. Промените са малки и засягат най - много един от бизнес процесите и съответния тест. Затова е необходимо бързо адаптиране на тестовете за следващите версии.

Внедряването на тестова методология, базирана на данни, има за цел да бъде постигнато максимално качество на разработвания продукт. Тестовият процес трябва да провери и осигури, че изискванията към продукта са изпълнени и запазени през целия процес на разработка. Предимствата на разглежданата методология ни позволяват да се осигури надежден тестов процес в среда на бързо променящи се изисквания и намалено време за тестване. **Основните принципи**, към които ще се придържа процесът на внедряване са:

- повторно използване на готови тестове;
- лесно адаптиране на тестовете;
- лесно създаване и поддръжка на таблиците за ръчно и автоматизирано тестване;
- история на тестовите цикли на версиите на продукта.

Тези принципи са основани на разгледаните в предишната част от дипломната работа предимства на базираната на данни методология за автоматизирано функционално тестване. Сега ще разгледаме как те ще бъдат имплементирани в контекста на конкретното решение за внедряване на методологията. За внедряване на методологията ще използваме разгледаните в предишните глави продукта Rational Robot на фирмата IBM за автоматизирано тестване и надграждащата го платформа RRAFS, предоставяща поддръжка на методологията, базирана на данни, както и разработен инструмент, предоставящ допълнителни функции. Разработването на инструмента ще проследим след като идентифицираме изискванията към него в процеса на внедряване.

Formatted: Bullets and Numbering

~~4.1.1~~4.2 Повторно използване на готови тестове

Два са основните начини за повишаване степента на повторно използване на готови тестове - използване на променливи и функционална декомпозиция на тестове.

Платформата за автоматизирано тестване поддържа ***използването на променливи*** в тестовите таблици и скриптовете. Тя ни предоставя възможности за:

- предаване на информация - данни и променливи, между таблиците;
- предаване на информация между скриптовете;
- предаване на информация от таблиците към скриптовете и обратно;
- параметризиране на команди;
- използване на променливи в картите на страниците за динамично разпознаване на компоненти.

Ще разгледаме подробно първите три възможности, които ще използваме и в примерните тестови сценарии, както и при имплементирането на инструмента.

В тестовите таблици променливите могат да се използват да предават и задават стойности на параметри на командите и ключовите думи. В таблиците от високо ниво променливите се използват за предаване на стойности към таблиците от по - ниско ниво, които извикват. Променливите се въвеждат като се използва символа '^', който трябва да е първия символ в съответното поле. Между него и името на променливата не се слага празен символ За присвояване на стойност на променлива се използва знака за равенство '='. Стойността на променливата е видима каскадно в таблиците от по-ниските нива. Стойност, присвоена в таблиците от по - ниските нива, предефинира тази стойност. Чрез използването на променливи тестовете могат да бъдат използвани многократно като се променят стойностите на променливите при извикване на тестовете в таблиците от по - високото ниво. На Пример 1 е представено използването на променливата ^UserName за присвояване на стойност на полето UserName_Enter в ред от тестова таблица.

T	LoginPage	UserName_Enter	SetTextCharacters	^UserName

Пример 1. Използване на променлива в тестова таблица

Стойност на променлива може да се присвои и чрез командата SetVariableValue по следния начин:

C, SetVariableValues, UserName= "John Smith".

Библиотеката DDVVariableStore предоставя функции за работа с променливи във всеки скрипт, който я използва, независимо дали е скрипт на платформата или не. Стойност на променлива се присвоява чрез функцията DDVSetVariableValue, която приема 2 параметъра - името на променливата и стойността:

```
$INCLUDE: "DDVariableStore.SBH"
Dim Result As Integer
Result = DDVSetVariableValue("UserName", "John Smith")
Стойност на променлива се извлича чрез функцията DDVGetVariableValue
Dim Result As Integer
Dim varUserName As Variant
Result = DDVGetVariableValue("UserName", varUserName)
```

Тестовата платформа предоставя възможност стойностите на променливите, използвани в тестовите таблица, да са достъпни в скриптовете чрез функциите на

разгледаната библиотека. Също така стойност на променлива, зададена чрез SetVariableValue в потребителски скрипт, може да бъде използвана в тестовите таблици. По този начин стойности на променливи, присвоени по време на изпълнение на тест могат да се обработват след това в скрипт. Също така в тестовите таблици могат да се използват случайно генерирани стойности от пулове от данни (datapools) като променливите се дефинират и им се присвояват стойности предварително в извикващия скрипт.

Функционалната декомпозиция е метод за повишаване на степента на повторно използване на тестовете /и разбира се на всякакъв вид код/ при всеки модулен и йерархичен подход на организиране на тестовете. Тъй като тестовите таблици оформят йерархична структура от тестове този метод може да се приложи при създаването на стъпковите таблици, които определят най - малкото ниво на грануларност. Стъпковите таблици задават декомпозицията на тестовете в 3-те основни типа - бизнес функционалност, навигационни и верифициращи. Именуваща конвенция на типовете стъпкови таблици, която ще използваме за лесната им поддръжка е:

- таблиците, които тестват бизнес логика да се именуват с бизнес функционалността, която тестват - например за добавяне на потребител - AddUser
- таблиците, които съдържат навигация към определена страница съдържат името на страницата, както е зададено в картата на страниците и се използва в тестовите таблици, пред което се поставя префикс ‘То’-‘към’. Например таблицата, която съдържа информацията за навигация към страницата за добавяне на потребител ще се казва ToAddUser
- таблиците, които съдържат верификация на дадена бизнес функционалност или навигационна връзка съдържат името на страницата пред което се поставя префикс ‘Verify. Могат да съдържат и информация дали верификацията се извършва върху валидни входни данни - VerifySuccessfulAddUser. Добра практика е да се създават положителни тестови сценарии, които приемат валидни данни както и отрицателни, които проверяват поведението на системата при невалидни входни данни

~~4.1.2~~4.3 Лесно адаптиране на тестовете

Formatted: Bullets and Numbering

Картата на таблиците осигурява допълнителното ниво на абстрактност между приложението и тестовите таблици. По този начин е възможно честите промени на интерфейса на интернет приложенията да не водят до промени в тестовите таблици, а само в картата на страниците. Така промени, които засягат идентификационните низове на компонентите, могат да бъдат отразени като се актуализира картата на страницата и се запазят асоциираните с тези низове имена на компоненти, които се използват в тестовите таблици.

В Rational Robot картите на страниците са текстови файлове, които съдържат именувани секции с именувани компоненти във всяка секция както е представено в Пример 2.

```
[LoginPage]
LoginPage="Type=Window;Caption={A I S*}"
UserName_Enter="Type=EditBox;Name=txtLogin"
Password_Enter="Type=EditBox;Name=txtPassword"
Login_Action="Type=PushButton;Name=btnLogin"
[HomePage]
HomePage="Type=Window;Caption={A I S*}"
left="Type=HTMLFrame;Name=left"
Administration_Link="Type=HTMLFrame;HTMLId=left;\;Type=HTMLLink;HTMLText=
Administration"
AddUser_Link="Type=HTMLFrame;HTMLId=left;\;Type=HTMLLink;HTMLText=Add
User"
```

Пример 2. Съдържание на карта на страниците

Всяка секция се отнася за една страница на приложението. Секцията се именува с името на страницата в квадратни скоби. Първият компонент във всяка секция е описанието на самата страница - име и идентификационен низ след което следват описанията на дъщерните елементи.

Тестовата платформа RRAFS предоставя инструмент, наречен Process Container за създаване на карти на страниците. Инструментът ни предоставя следните предимства:

- създава "най-правилните" [13] идентификационните низове на всички дъщерни елементи на страницата
- предоставя възможност определена от потребителя информация да бъде добавена към идентификационните низове
- създадената карта на страницата може да се добави към съществуващ файл с дефиниции на карти на страниците
- създадената карта на страниците може да се запише / или добави/ в електронна таблица на MS Excel

При записване на картите на страниците инструментът на платформата генерира и стойности по подразбиране на асоциираните с идентификационните низове имена, които се използват в тестовите таблици.

В картите на страниците могат да се дефинират константи и променливи, които да се използват в тестовите таблици. За разпознаване на контроли и компоненти, които не се поддържат от тестовата платформа, в инструментите за автоматизирано тестване се използват координати. Картите на страниците позволяват разпознаването на компоненти на базата на координати. Няма да се спираме подробно на тези възможности.

Тестовата платформа позволява дефиниране на картата на страниците във всеки тип тестова таблица. При внедряване на методологията ще наложим изискването картата на страниците да се дефинира само в стъпковите таблици. Стъпковите таблици са единиците, които ще подлежат на многократно повторно използване и трябва да е ясно коя е асоциираната карта на страниците с всяка от тях.

~~4.1.3~~4.4 Лесно създаване и поддръжка на таблиците за ръчно и автоматизирано тестване

Formatted: Bullets and Numbering

Три са основните начини за лесно създаване и поддръжка на таблиците за ръчно и автоматизирано тестване - формат на таблиците, именуваща конвенция и автоматизирано създаване на тестове

Формат на таблиците

Модулите за обработка на тестовите таблици в RRAFS работят с текстови файлове, в които елементите са отделени с определени разделителни символи. В скрипта за изпълнение се определя и какъв да бъде разделителя за всеки тип таблица. В зависимост от типа на тестовите таблици са определят и файловите разширения, които се използват - стъпковите таблици имат разширение .sdd, таблиците, които съдържат сюитите, имат разширение .std, а таблиците от най - високо ниво имат разширение . cdd. Тестовата платформа предоставя функции за преформатиране на електронни таблици на MS Excel в текстови файлове, с които работи платформата. Работата с продукта Excel на фирмата Microsoft е лесен и защитен от грешки начин за създаване и промяна на тестови таблици като се запазва правилния формат. При създаване на тестови таблици в MS Excel трябва да се спазват следните правила за формата на таблиците:

- всеки ред се обработва поотделно в зависимост от вида му
- за край на ред се счита срещата на колона без данни
- ред, чийто първи символ е ‘;’,’’’ или първата колона е празна, е коментар и не се обработва
- ред, в който първата колона е един от символите ‘T’,’C’,’S’ определят типа на записа по следния начин:
    - ‘T’ - тестова стъпка. В стъпковите таблици тези редове определят какво действие ще се предприеме в зависимост от ключовата дума и за кой компонент. Подредбата на колоните е следната:
        - В таблиците от по-високо ниво този тип на записа указва какви таблици от по - ниско ниво съдържа. Редовете от този тип имат следния формат:
        - Липсата на запис в колоната Разделител не е сигнал за край на ред. В този случай се счита, че разделител е табулация.
    - ‘C’ - команда. Командите са инструкции към тестовата платформа и имат следния формат:
    - ‘S’ - тестова стъпка, която не се обработва.
- ред, който не е коментар и типа на записа е различен от изброените, задава името на тестов скрипт на инструмента за автоматизирано тестване. Не е ограничено използването на скриптове в различните видове тестови таблици. При внедряване на методологията ще предполагаме използването на скриптове в таблиците от високо ниво. Използването на скриптове в таблиците от най - ниско ниво не е добра практика, защото скриптовете се изграждат с по - голямо ниво на грануларност.

Създаване на унифициран формат на таблиците е едно от основните неща, които трябва да бъдат съобразени при внедряване на методологията, защото те ще се използват и за ръчно тестване. Също така това е една от основните предпоставки за

лесната им поддръжка. Шаблоните, които ще използваме, са представени в Таблица 8, Таблица 9 и Таблица 10.

Таблица 8. Шаблон за стъпковите таблици

	Project Name:				
	Prerequisites:				
	Test Case:				
;RT	**WINDOW/ COMMAND**	**COMP/ ARG**	**ACTION/ ARG**	**ARG**	**ARG**

Таблица 9. Шаблон за таблиците, описващи сюити

	Project Name:			
	Test Server:			
	OS:			
	Browser:			
	QA Engineer:			
	Date:			
	Prerequisites:			
	Test Suite:			
;RT	**STEPS/ COMMAND**	**SEPARATOR/ ARG**	**ARG**	**ARG**

Таблица 10. Шаблон за таблиците, описващи тестови цикли

	Project Name:			
	Test Server:			
	OS:			
	Browser:			
	QA Engineer:			
	Date:			
	Prerequisites:			
	Test Cycle:			
;RT	**SUITES/ COMMAND**	**SEPARATOR/ ARG**	**ARG**	**ARG**

Като разделител във всички текстови файлове на текстовите таблици ще използваме табулация. Затова в тестовите таблици от високо ниво колоната с име

разделител (separator) винаги ще е празна.

Именуваща конвенция

Създаване на конвенция за именуване на компонентите на приложението също е предпоставка за лесната им поддръжка. Имената на компонентите трябва еднозначно и ясно да определят компонентите на страницата. Именуващата конвенция, която сме избрали е определена от името и функционалността на компонента. При наличие на конфликт при компоненти с еднакво предназначение и функционалност се добавя и типа на компонента. Записването на функционалността, а не типа на компонента ни позволява при промяна на типа на компонента с друг, близък по функционалност, да не се променят стъпковите таблици. Освен това компонентите с близка функционалност имат и идентични ключови думи за описване на действията им. Основните правила са:

- първата част от името на компонента се определя от текста на компонента, определящия го етикет или името на колона от таблица. При наличие на множество компоненти се слага индекс. Името на компонента може да наследява името на бащиния компонент, ако предишните правила не могат да бъдат изпълнени.
- втората част на името се определя от функционалността на компонента по следния начин:
    - компоненти, които се използват за въвеждане на информация от потребителя - TextBox, ComboBox - втората част на името е Enter
    - компоненти, които се използват за представяне на опции - CheckBox, RadioButton - втората част на името е Check
    - компоненти, които се използват за представяне на избор – Lists, ComboBox - втората част на името е Select
    - компоненти, които се използват за представяне на действие, което потребителя може да предприеме - Button, Icon - втората част на името е Action
    - при наличие на конфликт с имената към втората част на името се добавя и типа на компонента
    - всички останали компоненти, както и компонентите, споменати в горните правила, но изпълняващи други функции, се взима типа на компонента
- първата и втората част се разграничават с символа ‘_’

Автоматизирано създаване на таблиците от високо ниво

Таблиците от най-ниско ниво могат да бъдат създадени предварително и не претърпяват значителни промени след създаването си. При настъпването на промени те се отразяват предимно в картите на страниците или се налага добавянето на нови таблици. Основните тестови сценарии също се специфицират рано и някои от таблиците от високите нива също се създават предварително. Но таблиците от високите нива имат много по-динамично поведение, което се определя от конкретните нужди и типа тестване. Тяхното създаването е свързано с подреждане на вече създадени и нови таблици от ниско ниво и задаване на стойности на параметрите. В рамките на кратките интервали за създаване на версиите на продукта съотношението между новите и вече създадените таблици от ниско ниво е много малко. Пълни тестове на новата функционалност и всички зависими бизнес сценарии трябва да бъдат направени. Времето за тестване се използва предимно за специфициране на новите

тестови таблици от високо ниво. Автоматизираното им създаване ще намали както времето за описване така и възможните графични грешки при попълването на тестовите таблици.

Как точно се автоматизира процеса на създаване на таблиците от високо ниво ще проследим като разгледаме спецификацията и разработката на инструмента за автоматизирано създаване на тестове в следващата част.

~~4.1.4~~4.5 История на тестовите цикли на версиите на продукта

Formatted: Bullets and Numbering

Поддържане на история на тестовите цикли и версиите на продукта има няколко предимства:

- тестовите таблици могат да запазят името си при промяна в контекста на дадена версия. По този начин не е необходимо таблиците от високите нива, които съдържат таблицата да се променят
- за всяка една версия на продукта във всеки един момент има работещи тестови сценарии
- повторно използване на цели или части от тестове в следващи версии
- използване на тестовите логове за проследяване на изпълнението на тестовете през различните версии

Тестовата платформа RRAFS предоставя функции за работа с лог файлове. Те могат да се използват за записване на тестовите резултати и други съобщения към лог файла и/или конзолата на инструмента, както и към текстови лог файлове и лог файлове в xml формат. Логовете се дефинират чрез потребителския тип LogFacility. Платформата предоставя глобален лог - MainLog, който може да се използва от всички скриптове. Даден скрипт може да има няколко активни лога, в които да пише. При записването на всяко съобщение се специфицира и в кой от активните логове да бъде записано. При създаването на обектите, дефиниращи лога, се определя и начина на записване на съобщения в новия лог. Предефинираните константи, определящи начина на запис са:

- LOGGING_DISABLED - не се записват съобщения в този лог
- SQALOG_ENABLED - активирано е записването към лога на инструмента. Тъй като този лог е единствен всички обекти, които са дефинирани с тази опция, ще записват в един лог.
- TEXTLOG_ENABLED - активира се записването в текстов лог
- HTMLLOG_ENABLED - използва се като флаг, указващ, че текстовият файл трябва да се обработи допълнително и да се направи html рапорт
- CONSOLE_ENABLED - активира се записването в конзолата на инструмента. Тъй като тази конзола е единствена всички логове, дефинирани с този статус ще пишат в една конзола
- XMLLOG_ENABLED - активира се записването в текстов лог в xml формат
- MAX_LOGMODE - активира всички статуси

За определяне на начина на записване на съобщения в лога могат да се използва една или комбинация от разгледаните константи.

Поддържането на история на тестовите цикли и версии на продукта също ще бъде осигурено от инструмента, чиято спецификация и разработка е обект на разглеждане в следващата част.

5. Реализация на инструмента за автоматизирано създаване на тестове

5.1 Дизайн на инструмента за автоматизирано създаване на тестове

При разглеждането на основните принципи, към които се придържаме при внедряването на методологията, стана ясно, че част от тях ще бъдат осигурени от специално разработен инструмент. Нека обобщим какви ще бъдат изискванията при създаването на инструмента:

- *Автоматизирано създаване на таблици от високо ниво*
- *Поддържане на история на тестовите цикли и версиите на продукта*

Създаване на таблици от високо ниво е процес свързан с подредбата на тестовете в йерархична структура. Затова при реализацията на инструмента е използвано дърво за представяне на структурата на тестовете. В тази дървовидна структура се разграничават няколко вида елементи - елементи, представящи трите вида тестови таблици, както и елементи, представящи картите на таблиците, скриптовете, променливите и предпоставките за всеки от типовете тестови таблици. Елементите, представящи тестовите таблици могат да имат дъщерни елементи, докато другите елементи, представят листа на дървовидната структура. Всеки тип елемент дефинира множество от операции, които могат да бъдат изпълнени. Нека да разгледаме типовете елементи по – подробно.

Елементи, представящи променливи

Всички променливи, които се срещат в дадена тестова таблица са представени с дъщерни елементи на съответния клон за таблицата. Променливите са листа в дървовидната структура. Тези елементи се създават по два начина - имплицитно при добавяне на елемент, представящ съществуваща тестова таблица, към дървото или експлицитно. Експлицитното създаване е за да се дефинира параметър в таблиците на високите нива. Тези параметри са обвързани с таблицата от по - ниското ниво. Параметри за стъпковите таблици не могат да бъдат създавани експлицитно. За всички елементи променливи е възможно присвояване на стойност и промяна на стойността.

Елементи, представящи предпоставки

Предпоставките към тестовете са представени с дъщерен елемент на съответния елемент. Те се създават при всяко добавяне към дървото на елемент, представящ тестова таблица, в която присъства секцията за описване на предпоставки. Инструментът предоставя възможност за редактиране на текста на предпоставките.

Елементи, представящи картите на страниците

Елементи, представящи картите на страниците, са дъщерни елементи за елементите на стъпковите тестовите таблици. Те се създават при добавяне на стъпкова таблица към дървото. Инструментът предоставя възможност за редактиране на текста на картата на страниците.

Елементи, представящи скриптове на езика на инструмента за автоматизирано тестване

Както разгледахме скриптовете могат да участват в тестовите таблици от високо ниво. Елементите, представящи скриптове са дъщерни елементи на съответния елемент на таблицата. Такива елементи се създават имплицитно при добавяне на бащиния елемент към дървото или експлицитно се добавя към структурата на елемент, представящ таблица от високо ниво. Могат да се добавят само съществуващи скриптове. Инструментът предоставя възможност за разглеждане, но не и за редактиране на текста на скриптовете.

Елементи, представящи лог файловете

Лог файловете се създават при изпълнение на тестов цикъл. Лог файловете се генерират от тестовата платформа. Инструментът предоставя възможност за разглеждане и съхранение на файловете, които съдържат информацията за изпълнението на тестовете за всяка от версиите.

Елементи, представящи стъпковите таблици

Стъпковите таблици са представени с елементи, които са дъщерни на елементите, представящи таблица, описваща сюита от тестове. Тези елементи се създават по два начина - имплицитно при създаване на елемент на тестова сюита, съдържаща теста и експлицитно при добавяне към структурата на елемент, представящ тестова сюита. За всяка стъпкова таблица могат да се дефинират променливи, които да се използват като аргументи при извикването и от таблиците на по - високо ниво. Елементите, представящи тези променливи влизат в структурата на бащината тестова таблица. Инструментът предоставя възможност за разглеждане, но не и за редактиране на текста на стъпковите таблици.

Елементи, представящи таблиците, описващи тестовите сюити

Сюитите са едно ниво под тестовите цикли в йерархията на тестовете и съответно в дървовидната структура. Елементите на таблиците, описващи сюитите, се създават по няколко начина - имплицитно при създаване на елемент на тестов цикъл, съдържащ сюитата и експлицитно при добавяне към структурата на елемент, представящ тестов цикъл. Към дървовидната структура могат да бъдат добавяни елементи, представящи както нови така и съществуващи тестови сюити. За всяка сюита могат да се дефинират променливи, които да се използват като аргументи при извикването и от таблиците на по - високо ниво. Елементите, представящи тези променливи влизат в структурата на бащината тестова таблица. Инструментът предоставя възможност само за разглеждане на текста на таблиците, описващи сюити.

Елементи, представящи таблиците, описващи тестовите цикли

Инструментът дава възможност за създаване на нов тестов цикъл или добавяне на съществуващ. Към създадения елемент на дървото могат да бъдат добавяни елементи, представящи нови или съществуващи таблици със сюити от тестове. Инструментът предоставя възможност само за разглеждане на текста на таблиците, описващи тестовите цикли.

Дървовидната структура, представяща организацията на тестовете има един единствен корен, който се определя от версията на продукта, която подлежи на тестване. Инструментът дава възможност за създаване на нова версия на продукта или отваряне на съществуваща. При всяко инициализиране на корена на дървото с

текущата версия на продукта се създава нова директория в определена от потребителя папка. В тази директория ще се съдържа цялата информация, която е необходима за изпълнение на тестовите цикли за тази версия. При добавяне на елемент, представящ съществуваща тестова таблица към дървовидната структура, всички файлове, които са свързани с нея и самата таблица се копират в текущата директория. Свързаните с таблицата файлове се търсят в директорията, от която се взима тестовата таблица. При добавянето на нов елемент към дървовидната структура, който представя тестова таблица, се създава нов файл в директорията на текущата версия.

Съхранението на тестовите файлове, скриптовете и картите на страниците в директории по версии дава възможност за поддържане на история на тестовите цикли на продукта. Информация за изпълнението на тестовите сценарии за всяка версия се осигурява от лог файлове, които се генерират от тестовата платформа при всяко изпълнение на тестов цикъл. Лог файловете се прехвърлят от работната директория на тестовата платформа в директорията на съответната версия. Файловете могат да се добавят към текущата версия при всяко изпълнение на тестов цикъл.

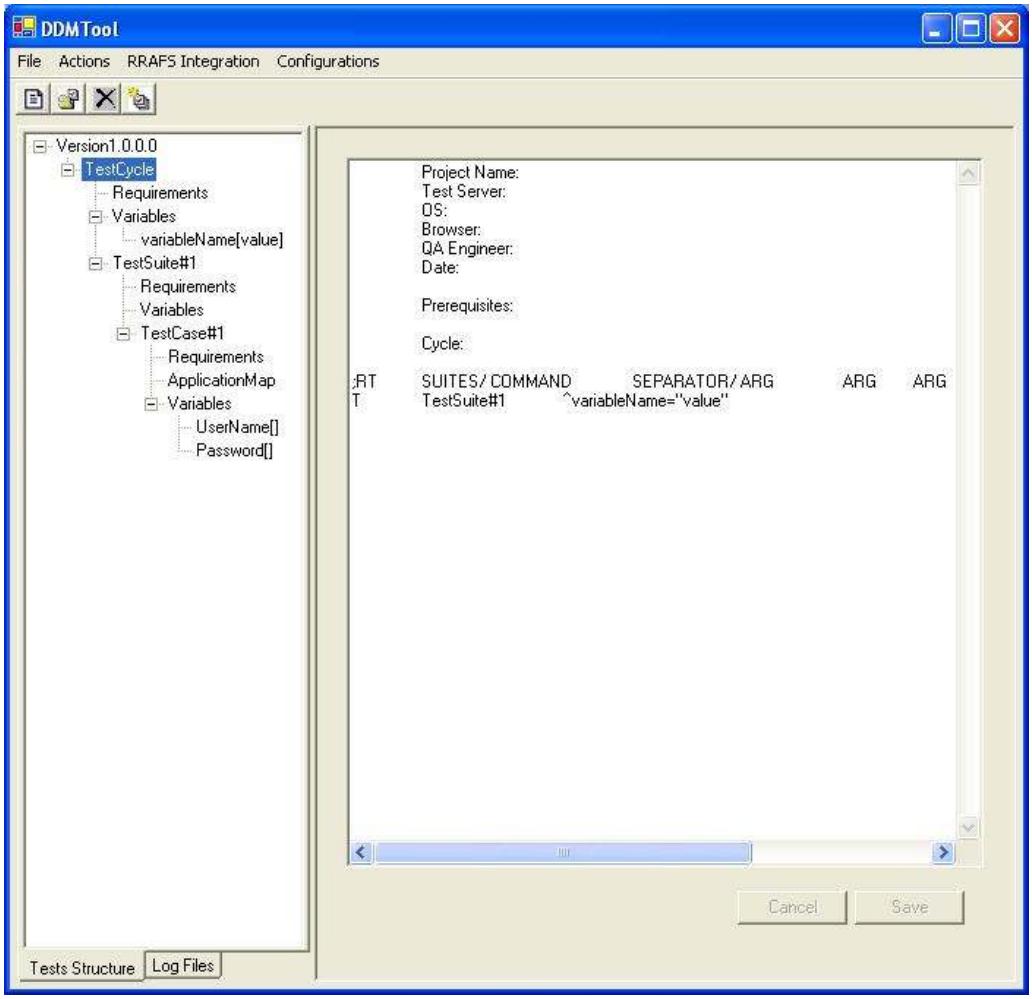
5.2 Имплементация на инструмента

За физическата и логическата организация на класовете при реализацията на инструмента са използвани няколко директории, върху които са дефинирани и пространства от имена. Нека разгледаме всяко едно от тях.

Пространство от имена DDMTool

В главното пространството от имена, наречено DDMTool са дефинирани елементите на потребителския интерфейс, класът за обработване на конфигурации, както и 3 дъщерни пространства от имена. Класът DDMTool дефинира главната форма, която описва дървото от тестове, дървото, представящо лог файловете, и главното меню. На Фигура 3 е представен интерфейсът на инструмента.

При обработване на информацията, представяна в елементите на дървото, се използват различни потребителски контроли. За визуализиране на файлове е реализиран класът FileEditor. Класът VariableEditor дефинира потребителски контрол за редактиране на променливи. Класовете InputFileDialog, InputVariableDialog и InputVersionDialog представят диалогови прозорци за въвеждане на данни от потребителя. Класът ProjectSettings съдържа и предоставя информация за различните конфигурации на проекта. Конфигурациите на проекта се записват в xml файл. Файлът съдържа информация за конфигурации на проекта като директорията по подразбиране, в която се създават тестовете, и за директорията, в която се намират шаблоните, които се използват при създаване на нови тестови таблици. Освен това във файла са описани и пътищата към директории на тестовата платформа RRAFS, които използва при прехвърляне на тестовите таблици за изпълнение и на лог файловете за анализиране на резултатите. Класовете RRAFSConfigurationsDialog и ProjectConfigurationsDialog дефинират диалогови прозорци за промяна на конфигурациите на проекта и интеграцията с тестовата платформа.

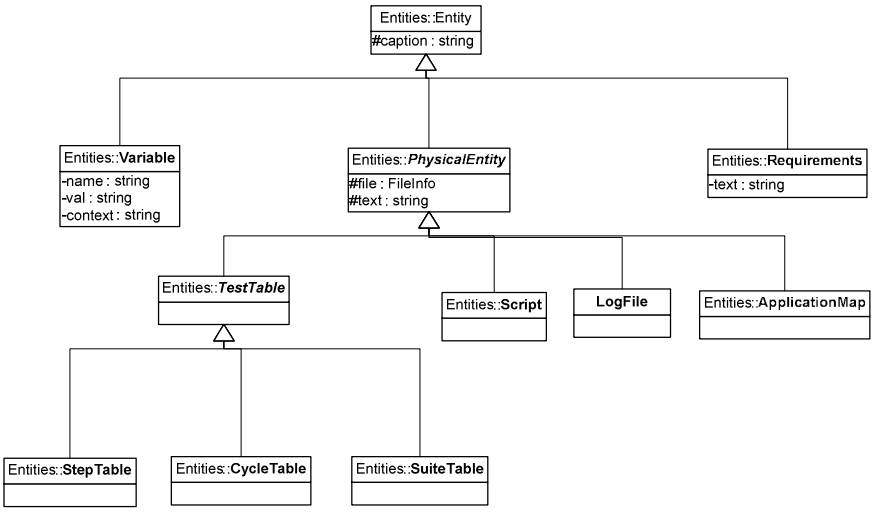


Фигура 3. Интерфейс на инструмента

Пространство от имена Entities

В пространството от имена Entities са дефинирани основните елементи, които ще участват в дървовидната структура. Това са елементите дефиниращи променливи, карти на страниците, предпоставки, скриптове, лог файлове, стъпкови таблици, таблици със сюити и таблици, описващи тестовите цикли. На Фигура 4 е представена каква структура са организирани класовете, представящи тези елементи, заедно с няколко абстрактни класа, които групират сходните елементи.

В класовете, дефинирани в това пространство от имена, е отделена работата с файлове при създаване на нови тестови таблици, четенето и копирането на съществуващи, както и записването на промените, направени през интерфейса.



Фигура 4. Диаграма на класовете в пространството от имена Entities

Пространство от имена LogicalStricture

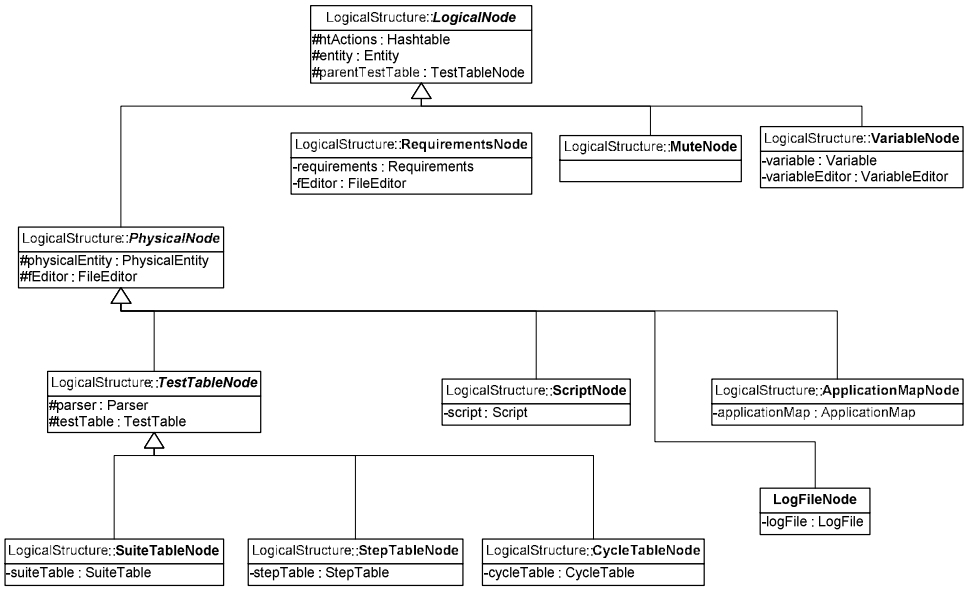
В пространството от имена LogicalStricture са дефинирани класовете, представящи различните типове клони на дървото. Различните типове са определени в зависимост от основните елементи. Тези класове определят същия тип структура, както и основните елементи. Структурата е представена на Фигура 5.

Към статичната структура е добавен е клас, наречен MuteNode, който представя неподредено множество от еднакви елементи. По този начин се представят променливите и логовете.

За всеки тип има дефинирани определено множество от операции, които могат да бъдат изпълнени. Всяка операция от множеството определя контекстното меню и менюто с действията за съответния тип клон на дървото. При избор на определена операция от менюто се извиква метод на клона на дървото, който в зависимост от типа на клона и името на операцията на менюто определя кой метод на класа трябва да изпълни операцията. При изпълнението на операцията при необходимост се използват и класовете за обработване на таблиците.

Операциите, които могат да бъдат изпълнени за всеки тип клон на дървото са следните:

- Тип променлива:
    - **View** - операция, която предоставя възможност да бъдат разгледани стойността, името и контекста, в който се среща променливата. Дава се възможност за промяна на стойността на променливата през интерфейса за разглеждане на променлива.



Фигура 5. Диаграма на класовете в пространството от имена Logical Structure

- Тип предпоставки**:**
  - **View** – Съдържанието на секцията за предпоставки може да бъде разгледано както и да бъде променено. Промените се отразяват в съответната тестова таблица.
- Тип карта на страниците**:**
  - **View** – Съдържанието на картата на страниците, която се използва от съответната тестова таблица може да бъде разгледано в инструмента.
- Тип скрипт**:**
  - **View** – Операцията показва текста на скрипта в контрола за показване и редактиране на текст. Съдържанието на скрипта може единствено да бъде разглеждано, но не и да бъде променяно.
- Тип лог файл:
  - **View** – Съдържанието на всеки лог файл може да бъде разгледано след като бъде преместен в тестовата директория.
- Тип стъпкова таблица**:**
  - **View** - Съдържанието на цялата стъпкова таблица може да бъде разглеждано, но не може да бъде директно променяно. Промените направени на променливите и предпоставките на таблицата се отразяват и могат да бъдат видени.
  - **Delete** - Тестът, представен от стъпковата таблица може да бъде изтрит от йерархията на тестовете. Съответния ред се изтрива от таблицата, представяща тестовата сюита. Физически файлът продължава да съществува в тестовата директория
  - **Define Variable** – Тази операция дава възможност да бъде добавена променлива, която да се използва при изпълнението на теста.

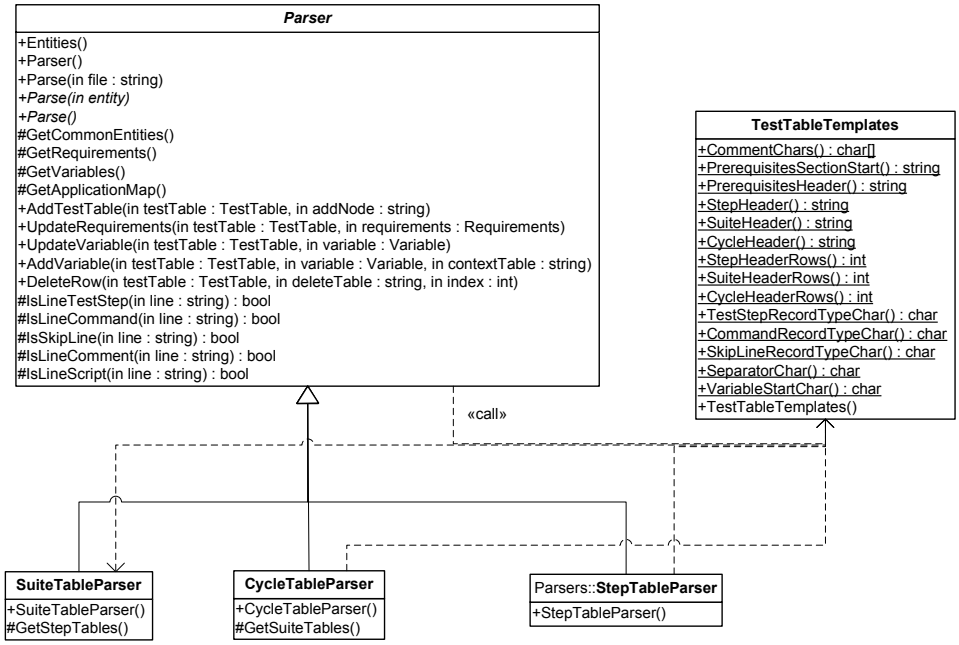
Променливата и нейната стойност се записват в тестовата таблица, представяща съответната сюита, в която участва теста. При изпълнение на теста може да се използва дефинираната променлива.

- **Add Existing TestCase Before** – Тази операция дава възможност да се добави в йерархията тест преди избрания тест в момента. При това действие съответния запис се добавя в таблицата от по- високо ниво.

- Тип таблица, представяща тестова сюита**:**
  - **View** - Съдържанието на цялата таблица може да бъде разглеждано, но не може да бъде директно променяно. Промените направени на променливите, предпоставките и записите на йерархията от тестове се отразяват и могат да бъдат видени.
  - **Delete** – Тестовата сюита, представена от таблицата може да бъде изтрит от йерархията на тестовете. Съответния ред се изтрива от таблицата, представяща тестовия цикъл. Физически файлът продължава да съществува в тестовата директория
  - **Define Variable** – Тази операция дава възможност да бъде добавена променлива, която да се използва при изпълнението на тестовата сюита. Променливата и нейната стойност се записват в тестовата таблица, представяща съответния тестов цикъл, в която участва тестовата сюита. При изпълнение на теста може да се използва дефинираната променлива.
  - **Add Existing TestCase** – Тази операция дава възможност да се добави в йерархията от тестове, представена от таблицата, съществуващ тест. Тестът може да се намира на произволно място. При добавянето му в тестовата сюита файлът с тестовата таблица както и всички други файлове, които реферира, се копират в директорията на текущата версия.
  - **Add Existing TestSuite Before** – При изпълнението на тази операция към йерархията от тестови сюити преди избраната се добавя съществуваща тестова сюита. Тестовата сюита може да се намира на произволно място. При добавянето й в тестовия цикъл файлът с тестовата таблица както и всички други файлове, които реферира, се копират в директорията на текущата версия. В дървото се добавят всички тестове, които описва таблицата, представяща тестовата сюита.
  - **Add New TestSuite Before** – При изпълнението на тази операция към йерархията от тестови сюити преди избраната се добавя нова тестова сюита. Файлът, който се създава, копира зададения в конфигурациите на проекта шаблон на таблица, представяща тестова сюита.
- Тип таблица, представяща тестов цикъл**:**
  - **View** - Съдържанието на цялата таблица може да бъде разглеждано, но не може да бъде директно променяно. Промените направени на променливите, предпоставките и записите на йерархията от тестове се отразяват и могат да бъдат видени.
  - **Delete** – Тестовият цикъл, представен от таблицата, може да бъде изтрит от дървото. Физически файлът продължава да съществува в тестовата директория
  - **Add Existing TestSuite** – При изпълнението на тази операция към йерархията от тестови сюити се добавя съществуваща тестова сюита. Тестовата сюита се добавя като последен елемент.
  - **Add New TestSuite** – При изпълнението на тази операция към йерархията от тестови сюити се добавя нова тестова сюита. Тестовата сюита се добавя като последен елемент.

Пространство от имена Parsers

В пространството от имена Parsers са дефинирани класовете, които обработват тестовите таблици и създават съответната йерархия от елементи. Структурата на класовете от това пространство от имена е представена на Фигура 6.



Фигура 6. Диаграма на класовете в пространството от имена Parsers

Класът StepTableParser обработва стъпковите таблици, класът SuiteTableParser обработва таблиците, представящи сюити, а класът CycleTableParser обработва таблици, описващи тестови цикли. В абстрактния клас Parser са описани и операции, които са общи за всички типове таблици.

В класа TestTableTemplates са описани основните елементи на тестовите таблици като заглавните части на секциите в таблиците, символите за разделител, тип на запис и др.

Инструментът предоставя и няколко функции за осъществяване на интеграцията с тестовата платформа RRAFS - Export Test For Execution и Import Log Files. Export Test For Execution е функция за прехвърляне на създадените тестови файлове в работната директория на тестовата платформа RRAFS. Когато йерархията от тестове бъде завършена автоматично генерираните тестови таблици могат да бъдат прехвърлени за изпълнение в определените директории на платформата. Import Log Files копира създадените в резултат на изпълнението на тестовете лог файлове в директорията на текущата версия. Лог файловете могат да бъдат разглеждани в редактора на инструмента.

Инструментът дава възможност за създаване на нови проекти и разглеждане на стари. Лог файловете към съответния проект носят информация за това как са протекли тестовете. Генерираните с инструмента тестови таблици могат да бъдат използвани в следващите версии.

5.3 Приложение на инструмента

Разработеният инструмент за автоматизирано създаване на тестови таблици се използва в отдела за управление на качеството на фирмата Рила Солюшънс. Новата методология намира приложение в процеса на тестване на интернет базирана система, разработвана на .Net. В процеса на разработка на инструмента екипът от инженери по качеството към фирмата предложи усъвършенствания на инструмента, свързани с дизайна на интерфейсите, конфигурирането на тестовите проектите и работата с лог файлове. При реалната експлоатация на инструмента ще възникнат и множество други възможности за промени и усъвършенствания , които ще бъдат отразени.

6. Внедряване на методологията

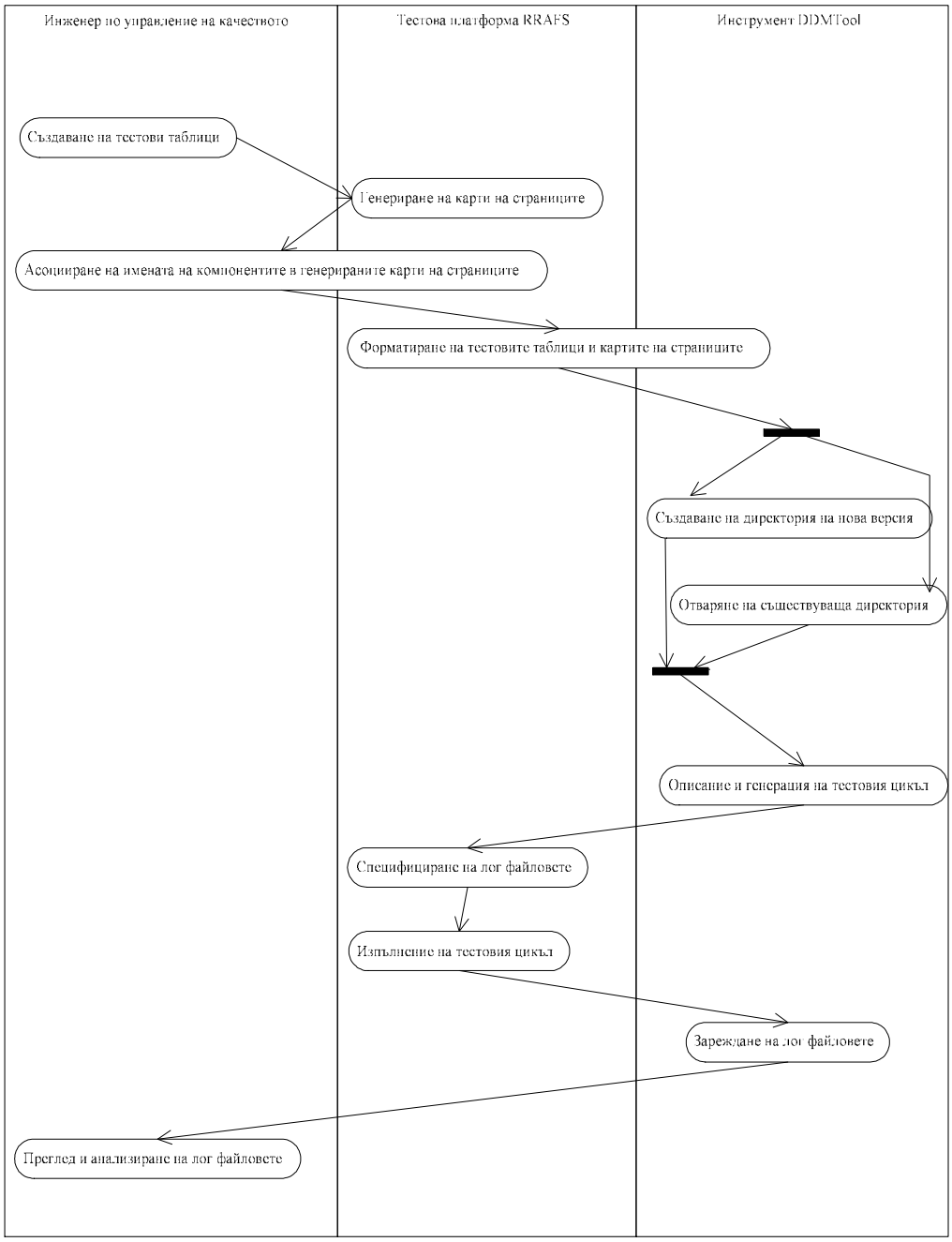
6.1 Процес на тестване

След като разгледахме всички компоненти на процеса на внедряване на методологията нека да разгледаме как протича цялостния процес на създаване и изпълнение на тестове. На Фигура 7 е представена диаграма на активностите на тестовия процес.

След завършване на спецификацията на продукта може да започне създаването на тестовите таблици, описващи тестовите сценарии на продукта, който подлежи на тестване. Тестовите таблици се създават ръчно на MS Excel като се използват шаблоните, които определят формата им. Непосредствено преди започване на даден тестов цикъл с инструмента ProcessContainer на тестовата платформа се изготвя картата/картите на страниците и се асоциират със съответните стъпкови таблици. Картата на страниците се генерира в електронна таблица на MS Excel. От нея се изключват компонентите, които не участват в тестовите таблици. Асоциираните с идентификационните низове имена на компонентите се променят спрямо именуващата конвенция.

Като се използват скриптове за промяна на файловия формат се създават текстовите файлове за тестовите таблици и картата на страниците, които се използват от RRAFS и от инструмента. Тестовата платформа предоставя функции за създаване на тестовите таблици и картите на страниците от електронни таблици на MS Excel. В следващия фрагмент от код представя част от скрипта за промяна на файловия формат:

```
If compExists=0 then
      extension= GetField(tableNAme,2,".")
      compExtension=StrComp(extension,"xls",1)
      If compExtension=0 then
         'input and output directories
         dirOUT = SQAGetDir(SQA_DIR_PROJECT) &"Datapool\"
         dirXLS = dirOUT
         'TAB field separator for test tables exported from Excel
         separator = Chr$(9)
         status = ExportXLSToFiles( fileXLS   := tableName, _
                        dirXLS    := dirXLS, _
                        dirOUT    := dirOUT, _
                        delimiter := separator)
         if status<>0 then
            MsgBox "Operation isn't completed successfully! Please check that the file exists!"
         End If
      Else
         MsgBox "File is not an Excel table!"
      End If
   Else
         MsgBox "Please specify file name!"
   End If
```



Фигура 7. Диаграма на тестовия процес при внедряване на методология, базирана на данни

Функцията за преформатиране на картите на страниците в подходящ формат е подобна на тази за тестовите таблици. Тя се казва ExportXLS2INIFile тъй като картите на страниците се представят в Rational Robot чрез такъв тип файлове.

С помощта на инструмента се описва последователността от тестове, която ще бъде изпълнена. Когато тестовете са готови се подава команда на инструмента да прехвърли създадените файлове в работната директория на тестовата платформа

След това се извиква скрипта за изпълнение с параметър тестовата таблица, описваща тестовия цикъл. Скриптът за изпълнение инициализира лог файловете и извиква функцията на модула на най - високо ниво за съответната тестова таблица, описваща тестовия цикъл. Лог файлове се генерират за изпълнението на всеки тип тестова таблица. Логовете на стъпковите таблици съдържат информация за изпълнението на всяка стъпка. Логовете на тестовите таблици, описващи сюитите, съдържат обобщена информация за протичането на всяка от стъпковите таблици в нея. Логовете на таблиците на тестовите цикли дават информация за успеха на всяка сюита от тестове, участваща в него. Нека да разгледаме как е дефиниран всеки лог файл:

```
cycleName= GetField(cycleTableName,1,".")
'enable text logs ONLY in Suite and Cycle Driver
msgID =TEXTLOG_ENABLED
sTemp =
SQAGetDir(SQA_DIR_PROJECT)+"Datapool\Logs\SuiteLog_"+cycleName+"_"+dateStrFormat+".txt"
Kill sTemp
InitLogFacility msgId, SuiteLog, sTemp
sTemp =
SQAGetDir(SQA_DIR_PROJECT)+"Datapool\Logs\CycleLog_"+cycleName+"_"+dateStrFormat+".txt"
Kill sTemp
InitLogFacility msgId, CycleLog, sTemp
'enable text log, Robot log, AND Console in Step Driver
msgID = TEXTLOG_ENABLED   &  SQALOG_ENABLED & CONSOLE_ENABLED
sTemp =
SQAGetDir(SQA_DIR_PROJECT)+"Datapool\Logs\StepLog_"+cycleName+"_"+dateStrFormat+".txt"
Kill sTemp
InitLogFacility msgId, MainLog, sTemp
```

Лог файловете, които ще се създават при изпълнението на таблиците от високо ниво ще се записват в съответните текстови файлове. Изпълнението на стъпковите таблици ще се записва в текстов файл, в лог файла на инструмента и ще се вижда в конзолата на инструмента непосредствено след и по време на изпълнение на тестовете. Имената на лог файловете съдържат името на тестовия цикъл и времето на изпълнение, което позволява многократното изпълнение на тестовете да не презаписва извършените вече тестове. Освен това може да се проследяват резултатите от изпълнението на тестовете във времето. Изпълнението на тестовете от тестовия цикъл започва с извикването на модула от най- високо нито чрез процедурата CDCycleDriver на която предаваме като параметри името на тестовата таблица, описваща тестовия цикъл и логовете,които ще използва за записване на изпълнението на тестовете.

След приключване на обработката на тестовите таблици, лог файловете са записани в работната директория тестовата платформа. Инструментът предоставя команда, която прехвърля лог файловете от директорията на платформата в директорията на текущата версия. Лог файловете са част от дървото могат да бъдат разглеждани в редактора на инструмента.

Този процес се изпълнява при всяка итерация на процеса на създаване на продукта. При някои итерации част от стъпките могат да бъдат пропуснати. Например

може да няма необходимост от генериране на нови тестови таблици и карти на страниците, ако съответната итерация е свързана с оправяне на грешки, които не са засегнали интерфейсите и не са променили функционалността. При промяна на интерфейсите на приложението може да е достатъчно да бъде актуализирана единствено картата на страниците.

6.2 Изпълнение на тестов цикъл за модул "Управление на потребителите" на интернет базирана системата

Нека да разгледаме процеса на създаване и изпълнение на тестов цикъл за модул за управление на потребители на интернет базирано .Net приложение. Приложението, наречено AIS- Active & Information Screens , реализира система за работа с крайни потребители на компания, предоставяща услуги в мрежата за обслужване на стационарни телефони. Компанията има различни клиенти, които предоставят различно множество и разнообразие от услуги. В системата могат да се въвеждат и обработват проблеми и запитвания на крайните потребители, както и да бъдат различни желания на потребителите за промени на доставените услуги. Клиентите на компанията могат да използват или да не използват системата за въвеждане на искания и проблеми на крайните потребители. Ако клиентите не желаят да предоставят тази услуга на крайните потребители, компанията я осигурява. Затова потребителите на системата са най - общо 2 типа - на компанията и на определен клиент. За всеки от типовете можем да имаме 3 роли - администратор на системата, оператор и управител на операторите. Администраторът на системата на компанията може на управлява потребителите на системата и на компанията и на различните клиенти. Потребителите на системата могат да обслужват различни типове крайни потребители - частни лица, компании и фирми или и двата типа.
Нека да проследим процеса на тестване и създаването на тестовите таблици за добавяне на потребител на системата от администратор на компанията:

а) Създаване на стъпковите таблици, които ще участват в тестовете:

В Таблица 11 е описано регистрирането на потребител на системата. Първите 3 реда задават команди за платформата. На първия ред е определена на картата на страниците, която ще бъде използвана. Следващите редове задават команди за стартиране на браузъра и за изчакване на зареждане на страницата. Редовете с тип на записа "Тестова стъпка" последователно описват попълването на полето за име, парола и натискане на бутона за регистриране.

Таблица 11. Стъпкова таблица за регистриране на потребител на системата - LoginUser

	Project Name: AIS			
	Prerequisites:			
	Test Case: LoginUser			

;RT	WINDOW/ COMMAND	COMP/ ARG	ACTION/ ARG	ARG
C	SetApplicationMap	AISApplicationMap.MAP		
C	StartWebBrowser	http://localhost/AIS/Login.aspx		
C	WaitForWebPage	""	""	
T	LoginPage	LoginPage	Maximize	
T	LoginPage	UserName_Enter	SetTextCharacters	^UserName
T	LoginPage	Password_Enter	SetTextCharacters	^Password
T	LoginPage	Login_Action	Click	

Таблица 12 представя стъпковата таблица за навигация към страницата за добавяне на потребител. Отново се задава картата на страниците,която се използва. Тестовите стъпки задават последователността от връзки, през които трябва да мине потребителя. Командите преди тях указват на тестовата платформа да изчака зареждането на връзките, преди да се опита да извърши действието.

Таблица 12. Стъпкова таблица за навигация към страницата за добавяне на потребители - ToAddUser

	Project Name: AIS			
	Prerequisites:			
	Test Case: ToAddUser			
;RT	**WINDOW/ COMMAND**	**COMP/ ARG**	**ACTION/ ARG**	**ARG**
C	SetApplicationMap	AISApplicationMap.MAP		
C	WaitForGUI	HomePage	Administration_Link	"20"
T	HomePage	Administration_Link	Click	
C	WaitForGUI	HomePage	AddUser_Link	"20"
T	HomePage	AddUser_Link	Click	

На Таблица 13 е представена стъпкова таблица за добавяне на клиентски потребител. Първите два реда задават използваната карта на страниците и указват на тестовата платформа да изчака зареждането на формата за добавяне на потребител. След това се задават стойности на полетата, които трябва да бъдат попълнени. Стойностите на полетата са зададени с променливи, което позволява тестовата таблица да бъде използвана многократно с различни стойности на полетата.

Таблица 13. Стъпкова таблица за добавяне на клиентски потребител - AddClientUser

	Project Name: AIS			
	Prerequisites:			
	Test Case: AddClientUser			

:RT	WINDOW/ COMMAND	COMP/ ARG	ACTION/ ARG	ARG
C	SetApplicationMap	AISApplicationMap.MAP		
C	WaitForGUI	AddUserPage	AddUser_Form	"20"
T	AddUserPage	FirstName_Enter	SetTextValue	^NewUserFirstName
T	AddUserPage	LastName_Enter	SetTextValue	^NewUserLastName
T	AddUserPage	Telephone_Enter	SetTextValue	^NewUserTelephone
T	AddUserPage	Email_Enter	SetTextValue	^NewUserEmail
T	AddUserPage	Role_Enter	SetTextValue	^NewUserRole
T	AddUserPage	Client_Enter	SetTextValue	^NewUserClient
T	AddUserPage	CustomerType_Enter	SetTextValue	^NewUserCustomerType
T	AddUserPage	LoginName_Enter	SetTextValue	^NewUserUserName
T	AddUserPage	Password_Enter	SetTextValue	^NewUserPassword
T	AddUserPage	ConfirmPassword_Enter	SetTextValue	^NewUserConfPassword
T	AddUserPage	ActiveUser_Check	Check	
T	AddUserPage	Save_Action	Click	

Таблица 14 описва стъпковата таблица за верифициране на резултата от добавянето на потребител. В таблицата е описано попълването на полетата на формата за търсене на потребител с данните на потребителя. Добавянето на потребителя ще е успешно, ако търсенето върне резултатна таблица с потребителя. В противен случай такава таблица няма да се появи.

Таблица 14. Стъпкова таблица за верифициране на резултата от добавянето на потребител - VerifySuccessfullAddUser

	Project Name: AIS			
	Prerequisites:			
	Test Case: VerifySuccessfullAddUser			
:RT	**WINDOW/ COMMAND**	**COMP/ ARG**	**ACTION/ ARG**	**ARG**
C	SetApplicationMap	AISApplicationMap.MAP		
C	WaitForGUI	SearchUsersPage	SearchUsers_Form	"20"
T	SearchUsersPage	FirstName_Enter	SetTextValue	^SearchUserFirstName
T	AddUserPage	LastName_Enter	SetTextValue	^SearchUserLastName
T	AddUserPage	Telephone_Enter	SetTextValue	^SearchUserTelephone
T	AddUserPage	LoginName_Enter	SetTextValue	^SearchUserUserName
T	AddUserPage	Email_Enter	SetTextValue	^SearchUserEmail
T	AddUserPage	Role_Enter	SetTextValue	^SearchUserRole
T	AddUserPage	Client_Enter	SetTextValue	^SearchUserClient
T	AddUserPage	Search_Action	Click	
C	WaitForGUI	SearchUsersResultPage	SearchUsersResultGrid	"30"

Таблица 15 описва стъпкова таблица за излизане от системата. Излизането от системата се осъществява чрез натискане на съответната връзка.

Таблица 15. Стъпкова таблица за излизане от системата - LogOut

	Project Name: AIS		
	Prerequisites:		
	Test Case: LogOut		
;RT	**WINDOW/ COMMAND**	**COMP/ ARG**	**ACTION/ ARG**
C	SetApplicationMap	AISApplicationMap.MAP	
T	HomePage	Logout_Link	Click

b) Втората стъпка от процеса е създаване на картата на страниците. В картата на страниците са описани страниците, през които минава теста и компонентите, които използва. Картата на страниците включва описание на компонентите на страницата за регистриране на потребителна системата, страницата, съдържаща главното меню, страницата за добавяне на потребител и страницата за търсене на потребител. Таблица 16 представя картата на страниците. В Приложение 2, 3, 4 и 5 може да се види интерфейсът на описаните страници.

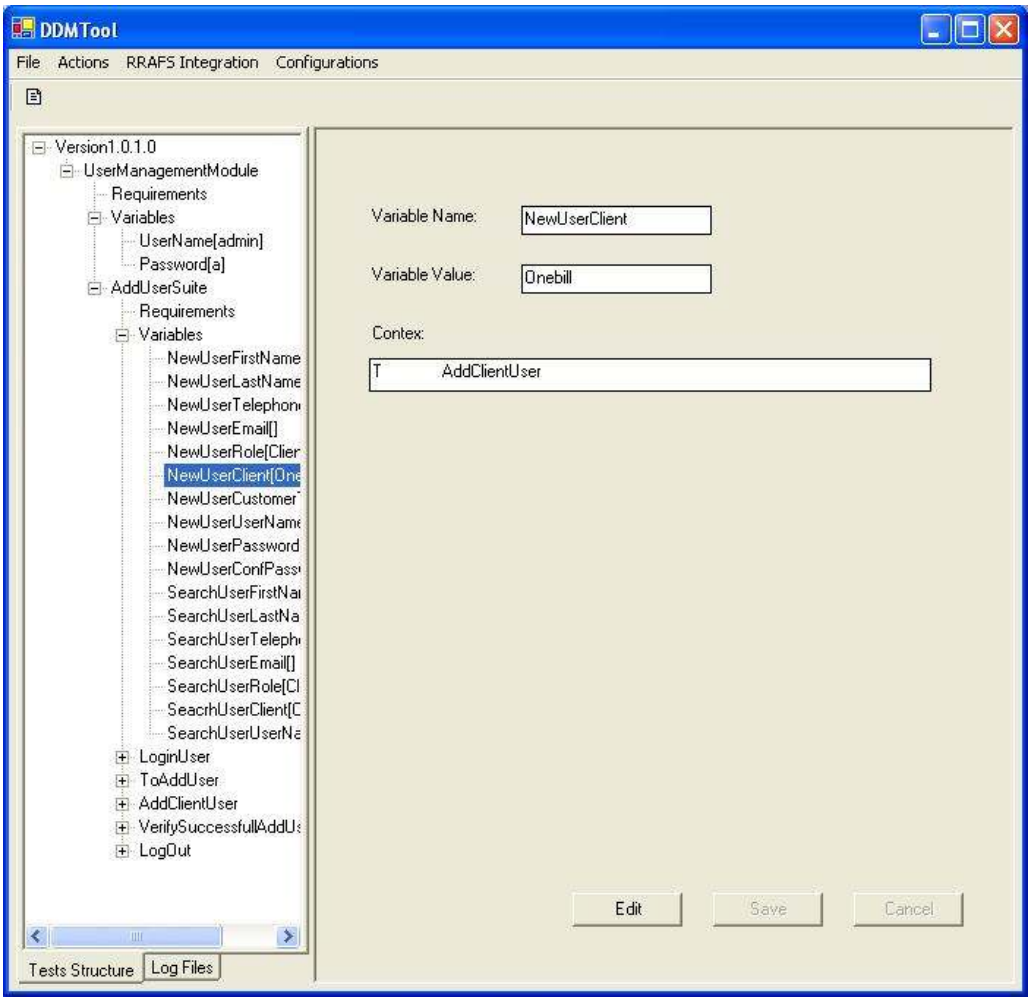
Таблица 16. Карта на страниците

[LoginPage]		
LoginPage	"Type=Window;Caption={A I S*}"	Window
UserName_Enter	"Type=EditBox;Name=txtLogin"	EditBox
Password_Enter	"Type=EditBox;Name=txtPassword"	EditBox
Login_Action	"Type=PushButton;Name=btnLogin"	PushButton
[HomePage]		
HomePage	"Type=Window;Caption={A I S*}"	
left	"Type=HTMLFrame;Name=left"	
;children of "left"		
leftDOC	"\;Type=Window;Caption=AIS- Microsoft Internet Explorer;\;Type=HTMLFrame; Name=left;\;Type=HTMLDocument;Index=1"	
Administration_Link	"Type=HTMLFrame;HTMLId=left;\;Type= HTMLLink;HTMLText=Administration"	
AddUser_Link	"Type=HTMLFrame;HTMLId=left;\;Type= HTMLLink;HTMLText=Add User"	
CurrentUserLogin_ TableCell	"Type=HTMLFrame;HTMLId=left;\;Type= HTMLTableCell;HTMLText=admin"	
[AddUserPage]		
AddUserPage	"Type=Window;Caption={*}"	Window
AddUserPageDOC	"Type=HTMLDocument;Index=1"	HTMLDocument
content	"Type=HTMLFrame;Name=content"	HTMLFrame
;children of "content"		
FirstName_Enter	"Type=HTMLFrame;HTMLId=content;\;Type= EditBox;Name=txtFirstName"	EditBox

LastName_Enter	"Type=HTMLFrame;HTMLId=content;\;Type= EditBox;Name=txtLastName"	EditBox
Telephone_Enter	"Type=HTMLFrame;HTMLId=content;\;Type= EditBox;Name=txtTelephone"	EditBox
Email_Enter	"Type=HTMLFrame;HTMLId=content;\;Type= EditBox;Name=txtEmail"	EditBox
Role_Enter	"Type=HTMLFrame;HTMLId=content;\;Type= ComboBox;Name=ddlRoles"	ComboBox
Client_Enter	"Type=HTMLFrame;HTMLId=content;\;Type= ComboBox;Name=ddlClients"	ComboBox
CustomerType_Enter	"Type=HTMLFrame;HTMLId=content;\;Type= ComboBox;Name=ddlCustomerType"	ComboBox
LoginName_Enter	"Type=HTMLFrame;HTMLId=content;\;Type= EditBox;Name=txtLoginName"	EditBox
Password_Enter	"Type=HTMLFrame;HTMLId=content;\;Type= EditBox;Name=txtPassword"	EditBox
ConfirmPassword_Enter	"Type=HTMLFrame;HTMLId=content;\;Type= EditBox;Name=txtConfirmPass"	EditBox
ActiveUser_Check	"Type=HTMLFrame;HTMLId=content;\;Type= CheckBox;Name=chkActive"	CheckBox
Save_Action	"Type=HTMLFrame;HTMLId=content;\;Type= PushButton;Name=btnSave"	PushButton
[SearchResultOnePage]		
SearchResultOnePage	"Type=Window;Caption={A I S*}"	Window
SearchResultOnePageDOC	"Type=HTMLDocument;Index=1"	HTMLDocument
content	"Type=HTMLFrame;Name=content"	HTMLFrame
;children of "content"		
SearchUsers_Form	"Type=HTMLFrame;HTMLId=content;\;Type= HTML;HTMLId=Form1"	HTML
FirstName_Enter	"Type=HTMLFrame;HTMLId=content;\;Type= EditBox;Name=txtFirstName"	EditBox
LoginName_Enter	"Type=HTMLFrame;HTMLId=content;\;Type= EditBox;Name=txtLoginName"	EditBox
Email_Enter	"Type=HTMLFrame;HTMLId=content;\;Type= EditBox;Name=txtEmail"	EditBox
Role_Enter	"Type=HTMLFrame;HTMLId=content;\;Type= ComboBox;Name=ddlRoles"	ComboBox
Client_Enter	"Type=HTMLFrame;HTMLId=content;\;Type= ComboBox;Name=ddlClients"	ComboBox
Telephone_Enter	"Type=HTMLFrame;HTMLId=content;\;Type= EditBox;Name=txtTelephone"	EditBox
Search_Action	"Type=HTMLFrame;HTMLId=content;\;Type= PushButton;Name=btnSearch"	PushButton
SearchUsersResult_Table	"Type=HTMLFrame;HTMLId=content;\;Type= HTMLTable;HTMLId=dgUsers"	HTMLTable

c) След като се подготвят тестовите таблици и картата /ите/ на страниците се генерират съответните файлове, във формата, който се използва от тестовата платформа и инструментите. Използваме скрипта за преформатиране на таблиците и създаване на тестовите файлове за всяка една от тестовите таблици и за картата на страниците.

d) Следващата стъпка е създаване на тестовия цикъл в инструмента. В инструмента създаваме нов проект за версия 1.0.1.0. Създаваме нов тестов цикъл за модула. Добавяме нова сюита към тестовия цикъл. В новата сюита се добавят теста за регистриране на потребител на системата с администраторски права, навигация до страницата за добавяне на потребител, добавянето на потребителя, верификация дали добавянето е било успешно и излизане на администратора от системата. Създават се променливите и се задават стойности, които ще се използат при тестовете. На Фигура 8 е представен интерфейсът на инструмента за автоматично генериране на тестови табици. Вижда се създадената йерархия от тестове за тестване на добавянето на потребител на системата.



Фигура 8. Йерархия на тестовете,създадени в инструмента

Новите тестови таблици се генерират автоматично. Файловете на използваните стъпкови таблици се копират в директорията на проекта. В Таблици 17 и 18 са представени съответно генерирания файл, описващ тестова сюита и файла на тестовия цикъл. Файловете се генерират в текстов файлов формат. В таблиците 17 и 18 са

представени преформатираните файлове в табличен вид. Таблица 17 представя таблицата от второ ниво, която описва тестовата сюита за добавяне на клиентски потребител на системата. Тестовата сюита се състои от регистриране на потребител на системата с администраторски права, навигация до страницата за добавяне на потребител, добавянето на потребителя, верификация дали добавянето е било успешно и излизане на администратора от системата. Зададени са и стойности на променливите, които се използват в стъпковите таблици. Таблица 18 представя тестовата таблица от най- високо ниво. В нея е описана само една тестова сюита - AddClientUserSuite .

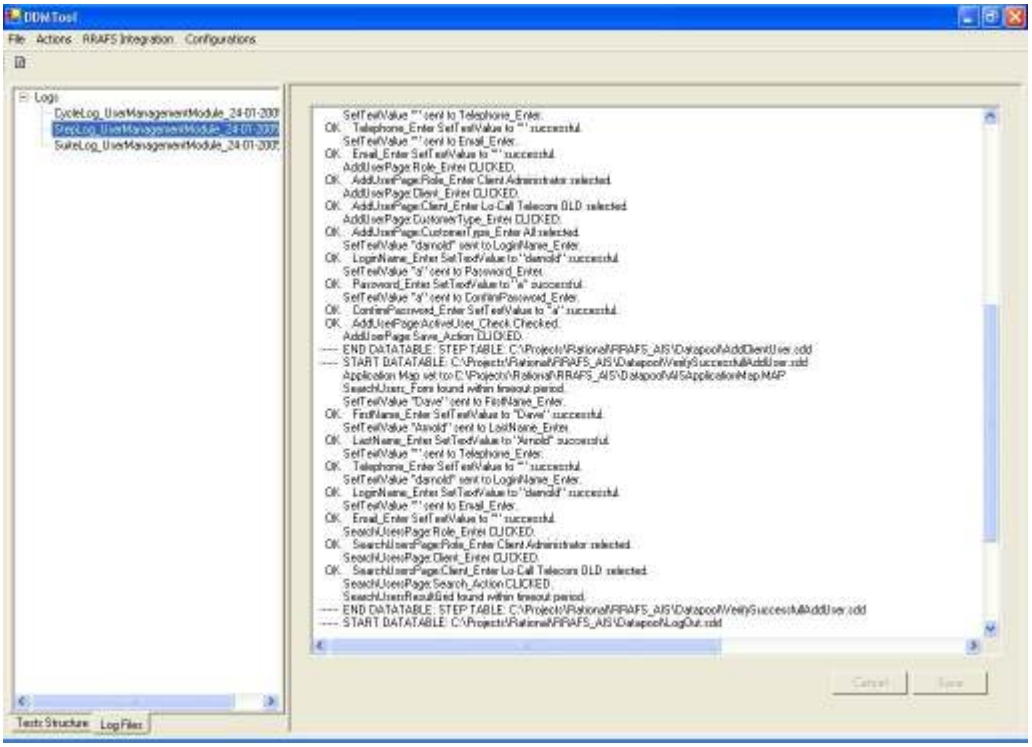
Таблица 17. Таблица от второ ниво, представяща тестова сюита за добавяне на клиентски потребител на системата

	Project Name:			
	Test Server:			
	OS:			
	Browser:			
	QA Engineer:			
	Date:			
	Prerequisites:			
	1. Client Onebill is client of Comms Factory and has own customer support			
	2. There is no user of the system with user name "darnold"			
	Test Suite: AddClientUserSuite			
;RT	STEPS/ COMMAND	SEP/ ARG	ARG	ARG
T	LoginUser			
T	ToAddUser			
T	AddClientUser		^NewUserFirstName="Dave"	^NewUserLastName="Arnold"
T	VerifySuccessfullAddUser		^SearchUserFirstName="Dave"	^SearchUserLastName="Arnold"
T	LogOut			

Таблица 18. Таблица от най- високо ниво, представяща тестов цикъл

	Project Name:							
	Test Server:							
	OS:							
	Browser:							
	QA Engineer:							
	Date:							
	Prerequisites:							
	1. User name "admin" and password "a" is Comms Factory Administrator User of the AIS System							
	Test Cycle: UserManagementModule							
;RT	STEPS/ COMMAND	SEP/ ARG	ARG	ARG	ARG	ARG	ARG	
T	AddUserSuite		^UserName="admin"	^Password="a"				

e) Следващата стъпка е прехвърляне на създадените тестове файлове за изпълнение от тестовата платформа чрез командата на инструмента Export Tests For Execution. След като е описан тестовия цикъл чрез йерархията от тестове, създадените тестови таблици автоматично се прехвърлят за изпълнение в работната директория на тестовата платформа.
f) След като тестовите таблици са преместени в работната директория на тестовата платформа се изпълнява тестовия цикъл. От инструмента за автоматизирано тестване Rational Robot се стартира скрипта за изпълнение с параметър UserManagementModule.cdd.
g) Последната стъпка е разглеждане и анализиране на лог файловете. След изпълнението на тестовия цикъл за модул "Управление на потребителите" на интернет базирана системата създадените лог файлове за стъпковите таблици, таблиците, описващи тестовите сюити и цикли, се копират в директорията на проекта чрез командата Import Log Files. Лог файловете могат да бъдат разглеждани и анализирани в редактора на инструмента. На Фигура 9 е показано генерираните от тестовата платформа лог файлове. На фигурата е показан лог файлът на тестовия цикъл



Фигура 9. Представяне на лог файловете в инструмента

7. Заключение

В настоящата дипломна работа е предложено решение за внедряване на методология, базирана на данни, за автоматизирано функционално тестване на интернет приложения. Разгледани са предимствата, които предлага методологията за повишаване на качеството на софтуерните продукти. Решението е реализирано с помощта на инструмента за автоматизирано тестване Rational Robot и надграждащата го платформа за тестване, базирано на данни, Rational Robot Automation Framework Support. За автоматизираното създаване на тестовете и тяхното съхранение и организация е разработен инструмент. Инструментът е интегриран с тестовата платформа за изпълнение на тестовете и анализиране на резултатите.

Могат да се намерят множество възможности за разширение на разработения инструмент. В бъдещи разработки може да се реализира функционалност за валидация на тестовите таблици спрямо използваните карти на страниците. По този начин ще се проверява за липсващи или грешни дефиниции на елементи. Също така полезно усъвършенстване ще бъде разширяване възможностите на редактора за работа с файловете, представящи тестовите таблици, както и поддръжка на работа с електронните таблици на MS Excel. Инструментът може да бъде разширен с допълнителни функции за представяне и анализиране на лог файловете, създадени от тестовата платформа. Осигурената от платформата поддръжка на XML предоставя възможност за интерпретация и анализ на резултатите от извършените тестове.

Решението може да бъде разширено и интегрирано в процес на тестване, обхващащ целия набор от тестови продукти, разработвани от IBM. Rational TestManager позволява интеграцията с различни инструменти за създаване на тестове, чиито резултати могат да бъдат анализирани чрез средствата, които предоставя продукта - лог файлове и рапорти. Също така в тестовия процес, базиран на разглежданата методология, могат да бъдат включени продуктите за следене на изискванията, направените промени и изследване на степента на покритие на приложението от тестовете. Rational TestManager предоставя възможности за създаване на тест планове.

8. Речник

8.1 Речник на използваните абревиатури

COM - Component Object Model
XML - Extensible Markup Language
SOAP - Simple Object Access Protocol
RRAFS - Rational Robot Automation Framework Support
AIS- Active & Information Screens

8.2 Речник на използваните термини

Анализиране на степента на покритие на приложението от тестове (Coverage analysis) – Определя коя част от тестваното приложение е покрито от тестове и коя не е. Покритието се определя чрез изследване на кода, който е бил изпълнен.

Гъвкави модели на процеси (Agile Process Models) - Гъвкавите модели на процеси се придържат към правилата за гъвкаво разработване на софтуер

Гъвкаво Разработване на софтуер (Agile Software Development) - Гъвкаво Разработване на софтуер е методология за разработване на софтуер, която се основава принципи и правила даващи възможност за лесно адаптиране и реагиране на промените при създаване на софтуер.

Методология за автоматизирано функционално тестване базирана на данни и ключови думи (Data-driven Methodology, Keyword- driven Methodology, Test Plan Driven Methodology, Table Driven Methodology) - Методологията, базирана на данни и ключови думи, се характеризира с параметризиране на входните и изходните данни, компонентите и интерфейсите. Тестовете се създават под формата на таблици, които се интерпретират от тестовата платформа.

Методология за автоматизирано функционално тестване Записване и изпълнение на скриптове (Record/Playback или Capture/Replay) – Методологията се състои от автоматизираното генериране на изпълним скрипт, който описва последователност от действия на потребителя на тестваното приложение. При изпълнение на създадените скриптове се пресъздават записаните действия и се проверява поведението на системата.

Методология за автоматизирано функционално тестване Функционална декомпозиция (Functional Decomposition) - Методологията Функционална декомпозиция използва йерархична структура и модулен дизайн при създаването на тестовете.Всеки тест се разделя на части, които се реализират в различни скриптове в зависимост от тяхната функционалност.

Методология за автоматизирано функционално тестване Параметризиране на входни и изходни данни (Data – driven Scripts) - При прилагането на методологията записаните скриптове се модифицират да могат да работят с променливи входни и изходни данни. Данните могат да се съхраняват във файлове или бази данни, от където се извличат и се ползват в тестовете.

Интеграционно тестване (Integration testing) - тестване на системата след интеграцията на различни модули за да се проверят грешки в интерфейса и взаимодействията между компонентите.

Описателни модели на процеси (Prescriptive Process Models) - Описателните модели на процеси дефинират строго множество и последователност от активности за създаване на софтуер.

Очаквани резултати(Expected results) - състояние на приложението в даден момент от изпълнението на тестовия сценарии

Платформа за автоматизирано тестване (Test Automation Framework) - множество от предположения, концепции и практики, които осигуряват поддръжка за автоматизирано тестване.

Пространство от имена (Namespace) – Логическа организация на класове (в случая), която подпомага разрешаването на конфликтите на имена.

Регресионно тестване(Regression testing) - Повторно тестване на програмата, след нейното модифициране, за да се провери, че не са настъпили грешки в модулите, които не са претърпявали промяна.

Системно тестване (System Testing) - тестване на цялата система за да се провери дали удовлетворява изискванията към нея. На това ниво се извършват тестове на функционалността, на сигурността и на производителността на системата.

Скрипт (Script) - програма, написана на езика на инструмента за автоматизирано тестване, която автоматизира изпълнението на тестов сценарий или сюита.

Тестване тип "черна кутия" (Black box testing) - Тестване, което се извършва чрез определяне на множество от входни данни и съответните очаквани резултати, без да се използва кода и архитектурата на програмата.

Тестване на програмни единици (Unit testing) - Тестване на най - малките компоненти на продукта - клас, процедура.

Тестване на програмни модули (Module testing) – Тестването на програмни модули се прилага при големи и сложни продукти, съставени от няколко модула

Тестване на производителността (Performance test) - Тестване, което се извършва за да се провери доколко системата отговаря на определени изисквания за производителност.

Тестване при доставяне на системата (Acceptance testing) - На това ниво на тестване се проверява как системата се държи в реална среда. Тестовете се извършват от потребителите на системата.

Тестов сценарий (Test case) – Тестовият сценарий дефинира множество от входни данни, предпоставки и очаквани резултати, които да верифицират определено изискване

Тестов цикъл (Test cycle) - Описва дейностите, които трябва да се извършат за изпълнението на определена тестова активност. Обикновено приложението преминава през няколко тестови цикъла по време на разработката.

Тестова сюита (Test suite) – Тестовата сюита представя подредено множество от тестови сценарии. Тестовите сюити въвеждат йерархична структура на създаване на тестове

Тестови данни (Input data) - Входни данни за тестовия сценарии

9. Литература

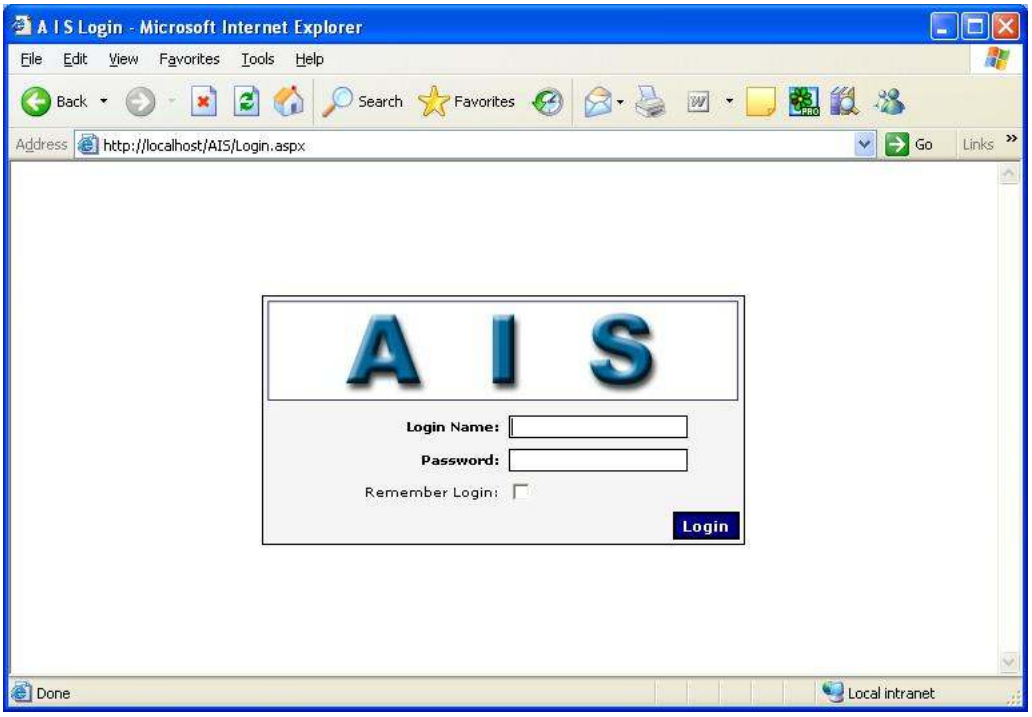
1. Titterington Graham, Woods Eric (1999) *Software Testing Tools*
2. Presman Roger S. (Sixth Edition, 2005) Software Engineering: A Practitioner's Approach. McGraw – Hill Science
3. http://www-306.ibm.com/software/rational/
4. http://www.microsoft.com/technet/itsolutions/msit/dotnetet.mspx- White Paper
5. http://safsdev.sourceforge.net/Default.htm
6. Rational Robot User Manual, Version 2003.06.00
7. www.agilealliance.org
8. Zambelich Keith (1998), *Totally Data-Driven Automated Testing*, A white paper, http://www.sqa-test.com/w_paper1.html
9. Nagle Carl J. (2000) , *Test Automation Frameworks,* http://safsdev.sourceforge.net/FRAMESDataDrivenTestAutomationFrameworks.htm
10. Faught Danny (2002), *Keyword-Driven Testing,* http://www.stickyminds.com/testing.asp
11. Kelly Michael, *Frameworks for Test Automation,* http://www.sqatester.com/MichaelKelly/FrameworksforTestAutomation.htm
12. Hu Jirong (2004), *Getting started with Rational Robot automation framework support,* http://www-106.ibm.com/developerworks/rational/library/5028.html
13. Nagle, C. *Data Driven Test Automation: For Rational Robot V2000* 1999-2000, http://safsdev.sourceforge.net/Default.htm
14. http://www.cbueche.de/FRM_DOC/webhelp_eng/index.htm
15. http://fitnesse.org/FitNesse.OneMinuteDescription
16. http://www.worksoft.com/ContentDisplay/G3/G3L.asp
17. http://www.sdtcorp.com/unifiedtestpro.html
18. http://www.logicacmg.com/testing_quality/testframe.asp
19. http://www.logigear.com/products/testarchitect/

10. Приложения

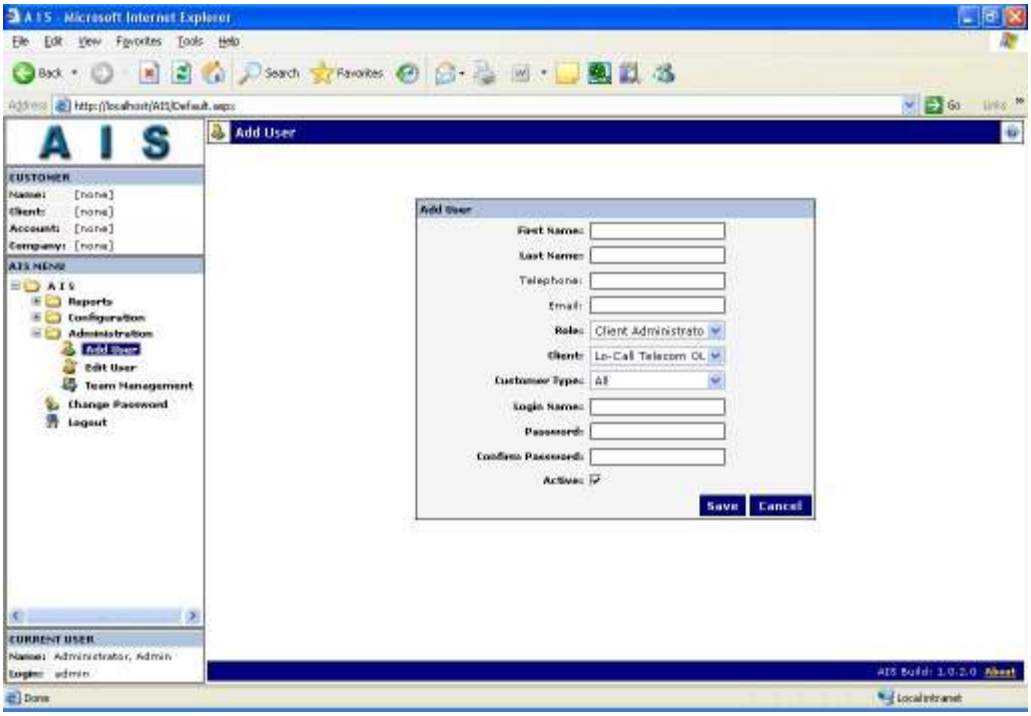
10.1 Таблица на асоциациите между използваните от Rational Robot обекти и HTML елементите

Обекти на Rational Robot	HTML елементи
PushButton	<INPUT type=Submit> <INPUT type=Reset> <INPUT type=Button> <BUTTON>
CheckBox	<INPUT type=Checkbox>
RadioButton	<INPUT type=Radio>
ComboBox	<SELECT size=1> <OPTION> . . . </SELECT>
ListBox	<SELECT size=>*n*> <OPTION> . . . </SELECT>
EditBox	<INPUT type=Text> <INPUT type=TextArea>
HTMLLink	<A> . . . </A>
HTMLImage	<IMG>
HTMLDocument	Целия текст между <BODY> и </BODY>
HTMLTable	Целия текст между <TABLE> и </TABLE>
HTMLActiveX	<OBJECT>
HTML	Всичнки останали маркери

10.2 Интерфейс на страницата за регистриране на потребители в системата AIS



10.3 Интерфейс на страницата за добавяне на потребител на системата AIS



10.4 Интерфейс на страницата за търсене на потребител на системата AIS