Пловдивски университет „Паисий Хилендарски”
Факултет по математика и информатика
Катедра „Компютърна информатика”

Денислав Иванов Лефтеров

Софтуерна рамка за тестване на уеб приложения

А В Т О Р Е Ф Е Р А Т
на дисертационен труд за присъждане
на образователна и научна степен „доктор“

Област на висше образование:
4. Природни науки, математика и информатика
Професионално направление: **4.6. Информатика и компютърни науки**
Докторска програма **Информатика**

Научен ръководител:
доц. д-р Светослав Христосов Енков

Пловдив, 2022 г.

Пловдивски университет „Паисий Хилендарски”
Факултет по математика и информатика
Катедра „Компютърна информатика”

Денислав Иванов Лефтеров

Софтуерна рамка за тестване на уеб приложения

А В Т О Р Е Ф Е Р А Т
на дисертационен труд за присъждане
на образователна и научна степен „доктор“

Област на висше образование:
4. Природни науки, математика и информатика
Професионално направление: **4.6. Информатика и компютърни науки**
Докторска програма **Информатика**

Научен ръководител:
доц. д-р Светослав Христосов Енков

Пловдив, 2022 г.

Дисертационният труд е обсъден и насочен за защита от катедрения съвет на катедра „Компютърна информатика“ на Факултета по математика и информатика при Пловдивския университет „Паисий Хилендарски“ на 01.04.2022 г.

Основният текст на изследването съдържа общо 150 страници. Списъкът на използваната литература в дисертационния труд съдържа 67 заглавия, от които на кирилица 3, а на латиница 64, включително и 21 заглавия в уеб-базирани източници.

Основните резултати на изследването са докладвани на катедрени и докторантски семинари, на национални и международни научни форуми.

Списъкът на авторските публикации се състои от 5 излезли от печат заглавия. От тях една публикация е на български език, а останалите 4 са на английски език.

Материалите по защитата са на разположение на интересуващите се в библиотеката на Факултета по математика и информатика на ПУ „Паисий Хилендарски“, бул. „България“ № 236, Пловдив.

Защитата на дисертационния труд ще се състои на 24.06.2022 г. от 12:30 ч. в Заседателната зала на ФМИ (Нова сграда, бул. „България“ №236). (ще бъде обявено в сайта на ПУ/ФМИ)

Автор: **Денислав Иванов Лефтеров**

Заглавие: **Софтуерна рамка за тестване на уеб приложения**

Тираж 50 бр.

Пловдив, 2022 г.

Съдържание

СЪКРАЩЕНИЯ

Съкращение		Наименование на английски
BDD	–	Behavior Driven Development
CI	–	Continuous Integration
DOM	–	Document Object Model
GUI	–	Graphical User Interface
ISTQB	–	International Software Testing Qualifications Board
JUnit	–	Java Unit Framework
POM	–	Page Object Model
POP	–	Page Object Pattern

УВОД

Актуалност на проблема

Последните две десетилетия с напредването на технологиите при разработката на софтуер, фазата на тестване е една от основополагащите раздели, които проследяват реализирането на продукта. Действието по разработката на стабилни и независими тестове налага изграждането на компонентна система от многослойни тестови сценарии и алгоритми за тестване, както в процеса на всяка фаза на разработвания продукт, така и на интегрираната система. В 21 век софтуерното инженерство се адаптира към промените в глобалния процес на разработка на софтуер, за да се облекчат високите разходи, високата сложност, времето за пускане на разработвания продукт на пазара, трудностите при поддръжка и увеличаване на удовлетвореността на крайните потребители.

Цели и задачи на дисертационния труд

Основната цел на дисертационното изследване е създаването на цялостна автоматизирана рамка за тестване на уеб приложения, която да удовлетвори нуждите на потребителите в процесите на тестване и осигуряване качеството на софтуера. За реализиране на поставената цел трябва да се изследват следните етапи в тестовия подход: функционално, нефункционално, приемно, компонентно и интеграционно тестване. По време на дисертационното изследване ще се проследи проектирането на една завършена тестова архитектура, върху която ще се изгради софтуерна рамка за автоматизация на тестови подходи.

За да се постигне основната цел, е нужно да бъдат решени следните ***задачи***:

Задача 1. Създаване на теоретичен програмен модел на бъдещата разработка, както и базови класове и методи за валидиране, верифициране и изпълнение на действия от общ характер.

Задача 2. Създаване на многослойна архитектура и разработка на софтуерна тестова рамка за функционални тестове и шаблон за абстракция на функционални автоматизирани скриптове.

Задача 3. Създаване на цялостна интеграция след вече разработената архитектурна софтуерна работна рамка от Задача №2, за лесна преносимост, съвместимост и използваемост при вграждането на софтуерната тестова рамка към нови уеб приложения.

Постигането на основната цел предполага провеждането на следните ***етапи***:

1. Избор на архитектура и подготовка за създаване на автоматизирана софтуерна рамка за тестване на потребителски интерфейс. Включват се следните важни аспекти: избор на платформа, съвместими технологии, популярен език за програмиране и бъдещо развитие на тестовата рамка. Също така и набор от инструменти симулиращи поведението на потребител, който извършва действия базирани на събития в рамките на тестваната среда, чрез уеб или мобилен браузър. В рамките на дисертационната разработка ще бъде създаден обособен модел, който валидира набор от често използвани функционалности на автоматизиран тестов подход.

2. Реализация на логически функционален инструмент. Този подход разчита на изграждането на набор от модули и правила в основата на общи действия под формата на методи, така че при генерирането на функционални тестове да се различава бизнес логиката от програмната презентация. Създаденият набор от програмни методи ще бъде разширен постепенно с развитието на проекта, така че да се създаде специфична логика насочена към тестването на конкретни сценарии, независими от вида на уеб приложението. Подобен подход намира приложение, когато е необходимо да се тестват, различни портали, които са позиционирани в рамките на едно голямо по сложност приложение.

3. Имплементиране на шаблона POP (Page Object Pattern) представлява набор от логически правила. Чрез предоставения архитектурен подход организира разделението на логиката от сглобяването на тестовите действия в рамките на отделни функционални слоеве, групирани на базата на различни визуални компоненти на базата, на които са дефинирани. Този шаблон групира функционалните методи на тестваните обекти в рамките на тестовите екрани.

4. Върху вече създадени тестови реализации и полагането на рамковите основи, ще създадем една обща многослойна рамкова архитектура. Базовата програмна завършеност води до провеждане на изследване на реализираната многослойно тестова архитектура върху реално Интернет приложение, разработвано, чрез комплексни микросървиси. Използване на гъвкавата методология на Agile - Scrum и анализиране на постигнатите резултати. Положителни и отрицателни страни, както и акцент за бъдещо развитие на тестовата архитектура.

5. Интегриране на автоматизираната рамка с облачно-базиран инструмент в помощ на тестовата автоматизация. Реализиране на програмна конфигурация, която да служи като медиатор, между локалната програмна рамка за автоматизация и външния облачно базиран инструмент. Такъв пример е BrowserStack, като този инструмент ще спомогне за тестването на приложенията върху различни видове браузъри, версии, операционни системи, мобилни устройства и други платформени зависимости, които е трудно физически да бъдат реализирани самостоятелно.

Контекст на разработката

Настоящият научен труд демонстрира теоретичните основи на тестовия механизъм и практическите подходи за реализация на софтуерна тестова рамка за тестване на уеб приложения, съобразена с всички стандарти, добри практики и тенденциите използвани на пазара.

Основната сфера на приложение, която ще бъде проследена в настоящия научен труд е функционалното тестване на уеб приложения. В допълнение ще бъдат разгледани, останалите видове тестови подходи, добрите практики при мобилно тестване и интеграцията на автоматизираната рамка с външни приложения и нейното конфигуриране.

Структура на дисертационния труд

Представеният дисертационен труд се състои от: увод, четири глави и заключение, списък на използваната литература, списък на авторските публикации по темата, декларация за оригиналност и приложения. Съдържа разработена автоматизирана рамка и инструменти за осигуряване на качеството на софтуера.

Глава първа – Теоретични основи на софтуерното тестване, представя описание на основните теоретични подходи и концепции на софтуерното тестване. Съдържа се информация за различните типове тестове, като се извършва подробна систематизация и се демонстрира тяхната цел и употреба. Проследени са предимствата и недостатъците на всеки един от тестовите процеси, направено е сравнение на комерсиалните инструменти и шаблони за тестване, които се използват в софтуерната разработка. Разгледаните теоретични похвати се използват като основа за бъдещата разработка на приложната автоматизирана тестова рамка. Основите заложени в първа глава целят да се даде представа за необходимите тестови механизми, които трябва да бъдат добавени в разработката на работната рамка. Използвани и спазени са стандартите на ISTQB – конвенционален стандарт използващ се при тестовите подходи.

Глава втора – **Разработка на програмна логика за валидиране на тестове**, се проследяват фазите, през които минава един тестов механизъм, както и подходите за разработката му. Също така се разглеждат основните тестови случаи, които могат да бъдат отнесени към общите задължителни действия, които се прилагат при тестването на повечето уеб приложения. Акцентът на този етап се концентрира върху стратегия и реализацията на програмен подход наречен библиотека за удостоверяване на тестове съдържаща валидиращи механизми, подход за изпълнение на тестове, както и различен набор от графични визуализационни доклади. В Глава втора се полагат основите на бъдещата архитектура и реализацията на последващите механизми поставени в Задача №1.

Глава трета – **Реализация на функционални тестови механизми**, се разглеждат подходи за програмна реализация на функционални тестове като Selenium WebDriver взаимодействащ с браузъра, включвайки в себе си: JUnit, или TestNG - Java библиотеки. Също така се представя цялостна настройка върху облачно-базиран инструмент BrowserStack. Като се демонстрират отделни примери по какъв начин може да изнесем логиката на вече разписаните автоматизирани тестове. Локално върху външна облачна система и изпълняването им на различни браузъри, платформи и операционни системи или други зависимости. Описана е работата с автоматизиран инструмент Jenkins, използван за продължителна интеграция при цялостно функционално тестване като: неговата употреба, зависимости, надстройка и ползи от неговото използване.

Глава четвърта – **Софтуерна рамка за тестване на уеб приложения**, тази глава е продължение на реализирането на процеса по фрагментното отделяне на функционални тестове, чрез допълването на работната рамка върху създадения архитектурен слой, дефиниран и разработен в Глава 1 и Глава 2 на научния труд. Върху вече създадени тестови реализации и полагането на рамковите основи, направихме една обща многослойна рамкова архитектура. Реализира се разделение на функционалните от тестовите подходи чрез шаблона POP. Подробно е демонстрирана архитектурната имплементация на многослойната тестова рамка. Предоставени са примери, визуализиращи използването на новосъздадената структура. Допълва се употребата и конфигурирането на инструмента за продължителна интеграция Jenkins с цел избягване на чести неуспехи на тестове при изпълнението. Глава 4 представя завършен софтуерен инструмент, който е необходим за нуждите на една организация, която цели пълноценно да внедри цялостното тестване на продуктите си и да осигури повишение на качеството на разработваните приложения.

В **Заключението** са систематизирани получените резултати и са изброени научно-приложните и приложните приноси на дисертационния труд. Очертани са насоките за бъдещо развитие на рамката.

Основният текст на изследването съдържа общо 150 страници. Списъкът на използваната литература в дисертационния труд съдържа 64 заглавия, от които на кирилица 3, а на латиница 59, включително и 21 заглавия в уеб-базирани източници.

Основните резултати на изследването са докладвани на катедрени и докторантски семинари, на национални и международни научни форуми.

Списъкът на авторските публикации се състои от 5 излезли от печат заглавия. От тях една публикация е на български език, а останалите 4 са на английски език.

Глава 1 – Теоретични основи на софтуерното тестване

Подходът при тестването на един софтуерен продукт представлява механизъм по изследване на различията между наличните и очакваните резултати, на които трябва да отговаря една система. Софтуерното тестване е подход, който се прилага по време на целия процес по създаване на продукта, както и проследяването след приключването на разработката.

1.1. Систематизация на тестовите подходи

Налични са два основни принципа, при изпълняване на тестове от гледна точка на извършителя. Те биват ръчно и автоматизирано тестване. Двата подхода притежават своите ползи и негативи, като намират голямо приложение, в различни етапи от софтуерната разработка.

1.2. Ръчно тестване

Подходът ръчно тестване представлява процес на проверка на софтуера за дефекти или грешки. За целта се изисква специалист – софтуерен тестер, който да симулира ролята на краен потребител, при което се използват повечето функции на приложението, за да се валидира правилното поведение.

За да гарантира пълнотата на тестването, тестерът често следва писмен план за тестване, който го води през набор от множество тестови случаи. [Craig, David et al.‘02]

1.3. Автоматизирано тестване

Подходът за автоматизирано тестване включва използването на специален софтуер за контрол на изпълнението на тестовете, сравнение на действителните резултати с прогнозирани, определяне и прилагане на предпоставките за тестване и други видове тестови контроли и тестови функции.

Този подход добавя автоматизиране на ръчния процес на тестване, който вече е наличен, като се използва агрегиране на наличния тестов подход и допълнението му в правила от програмен код. [Dustin et al.‘09]

1.4. Тестови типове

Фундаментално, софтуерното тестване може да бъде разделено на два основни подхода в хода на същинското тестването, които ще бъдат разгледани в тази част на разработката и върху които ще се създаде теоретичния модел на базата на който ще се разработи софтуерната тестова рамка. Тези подходи биват: затворени и отворени тестове.

1.5. Затворени тестове

Затворени тестове (Black box testing или тестове от тип затворена черна кутия), представляват тестове, които игнорират вътрешната структура на програмните модули и проследяват продукта, верифицирайки, дали крайният резултат отговаря на очакваното поведение на системата, и какви резултати целят да бъдат постигнати, базирайки се на предварително записани действия, които се извършват.

Тестването от този тип разглежда софтуера като "черна кутия", изучавайки функционалността без никакви познания за вътрешното устройство. Човекът, който извършва теста, знае само какво трябва да прави софтуерът, но не и как го прави. [Patton, Ron’05]

1.6. Отворени тестове

Отворени тестове (White box testing или тестове от тип отворена бяла кутия), представляват тестове, чиято цел е проверка на програмната логиката на вътрешните компоненти за една система.

Отворените тестове са метод за тестване на софтуер, който тества вътрешните структури или работата на дадено приложение, но не и неговата функционалност. Този подход се базира на вътрешния облик на системата, както и на уменията за програмиране на тестера.

Специалистът, който проверява, избира входните данни за проследяване на тяхната програмна обработка и определя дали получените резултати отговарят адекватно на очакваното. [Limaye, M.G.‘09]

1.7. Анализ на видовете тестове

В следващата таблица, са представени схематично основните характеристики на видовете тестове, тяхното покритие, техния подтип и реализация.

Таблица 1. Класификация на видовете тестове, според основни характеристики

Тестов вид	Покритие	Реализиран от
КОМПОНЕНТЕН	Малки единици от кода и/или отделни компоненти	Програмистът написал кода
ИНТЕГРАЦИОНЕН	Множество от компоненти	Множество от програмисти
ФУНКЦИОНАЛЕН	Целият продукт	Независими тестери
РЕГРЕСИОНЕН	Целият продукт, без новите функционалности (в нова версия)	Независими тестери
ПРОИЗВОДИТЕЛНОСТ	Целият продукт	Независими тестери
СИГУРНОСТ	Целият продукт	Независими тестери и/или множество от програмисти
ПРИЕМЕН	Целият продукт	Независими тестери и/или заинтересованите лица
ALPHA	Целият продукт	Независими тестери, тестващи в затворена среда и/или заинтересованите лица
BETA	Целият продукт	Независими потребители, които ще използват приложението. Продуктът е изцяло достъпен като се акцентира, че е в Beta версия.

Глава 2 – Разработка на програмна логика за валидиране на тестове

2.1. Удостоверяване на тестове (Test Assertion)

Тук ще представим примерен подход за логическа реализация на инструмент за валидиране на тестови резултати и разширеното приложение върху последващата работна рамка.

2.2. Валидиране на резултати

За потвърждение, че даден тест е успешен, е необходимо в края на фазата от неговото изпълнение да се извърши рационална проверка на резултата. За целта трябва логически да създадем правилото по оценка на получения резултат, за да гарантираме вярност на тестовите действия, които упражняваме

Сценарий

Логически ще тестваме, че правилно сме заредили дадена Интернет страница и виждаме логото в секцията „Начало“. Ако условието е изпълнено, значи нашият тест е успешен, ако не е, значи, че тестът ще пропадне и ще прихване проблем (бъг).

Тестов подход

За целта ще добавим методът isDisplayed (метод, който връща булев израз, дали условието е изпълнено 1 true или не е изпълнено 0 false) към променливата HomePageLogo и ще очакваме резултат от действието.

Описание на положителен тест

Положителният тестов процес винаги валидира действието като правилно, ако то отговаря на системните изисквания с които разполага разработчикът на теста. Ще представим процедурите по действията в следната таблица:

Таблица 2. Логическо разделение на тестовото състояние

Церемония	Списък от действия
Arrange (Given)	Подготовка на състоянието след което ще извършим действие. Състоянието представлява отправна точка от която ще започне нашето тестване; Програмна репрезентация: • Създаваме променливата HomePageLogo, която отговаря графично на селектора в DOM.
Act (When)	Извършваме действието в нашия случай тестовия сценарий. Започваме действие което ще е отправна точка към изходен резултат на тестваното от нас приложение; Програмна репрезентация: • Навигация до Интернет страницата на нашето приложение.
Assert (Then)	Проверка, дали върнатата стойност или държанието на приложението съвпада с очаквания резултат и валидираме изхода на теста; Програмна репрезентация: • Проверяваме, дали върната стойност от вече извиканият метод връща стойност TRUE, каквото очакваме. Ако е изпълнено условието, тестът се смята за положителен, в противен случай теста се проваля и логото на началната страница не е заредено успешно и алгоритмът не го отчита. o HomePageLogo.isDisplayed
Междинна клауза (and)	В междинната клауза “and” можем да изброяваме и отделяме набор от действия както в Act (When) така и в Assert (Then) клаузи. Тя служи за по-добра подредба и яснота, при множество от действия в някоя от посочените клаузи.

2.3. Основни методи за проверка на изрази

Освен основните церемонии, които бяха заложени в Таблица 2, е нужно и логическо манипулиране на изразите от действията на теста. В следващата Таблица 3 ще бъде проследена логическата връзка и ключовите думи на различните методи на работа, които ще се реализират в софтуерната тестова рамка.

Таблица 3. Основни подходи за проверка на изрази

Ключова дума на метода	Описание
assert / verifyThat	Проверява валидността на даден израз.
verifyAll	Проверява съвкупност от правила в рамките на един функционален блок. Също така се използва за „меки“ проверки (soft assertions);
isEqual / isEqualType	Проверява равенство на изрази по стойност и по стойност и тип;
isDisplayed / isNotDisplayed	Проверява, дали даден елемент е показан в DOM дървото или не е;
isFailed	Предизвиква изкуствен провал на теста, с идеята за негативни тестови проверки;
isNull / isNotNull	Проверява даден израз, дали няма резултат (не връща нищо) или връща даден резултат;
not / !	Оператор, поставен пред определен удостоверител, изменя логиката на действието и валидира противоположната стойност;
isTrue / isFalse	Проверява истинност или неистинност на даден булев израз;
isEnabled / isNotEnabled	Проверява, дали даден елемент е достъпен или не е.
isActive / isNotActive	Проверява, дали даден елемент е активен за взаимодействие или не е.

Глава 3 – Реализация на функционални тестови механизми

За целите на тестовете ще се използва примерен демо уебсайт предоставящ, електронен магазин и следните технологии: шаблона Page Object (PO) и програмният език Java. Помощните библиотеки за реализацията са: Java, Selenium WebDriver, JUnit. След пълното изпълнение на тестовете ще се генерира доклад, чрез който обстойно се наблюдават резултатите от всеки един скрипт.

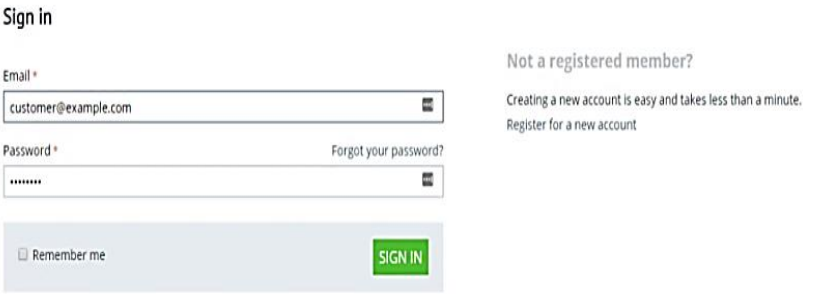
В рамките на настоящата разработка ще валидираме примерен уебсайт върху, който ще бъдат изпълнени примерни тестови скриптове. Системата включва в себе си множество графични компоненти, които могат да предоставят информация, дали даден функционален тест извършва работата, за която е създаден или не.

Начинът, по който е реализирана тестваната система, не е от значение за целите на научната разработка. Тя ще се използва като образец, който демонстрира как се изпълняват и конфигурират тестови подходи

върху произволен интерфейс. Всички примери в разработката могат да бъдат приложени върху произволна система, като се вземат предвид необходимите настройки на средата и целевите тестове.

Тестовете могат да бъдат много, но елементите визуализиращи се при изпълнението на дадена функционалност са едни и същи, с цел преизползване на ключово повтарящи се фрагменти. Софтуерните тестери трябва да разбият програмната логика на отделни модули, а именно това е концепцията при шаблона POP.

Задачата на функционалните тестове от този род са, освен за проверка на функционалното държание на конкретното действие, да проверят и цялостната структура на уеб страницата, чрез преминаване на всичките страници и верифицирането им като работещи.



Фигура 1. Графична визуализация на вход в системата

3.1. Сценарий на функционален тест

При наличие на система за тестване може да се реализира сценарий, който да тестваме с помощта на функционален тест. Функционалният тест представлява проверка на функционалностите на едно приложение и дали то е разработено да функционира според очакванията на клиентите. Пример: формата за вход в системата, показана на фигура 1. Изпълнява следната функционалност:

Потребителят въвежда своя е-мейл адрес и своята потребителска парола, след което натиска бутона Sign In.

3.1.1. Положителен тестов сценарий

Ако е-мейл адресът и паролата са валидни системата ще пренасочи потребителя към страницата на потребителския интерфейс, на базата на който ще валидираме резултата от успешния тест.

Стъпки:

1. Потребителят въвежда правилен е-мейл адрес;
2. Потребителят въвежда правилна парола;
3. Потребителят натиска бутона за вход в приложението;
4. Приложението зарежда нов екран към потребителския панел.

При удостоверяването на четвъртата стъпка, можем да кажем, че даденият положителен тестов сценарий е успешен и функционалността за вход в приложението работи. Трябва да се отбележи, че това не гарантира последващите действия на потребителя, а се ограничава само до успешния вход в системата. Другите зони на приложението ще се тестват от други тестови сценарии.

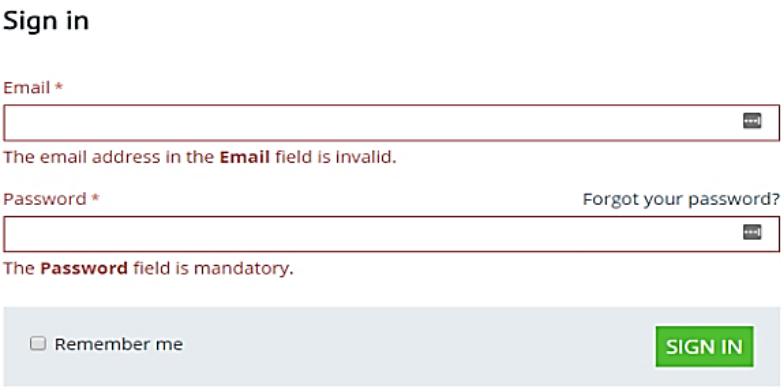
3.1.2. Негативен тестов сценарий

В подхода на тестване на едно софтуерно приложение, трябва ясно да бъдат описани и всички негативни подходи към тестовия процес. Трябва ясно да отбележим, че негативните тестове, като такива, трябва да минат успешно, за да верифицират очакваният негативен отклик на системата.

В следващата таблица са описание действия, целящи негативна проверка, дали приложението е адекватно защитено от негативни опити за въздействие.

Таблица 4. Негативни тестови случаи и системен отклик

Потребителско действие	Очакван системен отговор
Потребителят натиска бутон за вход без да е въвел е-мейл адрес или парола.	Системата извежда информация за липсващи данни.
Потребителят въвежда е-мейл адрес, без парола и натиска бутона за вход.	Системата извежда информация за липсваща парола.
Потребителят въвежда парола без да е въвел е-мейл адрес.	Системата извежда информация за липсващо е-мейл адрес.
Потребителят въвежда грешен е-мейл адрес, но вярна парола.	Системата извежда информация за грешен е-мейл адрес или парола.
Потребителят въвежда грешен е-мейл адрес, но невярна парола.	Системата извежда информация за грешен е-мейл адрес или парола.
Потребителят въвежда невалиден грешен е-мейл адрес и парола.	Системата извежда информация за грешен е-мейл адрес или парола.



Фигура 2. Графичен сценарий показващ, грешка при опит за вход в системата

Обхващането на всички положителни и негативни действия от дадения сценарий, ни осигурява правилна и консистентна работа на потребителския вход в приложението.

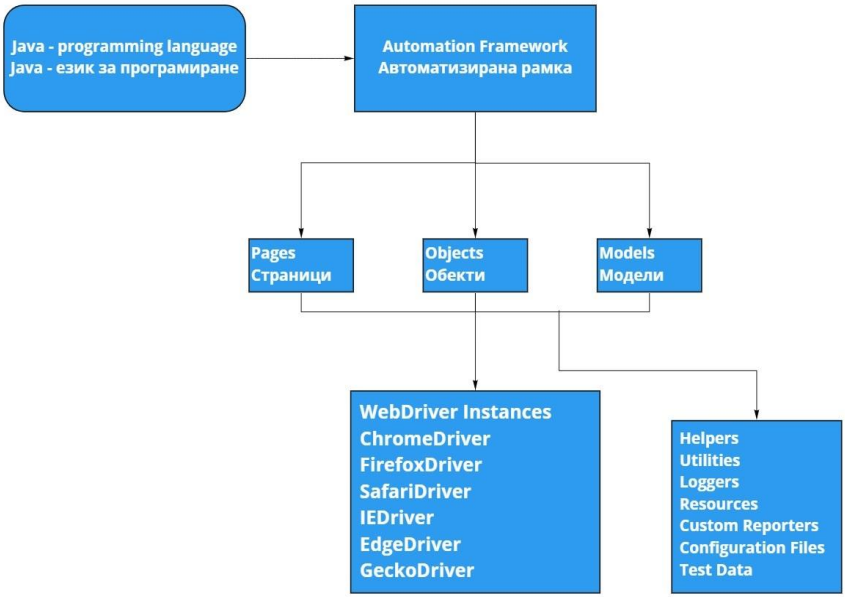
Глава 4 – Софтуерна рамка за тестване на уеб приложения

Ползата от създаването на автоматизирана рамка, е тя да се използва като шаблон за всеки нов проект, предоставяйки решения за избягване на всички известни проблеми. Действията, които можем да извършваме са: навигация до страница, кликане върху бутон или попълване на форма за вход, както много други, които потребителят може да извърши. След това трябва да се проверят и отчетът действителните спрямо очакваните резултати. Въпреки че разполагаме с много различни случаи/тестове, как и кога ги използваме, зависи от сценария на действие и нуждата.

В някои случаи ще изпълним няколко теста в определен ред. В други, ще се изпълняват само конкретни. За да се постигне всичко това, тестерите обикновено трябва да интегрират различни рамки или библиотеки заедно. Както бе споменато, повечето проекти имат общи потребителски действия, които трябва да се извършват в автоматизиран поток. Тези взаимодействия са разработени и имплементирани в самата рамка и тестерите могат да ги използват веднага без да губят време, за да ги напишат или пренапишат отново от самото начало.

Друго предимство от създаването на автоматизирана софтуерна рамка, е възможността да се подготвят готови конфигурирани класове и пакети за използване на външни източници като: облачно-базирани инструменти, външни носители, сървъри или други ресурси. Използването на готовите техники в автоматизираната рамка спестява време, предотвратява допълнителни усилия и улеснява разбирането на бизнес логиката.

Софтуерна рамка за тестване (Test Automation Framework). Автоматизираните софтуерни рамки имплементират списък от архитектурни шаблони (подходи), за разделение на логиката от съдържанието. Те са акцентирани към изграждането на архитектурно независими подходи увеличаващи гъвкавостта и преизползваемостта си с разрастването на тестваната система.



Фигура 3. Структура на функционалната софтуерна рамка за графични контроли

Още едно предизвикателството при реализация на автоматизирана рамка за тестване се състои в намирането на баланс при избора на регресионни групи от тестове и пакетни логики, изпълнявани върху малко зони и потенциално позволяващи пропускането на сериозен дефект. Последните тенденции показват, че разработчиците на софтуер, използващи „Agile", се обръщат към тестването на автоматизацията като отговор на намирането на точния баланс и бързина при разработка.

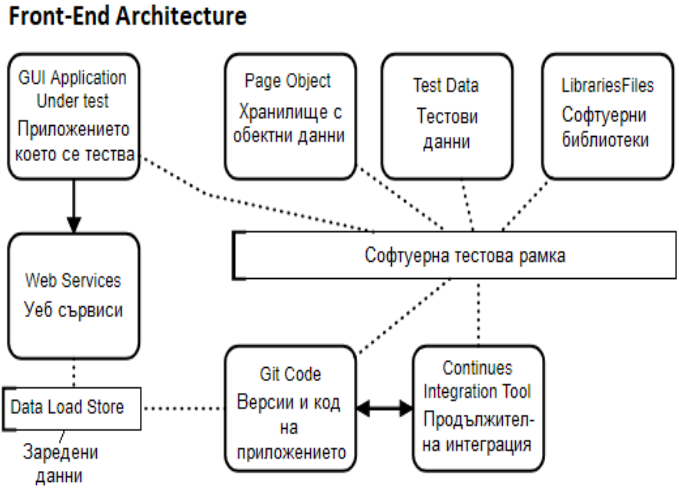
4.1. Многослойна архитектура

Софтуерните проекти следват многослоен процес на проектиране, където разработката е разделена на отделни слоеве. Системата предоставя различни интерфейси на различни типове клиенти. Всеки клиент се интересува от използването или консумирането на тази система по различни начини. Потребителският интерфейс може да взаимодейства с дадена услуга, за да генерира информация, необходима за визуализиране на потребителя. С уеб услуга се извличат или поддържат данните в базата данни или чрез външен интерфейс към други системи.

Тестовата автоматизация изисква значително разработване на софтуер, поради което усилията трябва да се третират по същия начин като всеки друг софтуер. Структурирането на автоматизацията в отделни слоеве създава абстракция, което увеличава гъвкавостта и дълбочината на покритие, това води до по-добро общо качество на тестваната система. Няма определен (фиксиран) брой нива, които дадена система трябва да има. Обикновено има поне две: представителен и бизнес слой. [Kelly'03] [Robot Framework] [SeleniumDoc]

4.2. Визуализационен слой

Слоят за визуализация обхваща графичния потребителски интерфейс (GUI), изобразен на фигура 4. Тази подредба се фокусира върху тестването на основното валидиране на GUI и взаимодействията на потребителите. Целта на тестовете в тази подредба е да потвърдят, че правилната функционалност се поддържа при потребителския слой. За този слой автоматизираните тестове са ориентирани към GUI, изградени чрез технологията Selenium WebDriver и взаимодействието с клиентския Web Browser. [David, Chappell'00] [SeleniumDoc]



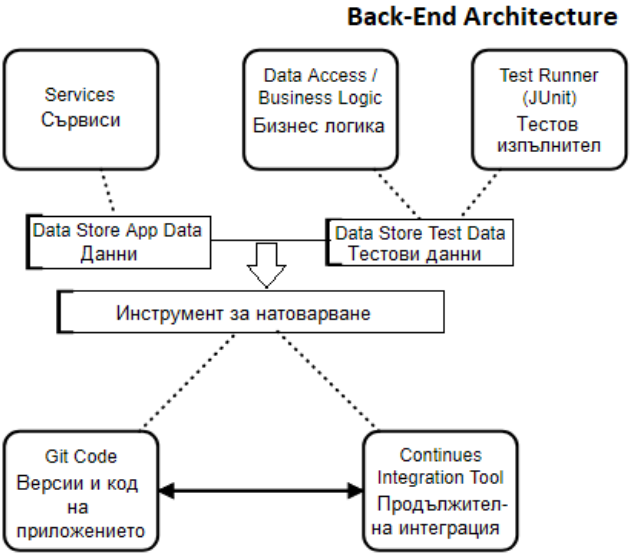
Фигура 4. Архитектура на презентационен слой

Сървърът Jenkins Cl се използва за управление на тестове, както и визуализиране на резултатите, при всяка нова промяна или внедряване на нова функционалност. Освен това настройката му включва изпълнението на всички тестове на определен интервал от време и генериране на специфичен отчет (report).

4.3. Приложно обработващ слой

Обработващия слой представлява подразделението на бизнес логиката на приложението: данни и уеб услуги, което е представено на фигура 5. Целта е да се изолират и валидират бизнес правилата, целостта на данните и функционалността на уеб услугите отделно от визуализационния слой GUI. Проектирането на алгоритми за проверяване и системните точки са добри примери за бизнес логика, която се нуждае от тестване в тези нива.

Проектът използва „Cucumber", рамка за разработване на тестове на поведението (BDD), заедно с JUnit - рамка за тестване на единиците (units), за да се справи с бизнес слоя. Данните се достъпват чрез обекти. Тестването на уеб услуги се извършва с Apache JMeter или Postman или друг инструмент с отворен код. JMeter осигурява възможността за едновременно тестване на функционалността и производителността. Отново всички автоматизирани тестови скриптове бяха съхранени в хранилището на изходния код. [Helleszy, Wynne'12] [Duvall, Glover'07] [CucumberDoc]



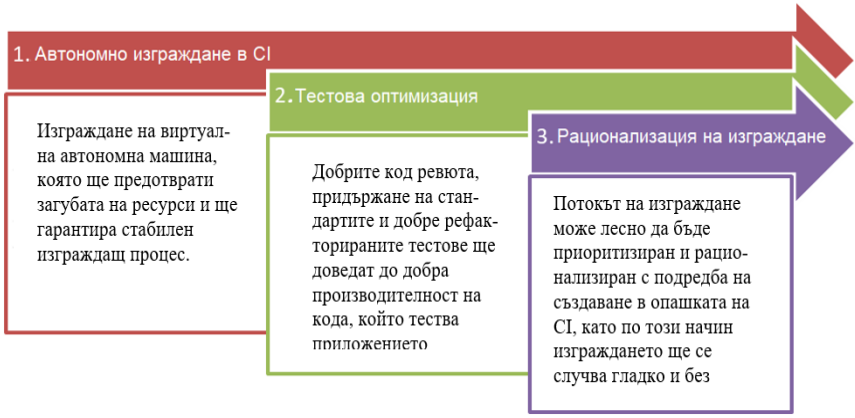
Фигура 5. Архитектура на приложно обработващ слой

4.4. Непрекъсната интеграция (Continuous Integration)

Непрекъснатата интеграция (Continuous integration – CI) представлява подход в софтуерната разработка, за свързване на всички налични копия, по които програмистите работят в споделено главно виртуално пространство. За първи път името се споменава от Грейди Бууч в метода му от 1991 г. Подходът е приет като част от екстремното програмиране (XP), което подкрепя интегрирането по повече от един път на ден (сглобяване на новия код към стария). Основната цел на CI, е да се предотвратят интеграционни проблеми. [Duvall, Glover'07]

В същността си непрекъсната интеграция представлява практиката, при която нов или променен код се добавя към главното хранилище на копието от последната версия на приложението. То трябва да се изпълнява доста често, за да не остава по-дълъг период между качването и изграждането, както и за да не възникнат грешки, които да останат незабелязани и некоригирани своевременно от програмистите. Нормално е да се създава „билд“ след всяко качване в главното хранилище. Много инструменти поддържат автоматизирано периодично качване.

Фигура 6. Пълен цикъл на последователност при подхода на непрекъсната интеграция [Duvall, Glover'07]



Фигура 7. Стъпки за гарантиране на безпроблемно изграждане до стабилна тестова интеграция

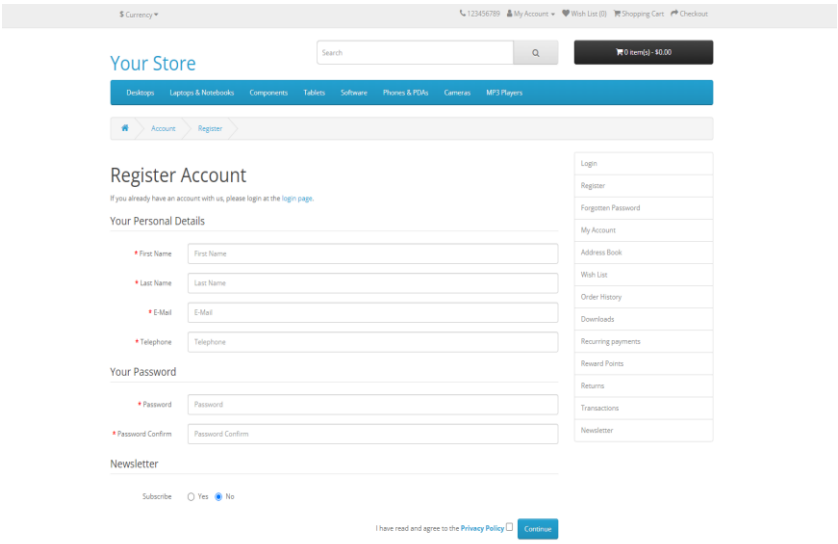
4.5. Тестване на работната рамка

За да демонстрираме успешната работа на създадената софтуерна рамка за тестване на уеб приложения, ще направим експеримент, доказващ нейната ефективност. За целта имаме разработени автоматизирани тестове, обхващащи примерното Интернет приложение: **https://demo.opencart.com**.

То представлява тестов онлайн магазин с много функционалности. Основната цел и изследване е да сравним поведението между разработената софтуерна тестова рамка и базов тривиален подход, използващ единствено Selenium WebDriver и JUnit. Ще демонстрираме тестов подход за регистрация в системата, чрез разписване на тестов сценарий и програмен код.

Тестов сценарий:

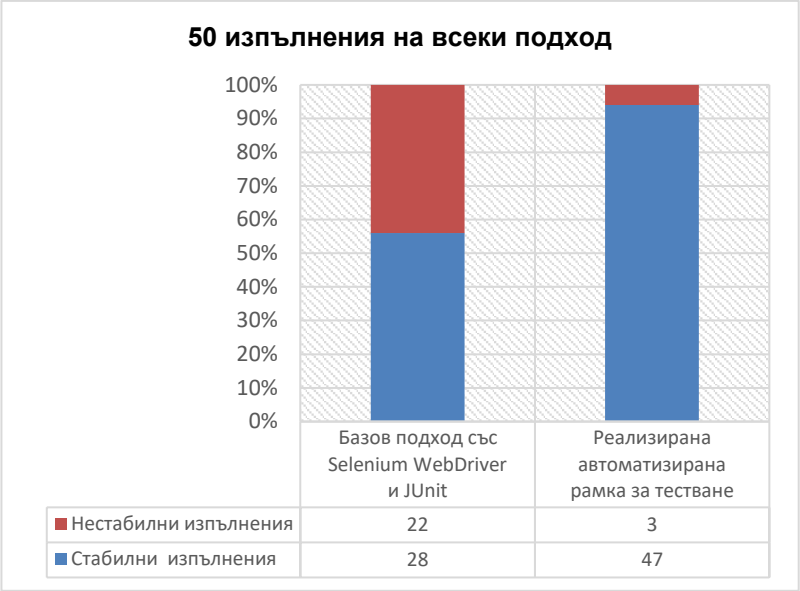
1. Отваряне на уебсайта: **https://demo.opencart.com**
2. Щракване на бутона “Account”
3. Натискане от падащото меню на “Register”
4. Попълване на следните потребителски данни:
    - First Name (текстово поле)
    - Last Name (текстово поле)
    - E-Mail (текстово поле с валидация за E-Mail адрес)
    - Telephone (числово поле)
    - Password (поле за парола)
    - Repeat Password (поле за парола)
    - Newsletter preferences (поле с отметка)
    - Terms agreement (поле с отметка)
5. Щракване на бутона “Continue”



Фигура 8. Потребителски изглед за регистрация на уебсайта

Изследване за стабилност и консистентност

Проведено е следното изследване на база 50 изпълнения на всеки един програмен подход на сценария за създаване на регистрация с уникални данни при всяко изпълнение.



Фигура 9. Сравнителна диаграма между реализираната рамка за тестване и базовата рамка

Резултатите на фигура 9 показват, че базовият подход доста често се проваля при опит да локализира елементите за навигиране и щракване, тъй като в нашата реализирана автоматизирана рамка са разработени и заложени конфигурационни от методи за изчакване на елементи, докато при базовия подход нямаме подобна структура.

От този факт може да бъде направено следното заключение: доста често при опит за намиране на дадения елемент, скриптът считаше, че елементът не е видим, а реално той е зареден успешно на браузъра.

В заключение може да се твърди, че подходът на реализираната автоматизирана рамка е с над 93% успешен, което води до стабилни тестове при графичните контроли, докато базовия подход има малко над 55% успеваемост, което прави този подход доста нестабилен и неконсистентен за работа и реализиране на автоматизирани тестове.

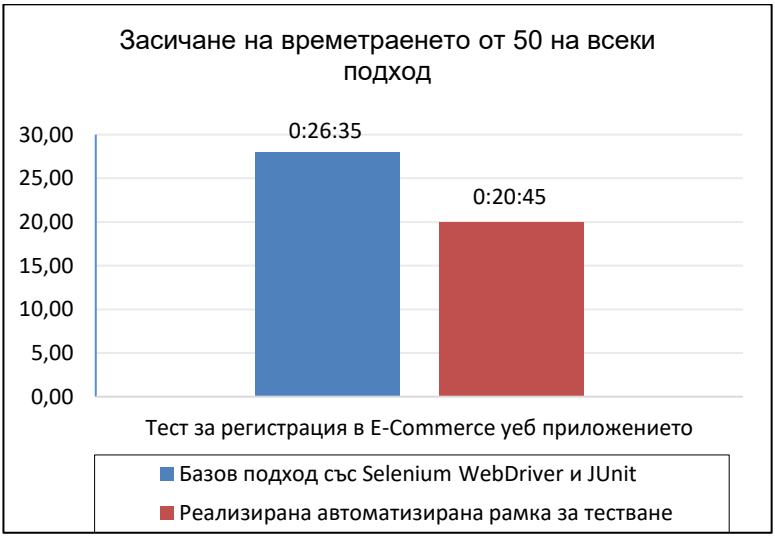
Времетраене на тестовите случаи

Проведено бе следното изследване използващо сценария от тестовия случай за регистрация в уебсайта. Целта е да демонстрираме времетраенето на изпълнение на тестовия сценарии. Тестовите повторения трябва да бъдат използвани едва когато имаме външен фактор за провал, като неработеща функционалност свързана с конфигурация или външен фактор.

Програмната логика в изчакващия клас Wait има заложени методи, които карат действията и проверките да имплементират тази времева рамка, с цел избягването на провал, поради не намирането на локатор в този интервал или забавена обработка на ниво приложен интерфейс, както бяха доста от случаите от фигура 9.

Стабилните тестове гарантират, че няма да бъдат провалени поради други фактори, като конфигурационни, не добре структуриран програмен код и др. При нестабилното изпълнение на даден тест или многократното му повторение, без това да бъде необходимо се губят както ресурси, така и време при неговото анализиране, което води до проблем в подхода и цялостната идея за автоматизирани тестове.

Затова в нашата автоматизирана рамка за тестване на уеб приложения, заложихме изчакващ клас с допълнителни конфигурационни параметри, които спомагат правилната и консистентна работа.



Фигура 10. Сравнителна диаграма за времетраенето на тестовото изпълнение

Наблюдения

След изпълнението на всеки един подход по 50 пъти, можем да заключим, че базовият завърши с 6 минути повече, сравнение с реализираната автоматизирана рамка. Причината за това бяха конфигурираните изчаквания и допълнително имплементирания Steps Listener, описан подробно в Глава 2, благодарение на който стъпките заложени на всяко действие се изпълняват плавно и консистентно. Базовият подход доста често се проваляше и започваше отначало, докато подходите при реализираната рамка изчакваха необходимото време и не се стигна до повторение на теста.

ЗАКЛЮЧЕНИЕ

В заключение, може да се каже, че поставените цели и задачи, поставени на научната разработка, са реализирани. Демонстрирани са теоретичните основи на софтуерното тестване, техните ползи и недостатъци. Успешно е създадена софтуерна тестова рамка с многослойна архитектура при тестването на уеб приложения. Показана е интеграцията с външни инструменти и са описани техните ползи.

Софтуерната разработка в днешни дни е изключително динамична и изисква голяма подготовка. Направените изследвания и анализи доказват, че при възникването на много проблеми има и изобилие от решения, някои от тях са демонстрирани в научната разработка. Софтуерното тестване е процес, който никога не приключва, дори след финалната разработката на софтуерния продукт, той продължава да се извършва периодично, с цел гарантиране качеството на продукта, неговата защита и пълнота.

Връзката между целите, задачите и мястото на описание в дисертационния труд и публикации са предоставени в следната таблица:

Таблица 5. Реализирани резултати от задачите на дисертационния труд

Задача	**Принос**	**Секция**	**Авторска публикация**
Създаване на програмен модел на бъдещата разработка, както и базови класове и методи за валидиране, верифициране и изпълнение на действия от общ характер.	Проучени са основните програмни модели в подхода на софтуерното тестване. Анализирани са добрите практики и са изведени теоретичните постановки в помощ на научната работа. Чрез представянето на шаблона POP е направен подробен анализ и реализация по дадения модел. Разглежда се капсулираната логика и нейните предимства във връзка с бъдещата разработка на тестовете. Представяне на методика за реализация на функционални тестове и модел за бъдещото им поддържане.	Глава 1 Глава 2	Публикация 1 Публикация 3
Създаване на многослойна архитектура и разработка на софтуерна тестова рамка за функционални тестове и различни шаблони за абстракция на функционални автоматизирани скриптове.	В Глава 2 са описани техническите средства и предпоставки за реализацията на тестовите механизми. Създадена е основополагаща логическа последователност като основа за изпълнение на функционални тестове. Представени са няколко подхода за записване на валидни действия и тяхната поддръжка в последствие. Направен е анализ с ползи и негативи от използването на предоставените инструменти.	Глава 2 Глава 3 Глава 4	Публикация 2
Дефиниране на цялостна интеграция след вече разработената архитектурна софтуерна работна рамка от Задача №2, за лесна преносимост, съвместимост и използваемост при вграждането на софтуерната тестова рамка към нови уеб приложения.	В Глава 3 е представена употребата на автоматизиран инструмент, който е облачно-базиран и се използва за изпълнението на автоматизираните скриптове, върху множество различни устройства, платформи, операционни системи и други. В Глава 4 се описва финалната интеграция на всички под разработки в едно цяло, а именно – многослойна тестова архитектура за тестване на уеб приложения, използваща гъвкавата методология Agile и церемонията на Scrum.	Глава 3 Глава 4	Публикация 3 Публикация 4 Публикация 5

АПРОБАЦИЯ НА ДИСЕРТАЦИОННИЯ ТРУД

Публикации

1. Lefterov D., E-Commerce Testing Framework - Научна сесия "Дни на науката 2018 с международно участие" СУБ Пловдив, 2-3 ноември 2018, Научни трудове на СУБ-Пловдив, Серия В, Техника и технологии, том XVII, 2019, 26-32, ISSN: 1311-9419.
2. Лефтеров Д, Енков С., Многослойна архитектура за автоматизация на тестове в Agile - Научна сесия "Дни на науката 2018 с международно участие" СУБ Пловдив, 2-3 ноември 2018, Научни трудове на СУБ-Пловдив, Серия В, Техника и технологии, том XVII, 2019, 18-25., ISSN: 1311-9419.
3. Denislav Lefterov, Svetoslav Enkov, Automated software testing framework “Stassy”, International Scientific Journal "Industry 4.0", Vol. 4 (2019), Issue 4, pp. 171-174, ISSN (Web): 2534-997X; ISSN (Print): 2534-8582.
4. Lefterov D., Enkov S., Avoiding frequently failing tests in Continuous Integration, IOSR Journal of Computer Engineering (IOSR-JCE), Vol. 21, Issue 6, Ser. I (Nov-Dec 2019), pp. 45-55, DOI: 10.9790/0661-2106014655, ISSN (Print): 2278-8727, ISSN (Web): 2278-0661.
5. Denislav Lefterov, Svetoslav Enkov, Automated Testing Framework with BrowserStack Integration, International Scientific Journal "Industry 4.0", Vol. 5 (2020), Issue 3, pp. 105-108, ISSN (Web): 2534-997X; ISSN (Print): 2534-8582.

Забелязани цитирания

- Lefterov D., E-Commerce Testing Framework - Научна сесия "Дни на науката 2018 с международно участие" СУБ Пловдив, 2-3 ноември 2018, Научни трудове на СУБ-Пловдив, Серия В, Техника и технологии, том XVII, 2019, 26-32, ISSN 1311-9419.
    - Olena Trofymenko, Manakov S., Pasternak Yu, Yuliia Loboda, Automation Of Testing E-commerce Websites, Сучасна Спеціалнъна Техника - Modern Special Technics, ISSN 2411-3816 (PRINT), DOI: https:/doi.org/10.36486/mst2411–3816.2021.2(65) Issue 2(65) 2021, https://www.researchgate.net/publication/359341966

ОСНОВНИ ПРИНОСИ НА ДИСЕРТАЦИОННИ ТРУД

Основно приносите на дисертационния труд могат да бъдат характеризирани като научни, научно-приложни и приложни.

Научни приноси на дисертационното изследване са:

Н1. Създаден е общ теоретичен модел за софтуерно тестване

Научно-приложни приноси на дисертационното изследване са:

НП1. Проектирана е софтуерна рамка за тестване на уеб приложения на базата на многослойна архитектура с интегрирани инструменти за софтуерно тестване.

НП2. Разработен е алгоритъм за валидиране на софтуерни тестове на базата на системното поведение на приложението.

Приложни приноси на дисертационното изследване са:

П1. Реализирана е автоматизирана софтуерна рамка за тестване на уеб приложения

П2. Проведен и анализиран е експеримент с реален уеб проект за доказване на приложимостта на създадената рамка

П3. Направен е сравнителен анализ на разработената софтуерна тестова рамка спрямо съществуващи базови комерсиални рамки.

Теоретичните постановки и софтуерните практики, създадени в рамките на дисертационния труд, са използвани за създаване на учебни материали в ПУ „Паисий Хилендарски“. Списък на проведени учебни дисциплини свързани с темата на дисертацията:

- Избираема дисциплина „Технологии за управление на софтуерно тестване“ през учебната 2018/2019 г.;
- Избираема дисциплина „Технологии за управление на софтуерно тестване“ през учебната 2019/2020 г.;
- Избираема дисциплина „Технологии за управление на софтуерно тестване“ през учебната 2020/2021 г.

БЛАГОДАРНОСТИ

Специални благодарности към доц. д-р Светослав Енков за оказаната подкрепа през годините, посветени на настоящето дисертационно изследване, както към всички колеги от катедра „Компютърна информатика“ на ФМИ за тяхното разбиране, колегиалност и подкрепа.

БИБЛИОГРАФИЯ

Книги и статии

[Craig, David et al.‘02] Systematic Software Testing. Artech House. p.7.ISBN: 1-58053-508-9.

[Dustin et al.‘09] Implementing Automated Software Testing. ISBN: 978-0-321-58051-1.

[Patton, Ron’05] Software Testing (2nd ed.). Indianapolis: Sams Publishing. ISBN 978-0672327988.

[Limaye, M.G.‘09] Software Testing. Tata Education. pp. 108–11. ISBN: 9780070139909.

[David, Chappell’00] Java Message Service O’Reilly Media, 2000. p.240.

[Helleszy, Wynne’12] The Cucumber Book: Behavior-Driven Development for Testers and Developers, 2012.

[Duvall, Glover’07] Continuous integration: improving software quality and reducing risk, 2007, ISBN: 978-0-321-33638-5.

Интернет ресурси

[Kelly’03] 'Choosing a Test Automation Frarnework’ 2003
http://www.ibm.com/developerworks/rationa1/1ibrary/591.htm1 (последно посетен 22.12.2018).

[SeleniumDoc] документация на „Selenium“
https://www.selenium.dev/documentation/en/ (последно посетен 10.08.2020).

[CucumberDoc] документация на програмният език „Cucumber“
https://cucumber.io/docs/cucumber/ (последно посетен 18.08.2020).