МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г.ЧЕРНЫШЕВСКОГО»

Кафедра информатики и программирования

АВТОМАТИЗИРОВАННОЕ ТЕСТИРОВАНИЕ КЛИЕНТ-СЕРВЕРНОГО ПРИЛОЖЕНИЯ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 441 группы
направления 02.03.03 Математическое обеспечение и администрирование информационных систем
факультета компьютерных наук и информационных технологий
Лукашина Кирилла Владиславовича

Научный руководитель:
доцент кафедры ИиП Е.В.Кудрина

Зав. кафедрой ИиП,
к.ф.-м.н., доцент М.В. Огнева

Саратов 2021

ВВЕДЕНИЕ

Актуальность темы. В настоящее время наблюдается увеличение сложности программных продуктов, продолжительности их жизненного цикла, а также и числа разработчиков, поддерживающих данные продукты. В свою очередь повышается риск появления ошибок. Устранение ошибки на стадии сопровождения готового ПО обходится в среднем в 200 раз дороже, чем на стадии определения требований, а в результате позднего выявления ошибок общий бюджет проекта возрастает на 40-50%. Очевидно, что для избежания ошибок и проверки удовлетворения требованиям в программном продукте необходимо его систематизированное тестирование.

Тестирование вручную уже не может качественно и количественно покрыть современные приложения. Поэтому сейчас широко используются не только различные методики тестирования, позволяющие максимально эффективно покрыть ПО тестами, но и инструментальные средства, направленные на повышение качества и сокращение ресурсозатрат на тестирование ПО. В последнее время стали популярными так называемые фреймворки автоматизированного тестирования. Фреймворк для тестирования — программное обеспечение, содержащее набор инструментов и библиотек, которое позволяет удобно работать с тестовыми данными, сокращает время создания новых тестов, упрощает их запуск и предоставляет возможность создания отчетов. Разработкой фреймворков автоматизированного тестирования занимаются QA-инженеры.

Цель бакалаврской работы – разработать и реализовать фреймворк для автоматизированного тестирования клиент-серверного приложения (на примере приложения «Point of Store»).

Поставленная цель определила **следующие задачи**:

1. Обобщить и систематизировать знания в области разработки клиент-серверных приложений, приобретенные в ходе обучения.

2. Изучить современные методы, технологии и инструментальные средства, используемые для реализации автоматизированного тестирования клиент-серверных приложений.
3. Описать модель приложения «Point of Store».
4. Разработать фреймворк для автоматизированного тестирования приложения «Point of Store».
5. Выполнить автоматизированное тестирование приложения «Point of Store» и проанализировать полученные результаты.

Методологические основы. Автоматизированное тестирование клиент-серверных приложений представлено в работах Стива Макконелла [1], Сэмми Пьюривала [2], Святослава Куликова [3], Лайза Криспин и Джанет Грегори [4], Поля Дюваль и др [5].

Практическая значимость бакалаврской работы. В ходе выполнения бакалаврской работы был разработан и реализован тестовый фреймворк для автоматизированного тестирования клиент-серверного приложения «Point of Store». Путем сбора статистики и анализа запусков автоматизированных тестов было выявлено, что фреймворк справляется с задачей выявления ошибок и имеет достаточно высокую эффективность в перспективе.

Структура и объем работы. Бакалаврская работа состоит из введения, 2 разделов, заключения, списка использованных источников и 1 приложения. Общий объем работы – 72 страницы, из них 53 страницы – основное содержание, включая 10 рисунков и фрагменты исходного кода. Список использованных источников информации содержит 25 наименований.

КРАТКОЕ СОДЕРЖАНИЕ РАБОТЫ

Первый раздел «Тестирование клиент-серверных приложений» посвящен теоретическим основам архитектуры клиент-серверных приложений, теории тестирования и особенностей его автоматизации. В данном разделе были определены следующие понятия:

«Клиент — сервер» (англ. client–server) — вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами.

GUI, Graphical User Interface — система средств для взаимодействия пользователя с компьютером, основанная на представлении всех доступных пользователю системных объектов и функций в виде графических компонентов экрана.

Веб-приложение — клиент-серверное приложение, в котором клиент взаимодействует с веб-сервером при помощи браузера.

Тестирование программного обеспечения — процесс исследования, испытания программного продукта, имеющий своей целью проверку соответствия между реальным поведением программы и ее ожидаемым поведением на конечном наборе определенных тестов. Тестирование включает в себя задачи по планированию тестовых сценариев, дизайну тестов, их выполнению и анализу полученных результатов — все это называется функциональным тестированием.

Автоматизация тестирования — часть процесса тестирования программного обеспечения, которое включает проведение таких основных функций и шагов теста, как запуск, инициализация, выполнение, анализ и выдача результата автоматически посредством специализированных инструментов, что помогает сократить время тестирования и упростить его процесс в перспективе.

Во втором пункте раздела была описана проблема архитектуры ПО для автоматизированного тестирования и ее решение в виде фреймворка автоматизированного тестирования с описанием решаемых с его помощью проблем:

- Исключение дублирования кода.
- Упрощение написания тестов путем простого составления последовательности шагов, вытекающих в тестовые сценарии.

- Упрощение поддержки уже написанных тестов.

Помимо устранения данных проблем, фреймворк также может иметь следующие возможности:

- Средства сборки позволяют легко добавлять новые библиотеки и запускать тесты из консоли.
- xUnit-библиотеки позволяют запускать тесты параллельно.
- Page Object Model разбивает фреймворк на структуру, позволяющую переиспользовать разные его части[8].
- Возможности Continuous Integration[9].
- Возможность использования логов, отчетов, нестандартных исключений, что сильно облегчают последующую отладку и улучшает отчетность.

В конце раздела также приведен краткий обзор инструментальных средств для автоматизированного тестирования.

Второй раздел «Автоматизированное тестирование приложения “Point of Store”» посвящен реализации фреймворка для автоматизированного тестирования клиент-серверного приложения «Point of Store» с помощью графического интерфейса.

Point of Store (POS) — это программно-аппаратный комплекс, позволяющий осуществлять торговые операции, как это делает обычный кассовый аппарат. За POS-системой закреплен типичный набор кассовых функций: учет и отпуск товара, прием и выдача денег, аннулирование факта покупки.

Для реализации фреймворка автоматизированного тестирования были выбраны следующие инструменты и технологии:

Java — сильно типизированный объектно-ориентированный язык программирования. Изначально создавался для программирования бытовых электронных устройств, а в итоге стал одним из наиболее популярных языков для множества различных задач. Основной особенностью языка является то, что программы на Java транслируются в байт-код, выполняемый виртуальной машиной Java (JVM) —— программой, обрабатывающей байтовый код и

передающей инструкции оборудованию как интерпретатор. Достоинством подобного способа выполнения программ является полная независимость байт-кода от операционной системы и оборудования, что позволяет выполнять Java-приложения на любом устройстве, для которого существует соответствующая виртуальная машина. И хотя использование виртуальной машины снижает производительность языка, для реализации фреймворка для автоматизированного тестирования это не является серьезной проблемой, так как во многом скорость тестов зависит от качества сетевого подключения и скорости отклика тестируемого приложения. В связи с крайне высокой популярностью языка, для него существует множество фреймворков, позволяющих значительно упростить разработку и тестирование программного обеспечения. Именно этот факт стал основным при выборе языка для реализации данного приложения.

Gradle — фреймворк для автоматизации сборки проектов на основе описания их структуры в файлах на языках Groovy и Kotlin вместо традиционной XML-образной формы представления конфигурации проекта. Gradle построен на принципах Apache Maven, но в отличие от Maven, имеет преимущества в удобстве и скорости работы. Например в Maven для описания структуры проекта используются xml-файлы конфигурации, и при необходимости кастомизировать сборку и запуск приложения придется прибегнуть к Batch-скриптам и java. Для реализации того же самого gradle предлагает написать нужный скрипт прямо в файле конфигурации на том же языке Groovy, что позволяют хранить все необходимое для сборки проекта в одном месте. А прирост в скорости работы достигается за счет того, что gradle собирает ациклический граф зависимостей и вызывает только необходимый минимум их них. Gradle используется для построения и управления проектами, написанными на Java, Kotlin, Groovy и Scala. Использование этого фреймворка значительно упрощает сборку проекта, а при совместном использовании многих других фреймворков, сборка проекта

вручную превращается в практически непосильную задачу. В связи с этим было принято решение использовать в проекте данный фреймворк.

Spring Framework — универсальный фреймворк с открытым исходным кодом для Java-платформы. Несмотря на то, что Spring не обеспечивает какую-либо конкретную модель программирования, он стал широко распространённым в Java-сообществе главным образом как альтернатива и замена модели Enterprise JavaBeans. Spring предоставляет большую свободу в проектировании. Этот фреймворк предлагает последовательную модель и делает её применимой к большинству типов приложений, которые уже созданы на основе платформы Java. Считается, что Spring реализует модель разработки, основанную на лучших стандартах индустрии, и делает её доступной во многих областях Java. Spring может быть рассмотрен как коллекция меньших фреймворков или фреймворков во фреймворке. Большинство этих фреймворков может работать независимо друг от друга, однако они обеспечивают большую функциональность при совместном их использовании. В данном приложении для создания каркаса используется Spring Boot, а точнее встроенный Inversion of Control-контейнер, который отвечает за конфигурирование компонентов приложений и управление жизненным циклом Java-объектов, что позволяет легко сконфигурировать приложение при запуске, например задать url тестируемого приложения, выбрать необходимую платформу(десктоп, мобильная версия) и так далее. И хотя возможности Spring будут использоваться не в полной мере, за счет своей популярности и доступной документации для реализации приложения был выбран именно он.

Allure — фреймворк для создания простых и понятных отчетов автотестов, интегрируемый почти с любым языком программирования. В Allure используется понятие тестового шага(англ. Step), — простого действия пользователя. Для этого необходимо пометить функции, реализующие различные действия, аннотацией @Step. Соответственно, любой тест превращается в последовательность таких шагов. Для упрощения поддержки

кода автотестов была реализовали вложенность шагов. Если одна и та же последовательность шагов используется в разных тестах, ее можно описать одним шагом и затем переиспользовать.

В результате Allure позволяет сгенерировать отчёт, понятный любому человеку из команды — имя тестового метода становится названием тест-кейса в отчете, а последовательность вызовов внутри представляется последовательностью вложенных шагов. А удобный графический интерфейс, наглядные графики, и отличная документация лишь способствовали выбору именно этого инструмента для отчетности.

TestNG — библиотека для тестирования для языка программирования Java. TestNG позволяет охватить широкий диапазон категорий тестирования: модульные, функциональные, сквозные и интеграционные.

При построении архитектуры фреймворка был применен шаблон проектирования Page Object Model и многоуровневый шаблон (Layered pattern). И путем отделения логики тестов от тестовых данных и от реализации действий были выделены следующие слои:

- Page Object — содержит локаторы элементов и простые действия с ними, такие как: клик, ввод и так далее. Методы доступны вышележащим слоям: test steps и самим тестам. Но не следует работать с ними напрямую из теста без необходимости.
- Test steps — объединение отдельных действий как из одной, так и из разных страниц в осмысленные пользовательские шаги(бизнес-функции). Работа происходит через интерфейсы страниц. За счет этого существенно уменьшается объем кода, так как повторяющиеся шаги будут объединены, а также улучшается читаемость тестовых сценариев. Используются в тестовом репорте.
- Test scenario — из отдельных шагов собирается тестовый сценарий, а также происходит передача тестовых данных.
- DTO — доменные объекты, реализующие бизнес-логику программы.
- Test Data — тестовые данные, необходимые для сценариев.

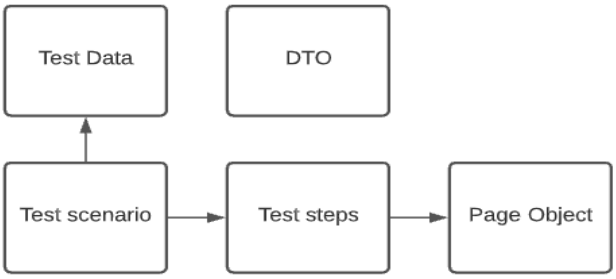

Рисунок 1 — Слои фреймворка

Далее в разделе описывается реализация выделенных слоев и полученный в результате класс BaseTest, который был определен как общий класс-предок для всех тестов, который хранит в себе TestSteps, TestData, WebDriver и различные тестовые данные и преднастройки. Затем проектировались и реализовывались тесты.

Затем описываются возможности конфигурирования запуска тестовых сценариев, и возможность их запуска на удаленном сервере. Автоматический запуск тестов, при обновлении исходного кода приложения называется CI — Continuous Integration — непрерывная интеграция которая является частью процессов CI/CD. С помощью данного подхода можно сильно увеличить качество приложения, так как если в новой версии кода появились ошибки, и на эти ошибки были написаны тесты, то система не пропустит данную версию кода в репозиторий, таким образом максимально рано исправить ошибки и не допустить возможные негативные последствия.

В конце раздела проведен анализ результатов автоматизированного тестирования. Реализованный фреймворк стал запускаться на ежедневной основе в марте 2021 года. За 3 месяца только с помощью тестового фреймворка было найдено не менее 100 ошибок, допущенных к сборке и развертыванию приложения в тестовом окружении. В процессе автоматизированного тестирования были выявлены ошибки по типу пропавшего текста с кнопок графического интерфейса, или недокументированные изменения логики работы приложения, такие как обновленный порядок всплывающих окон. Существовали ошибки связанные с неработающим поиском товаров и с отображением даты в различных

часовых поясах, а также ряд других более критичных для предприятия дефектов.

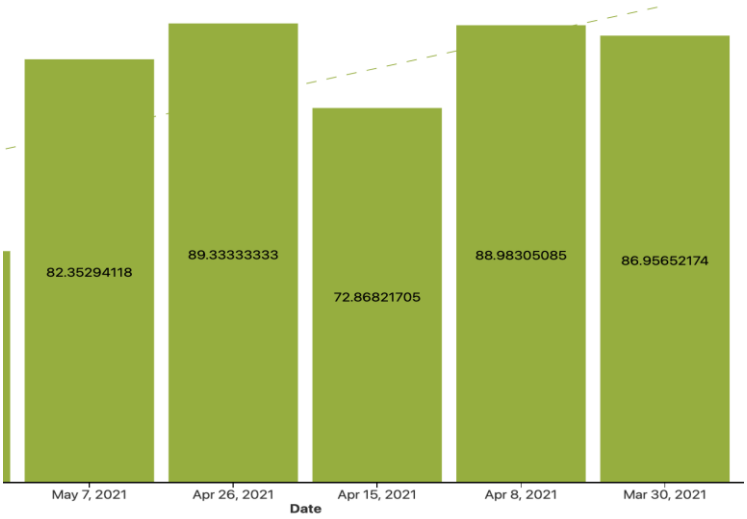

Рисунок 2 — Еженедельный отчет

В еженедельном отчете за 1 месяц на рисунке 2 видна стабильность в процентном соотношения прохождении тестов, но следует помнить, что новые ошибки появлялись с учетом исправления старых. Более 80% документированных ошибок, найденных с помощью автоматизированного тестирования были устранены в течение этого же месяца. Отсюда можно сделать вывод, что новые изменения или исправления часто приводили к ошибкам в уже существующем коде и соответственно сценариях использования. И фреймворк автоматизированного тестирования отлично справляется с задачей выявления ошибок подобного рода.

А ввиду того, что приложение находится в активной стадии разработки, и многое необходимо тестировать вручную на начальных этапах, то многие ошибки были выявлены немногим раньше самими инженерами, а не авто-тестами. Отсюда можно сделать вывод, что эффективность данного фреймворка со временем будет только расти.

ЗАКЛЮЧЕНИЕ

В ходе работы были выполнены все поставленные задачи. Были рассмотрены и изучены методы, технологии и инструментальные средства

для автоматизированного тестирования клиент-серверных приложений, в частности для построений фреймворка автоматизированного тестирования для веб-приложения.

Полученные знания применены на практике, в ходе разработки фреймворка автоматизированного тестирования для веб-приложения с возможностью запуска тестов на удаленном сервере. Применение рассмотренного решения повлекло за собой улучшение качества тестируемого программного продукта.

В перспективе данную тему можно развить в направлении распараллеливания запуска тестовых сценариев, для ускорения прохождения тестов.

Основные источники информации:

1. Макконнелл С. Совершенный код. Мастер-класс/ Пер. с англ. — М. : Издательско-торговый дом «Русская Редакция» ; СПб.: Питер, 2005. — 896 с
2. Пьюривал С. Основы разработки веб-приложений. — СПб.: Питер, 2015. — 272 с.: ил.
3. Куликов С.С. — Тестирование программного обеспечения. Базовый курс. — Минск: Четыре четверти, 2017 – 312 с
4. Лайза Криспин, Джанет Грегори. Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд = Agile Testing: A Practical Guide for Testers and Agile Teams. — М.: «Вильямс», 2010. — 464 с
5. Дюваль, Поль М., Матиас III, Стивен М., Гловер, Эндрю. Непрерывная интеграция: улучшение качества программного обеспечения и снижение риска. : Пер. с англ. — М. : ООО “И.Д. Вильямс”, 2008. — 240 с. : ил. — Парал. тит. англ.