МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра Прикладная математика и информатика
(наименование)

09.04.03 Прикладная информатика
(код и наименование направления подготовки)

Управление корпоративными информационными процессами
(направленность (профиль))

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ)

на тему «Модели и алгоритмы интегрированной среды автоматизированного тестирования программного обеспечения»

Обучающийся И.А. Андрианкин
(Инициалы Фамилия) (личная подпись)

Научный руководитель к.т.н., доцент, О.В. Аникина
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2024

Оглавление

Введение

Проектирование качественного программного обеспечения (ПО) связано с использованием большого количества тестов, разработка и выполнение которых представляет собой медленный и трудоемкий процесс.

Как показывает практика, высокая эффективность процесса тестирования ПО достигается с помощью его автоматизации, которая помимо повышения производительности позволяет снизить негативное влияние человеческого фактора на результаты тестирования. Автоматизированное тестирование ПО предлагает применение инструментария, используя который тестировщики ИТ-компании не будут тратить время и другие ресурсы на монотонные задачи, так как автоматизированные тесты могут выполняться непрерывно или по расписанию через определенные промежутки времени.

Для решения данной задачи используются интегрированные среды автоматизированного тестирования ПО.

Разработка моделей и алгоритмов такой среды является актуальной и представляет научно-практический интерес.

Объектом настоящего исследования является интегрированная среда автоматизированного тестирования ПО.

Предметом исследования являются модели и алгоритмы интегрированной среды автоматизированного тестирования ПО.

Целью работы является исследование и разработка моделей и алгоритмов интегрированной среды автоматизированного тестирования ПО, обеспечивающей повышение эффективности процесса тестирования ПО.

Для достижения цели необходимо решить следующие задачи:

- произвести обзор и анализ существующих решений интегрированных автоматизированных сред тестирования ПО;
- проанализировать методологические подходы к построению интегрированных автоматизированных сред тестирования ПО;
- разработать модели и алгоритмы интегрированной среды

автоматизированного тестирования ПО;

- выполнить апробацию и оценить эффективность интегрированной среды автоматизированного тестирования ПО, реализованной на основе разработанных моделей и алгоритмов.

Гипотеза исследования: применение интегрированной среды автоматизированного тестирования ПО, реализованной на основе разработанных в рамках диссертационного исследования моделей и алгоритмов, обеспечит повышение эффективности процесса тестирования ПО.

Методы исследования. В процессе исследования использованы: программная инженерия, методологии и технологии проектирования информационных систем.

Новизна исследования заключается в разработке моделей и алгоритмов интегрированной среды автоматизированного тестирования, обеспечивающей автоматизацию тестирования ПО.

Практическая значимость исследования заключается в возможности практического применения разработанной интегрированной среды автоматизированного тестирования для повышения эффективности тестирования ПО в ИТ-компаниях.

Основные этапы исследования: исследование проводилось с 2020 по 2023 год в несколько этапов.

На первом (констатирующем) этапе формулировалась тема исследования, выполнялся сбор информации по теме исследования из различных источников, проводилась формулировка гипотезы, определялись постановка цели, задач, предмета исследования, объекта исследования и выполнялось определение проблематики данного исследования. Второй этап – теоретический. В ходе проведения данного этапа осуществлялся анализ методологических подходов к построению интегрированных автоматизированных сред тестирования ПО, разработаны модели и алгоритмы интегрированной среды автоматизированного тестирования ПО, опубликована статья по теме исследования в научном сборнике. Третий этап

– практический. В ходе проведения данного этапа проводилась апробация предлагаемых проектных решений, произведена оценка их эффективности, сформулированы выводы о полученных результатах по проведенному исследованию.

На защиту выносятся:

- модели и алгоритмы интегрированной среды автоматизированного тестирования ПО;
- результаты апробации и оценки эффективности предлагаемых проектных решений.

По теме исследования опубликована 1 статья:

Андрианкин И.А. Модель интегрированной среды автоматизированного тестирования программного обеспечения // Вестник научных конференций. 2024 (принята к публикации).

Диссертация состоит из введения, четырех глав, заключения и списка литературы.

Во введении обоснована актуальность темы исследования, представлены объект, предмет, цели, задачи и положения, выносимые на защиту диссертации. В первой главе дан анализ состояния исследований в области построения интегрированных автоматизированных сред тестирования ПО. Во второй главе дан анализ методологических подходов к построению интегрированных автоматизированных сред тестирования ПО.

Третья глава посвящена разработке моделей и алгоритмов интегрированной среды автоматизированного тестирования ПО. В четвертой главе выполнены апробация предлагаемых проектных решений и оценка их эффективности.

В заключении приводятся результаты исследования.

Работа изложена на 74 страницах и включает 36 рисунков, 14 таблиц и 39 источников.

Глава 1 Анализ современного состояния исследований в области автоматизации тестирования программного обеспечения

1.1 Анализ методов автоматизированного тестирования

«Автоматизированное тестирование – это метод тестирования (ПО), который выполняется с использованием специальных программных средств для выполнения набора тестовых примеров.

Проблематика автоматизации тестирования ПО рассмотрена в работах Э. Дастина, М. Кона, М. Фаулера, специалистов компаний-вендоров ПО и др.

Общепринятым считается следующее определение автоматизированного тестирования ПО - это процесс верификации ПО, при котором основные функции и шаги теста, такие как запуск, инициализация, выполнение, анализ и выдача результата, выполняются автоматически при помощи инструментов для автоматизированного тестирования» [1].

«Автоматизация тестирования обеспечивает преимущества:

- экономический эффект от уменьшения затрат по сравнению с выполнением ручного тестирования;
- улучшение качества за счет исключения человеческого фактора, повторного использования автотестов на разных этапах разработки, регулярной проверки функциональности в процессе разработке и того, что QA-специалисты смогут уделить больше внимания проверкам, не покрытым автотестами;
- оптимизация объема тестирования – возможность проверить больший объем функциональности за то же время, а также применить более креативные методы тестирования;
- сокращение времени тестирования – регулярный запуск регрессионных тестов позволяет быстро обнаруживать старые дефекты, а уменьшение времени за счет автоматизации дает возможность быстрее выводить программный продукт на рынок.

Автоматизированное тестирование предложено М. Коном, который представил процесс автоматизации системы и проверки ПО в форме пирамиды.

Структура пирамиды тестирования изображена на рисунке 1» [22].

Рисунок 1 – Пирамида автоматизированного тестирования ПО

Согласно данному подходу, эффективная стратегия автоматизации тестирования требует обеспечение автоматизации тестов на трех разных уровнях.

Уровень тестирования – это активности тестирования, объединенные в группу исходя из общих характеристик, связанных с жизненным циклом разработки ПО [38].

В основе пирамиды модульное тестирование.

Модульное тестирование должно быть основой надежной стратегии автоматизации тестирования и поэтому представляет собой самую большую часть пирамиды. Автоматические модульные тесты эффективны, потому что они предоставляют конкретные данные программисту. Кроме того, поскольку модульные тесты обычно пишутся на том же языке, что и система, программистам часто удобнее их писать.

Интеграционное тестирование - это способ тестирования программного

обеспечения путем группировки программных компонентов.

«Автоматизированное интеграционное тестирование проводится для оценки соответствия приложения или его компонентов заданным функциональным требованиям.

Автоматизированное сквозное тестирование (end-to-end или E2E-тестирование) – это тестирование пользовательского интерфейса, которое находится на вершине пирамиды. Следует отметить, что сквозное тестирование стоит дорого и должно быть сведено к минимуму.

Элементы пирамиды Кона отличаются друг от друга объемами тестирования, поскольку они содержат различное количество тестовых случаев, необходимых для выполнения того или иного типа тестирования.

Представленная пирамида типична для тестирования ПО. Но есть много ее модификаций - все зависит от типа тестируемого приложения» [6].

Веб-приложения динамичны и интерактивны по сравнению с традиционными приложениями. Следовательно, традиционных методов и инструментов тестирования недостаточно для тестирования веб-приложений.

Следовательно, эффективность работы команды тестирования во многом зависит от того, какие именно задачи было решено автоматизировать и как эта автоматизация была проведена.

1.2 Обзор и анализ источников по теме исследования

Интегрированная среда автоматизированного тестирования – это комплекс средств автоматизации тестирования, интегрированных в среду разработки ПО.

Girgis M.R. и др. предлагают подход к веб-тестированию, при котором гиперссылки тестируемого веб-сайта автоматически следуют одна за другой, чтобы получить все HTML-тексты его страниц, начиная с домашней страницы. HTML-текст каждой обнаруженной страницы анализируется для извлечения необходимой информации о ней. Затем собранная информация используется в

процессе проверки ошибок [28].

Архитектура автоматизированной системы тестирования, реализующей описанный подход, представлена на рисунке 2.

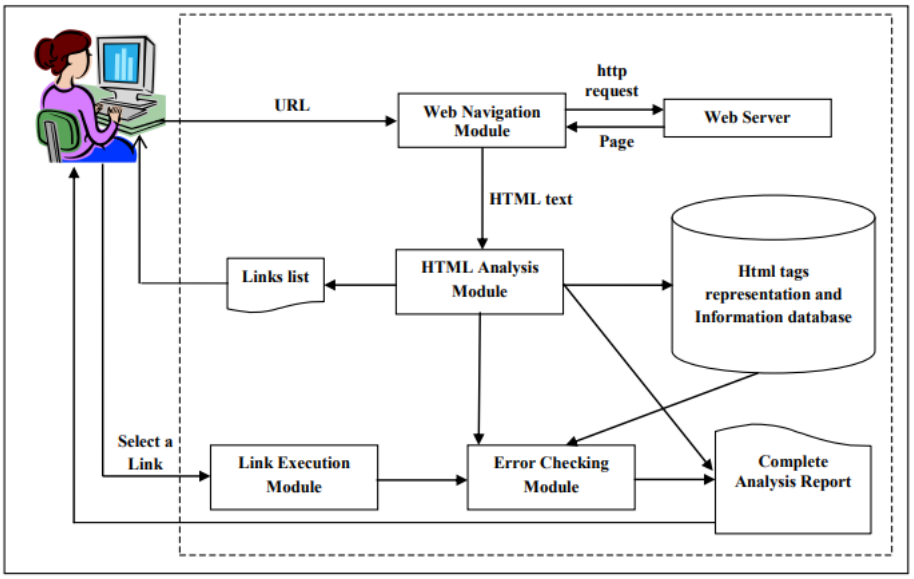

Рисунок 2 – Архитектура автоматизированной системы тестирования веб-приложений

По мнению авторов решения, предлагаемый подход гарантирует соответствие двум критериям тестирования веб-приложений, а именно критерию покрытия страницы и критерию покрытия гиперссылок.

Во многих решениях для автоматизации тестирования используются фреймворки.

«Фреймворк – это программная платформа, определяющая структуру программной системы и ПО, облегчающее разработку и объединение разных компонентов большого программного проекта.

Фреймворк автоматического тестирования – это набор условий, концепций и практик, направленный на повторное использование, уменьшение затрат на поддержку и повышение надежности использования тестов» [18].

Как отмечает М. Фаулер, «пирамида тестирования также представляет

промежуточный уровень тестов, которые действуют через сервисный уровень приложения. Они могут обеспечить многие преимущества сквозных тестов, но позволяют избежать многих сложностей работы с UI-фреймворками.

В веб-приложениях это будет соответствовать тестированию через слой API, в то время как верхняя часть пирамиды пользовательского интерфейса будет соответствовать тестам, использующим что-то вроде Selenium или Sahi» [26].

B. Garcıa и J.C. Duenas предлагают метод автоматизации тестирования, который выполняется на четырех уровнях: генерация тестового примера, получение тестовых данных, выполнение тестового примера и составление отчетов по тест-кейсу.

В основе этого метода лежат три типа входных данных: модели UML, скрипты фреймворка Selenium и XML-файлы. Подход реализован в среде тестирования с открытым исходным кодом под названием Automatic Testing Platform [27].

В исследовании M. Kalmo описан эксперимент оценки влияния фрейворков на эффективность тестирования веб-приложений, разработанных на языке Java [32].

Суть этого эксперимента состоит в том, чтобы собраны два прототипа приложения для автоматизации тестирования - с поддержкой фреймворка и без него. Как показал сравнительный анализ, применение фреймворков существенно повышает эффективность тестирования веб-приложений на всех уровнях пирамиды тестирования.

M. Hanna и др. предлагают фреймворк автоматизированного тестирования для веб-приложений, который, по их мнению, улучшит процесс автоматизации (рисунок 3) [29].

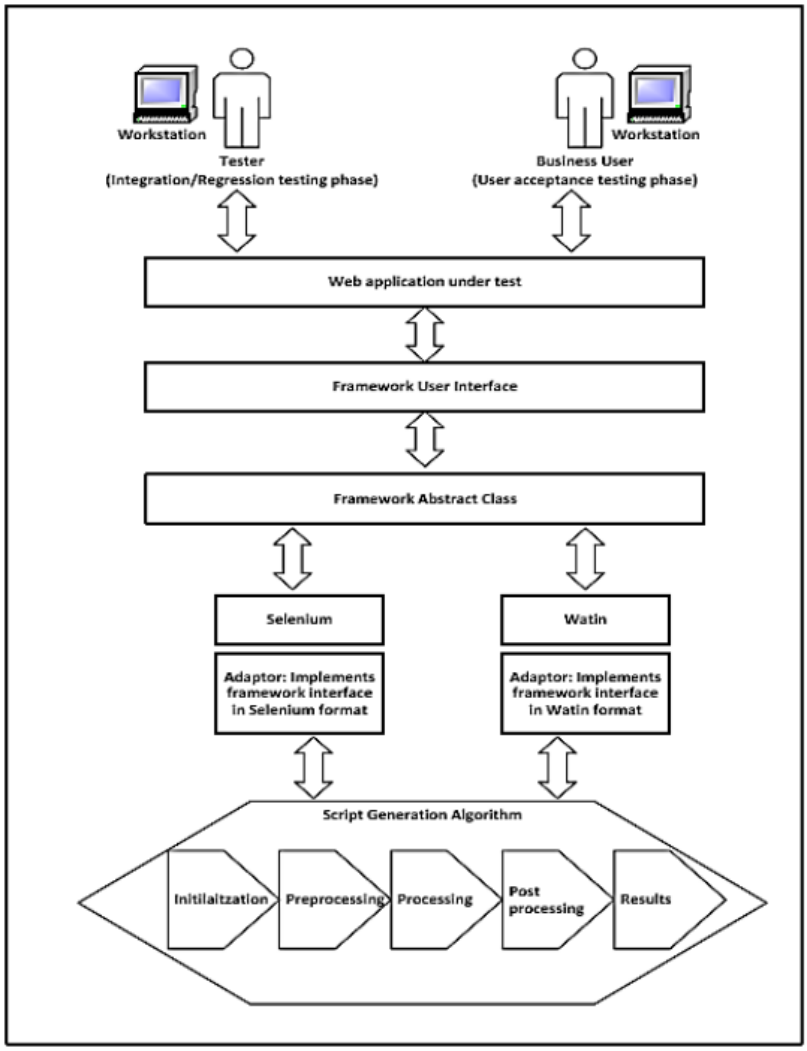

Рисунок 3 – Модель фреймворка для тестирования веб-приложений

Таким образом, наиболее эффективным подходом к автоматизации тестирования веб-приложений является использование специализированных фреймворков.

В работе [7] описана интегрированная среда автоматизации тестирования, построенная на основе технологии Eclips.

«Такое решение позволяет обеспечить платформонезависимость и максимально упростить работы по настройке и сопровождению. Интегрированная среда состоит из следующих модулей (рисунок 4):

− парсера, предназначенного для импорта и экспорта данных по настройке тестового окружения из внутренней модели данных;

модели данных, позволяющей хранить информацию из конфигурационного файла в специализированной модели среды Eclipse;

- пользовательского интерфейса, основанного на применении технологии GEF (Graphical Editing Framework), дающего возможность быстрого ввода и редактирования данных;
- менеджера запуска – модуля, позволяющего легко собирать произвольные цепочки запуска инструментов TAT и вспомогательных скриптов» [7].

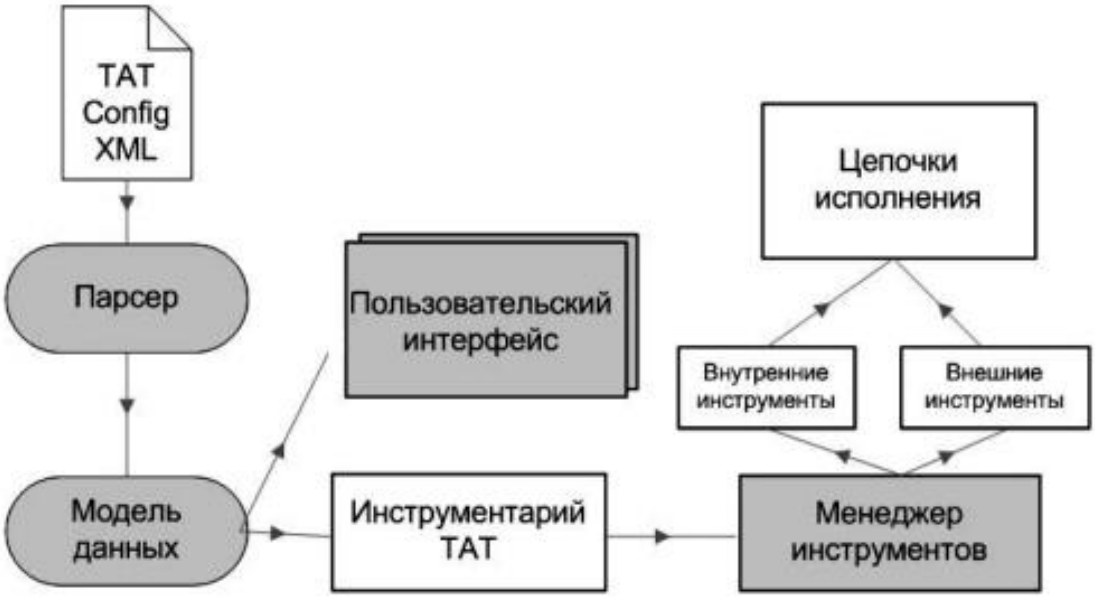

Рисунок 4 – Архитектура интегрированной среды автоматизации тестирования

Вместе с тем, необходимо констатировать недостаточность работ, посвященных разработке моделей и алгоритмов интегрированной среды автоматизированного тестирования ПО, что подтверждает актуальность темы исследования.

Выводы по главе 1

В результате проделанной работы были сделаны следующие выводы:

- проблематика автоматизированного тестирования ПО с помощью современных информационных технологий широко рассмотрена в работах отечественных и зарубежных ученых;
- эффективная стратегия автоматизации тестирования требует обеспечение автоматизации тестов на трех разных уровнях: модульном, интеграционном и E2;
- эффективность работы команды тестирования во многом зависит от того, какие именно задачи было решено автоматизировать и как эта автоматизация была проведена;
- наиболее эффективным подходом к автоматизации тестирования веб-приложений является использование специализированных фреймворков.

Необходимо констатировать недостаточность работ, посвященных разработке моделей и алгоритмов интегрированной среды автоматизированного тестирования ПО, что подтверждает актуальность темы исследования.

Глава 2 Анализ методологий и технологий построения интегрированных сред автоматизированного тестирования программного обеспечения

2.1 Обзор и анализ аналогов интегрированной среды автоматизированного тестирования программного обеспечения.

«По своим функциональным и архитектурным особенностям интегрированная среда автоматизированного тестирования (ИСАТ) ПО относится к средам автоматизированного тестирования» [5].

Среда автоматизированного тестирования (САТ) - это комбинация оборудования, программного обеспечения, данных и конфигурации, необходимых для выполнения тестовых примеров.

Для разработки требований к ИС используем методологию FURPS+.

«FURPS+ – это метод проверки приоритетных требований после понимания потребностей клиента.

Методология FURPS+ в классификации требований делает упор на понимание различных типов функциональных и нефункциональных требований.

Функциональные требования описывают, что должна делать ИС, в то время как нефункциональные требования накладывают ограничения на то, как система ИС будет это делать» [9].

В таблице 1 представлены основные требования к ИСАТ ПО в методологии FURPS.

Таблица 1 – Требования к ИС тестирования веб-приложений

№	Требование	Статус	Полезность	Риск	Стабильность
Functionality — Функциональные требования					
1.	«Обеспечение автоматизации тестов на 3-х уровнях тестирования веб-приложений	Одобренное	Критическая	Средний	Низкая
Usability— Требования к удобству использования					
2.	Дружественный интуитивный интерфейс	Одобренное	Критическая	Средний	Низкая
Reliability— Требования к надежности					
3.	Допустимая частота/периодичность сбоев: 1 раз в 300 часов	Одобренное	Важная	Средний	Средняя
4.	Среднее время сбоев: 1 раб. день	Одобренное	Важная	Средний	Средняя
5.	Возможность восстановления системы после сбоев: 1 раб. день	Одобренное	Важная	Средний	Средняя
6.	Режим работы: 7/24/365	Одобренное	Важная	Средний	Средняя
Performance — Требования к производительности					
7.	Допустимое количество одновременно работающих пользователей: 1	Предложенное	Важная	Средний	Средняя
8.	Время реакции на возникновение аварийной ситуации: 10 с	Предложенное	Важная	Средний	Средняя
Supportability — Требования к поддержке					
9.	Время устранения критических проблем: в течение рабочего дня	Предложенное	Важная	Средний	Средняя
Проектные ограничения					
10.	Применение современных ИТ	Предложенное	Критическая	Средний	Низкая
11.	Низкая совокупная стоимость владения	Предложенное	Критическая	Средний	Низкая» [9]

Разработанный набор требований является основой для реализации проектного решения по разработке ИСАТ ПО.

В процессе анализа известных решений САТ были выявлены следующие средства автоматизированного тестирования веб-приложений:

- Sahi Pro;
- Selenium IDE;
- Watir.

Рассмотрим характеристики данных решений.

«Sahi Pro – это набор зрелых, готовых к работе инструментов для автоматического тестирования веб-приложений, веб-сервисов, мобильных устройств, Windows Desktop, SAP GUI и приложений Java. По мнению вендора, для команд тестирования, которым нужна быстрая и надежная автоматизация, Sahi Pro станет лучшим выбором среди инструментов автоматизации» [12].

Sahi Pro (последняя версия Sahi Pro v9.5.0) помогает автоматизировать функциональное тестирование веб-приложений. Sahi Pro по умолчанию поддерживает автоматизацию веб-приложений и REST API.

Sahi использует прокси для вставки JavaScript в веб-страницы (рисунок 5).

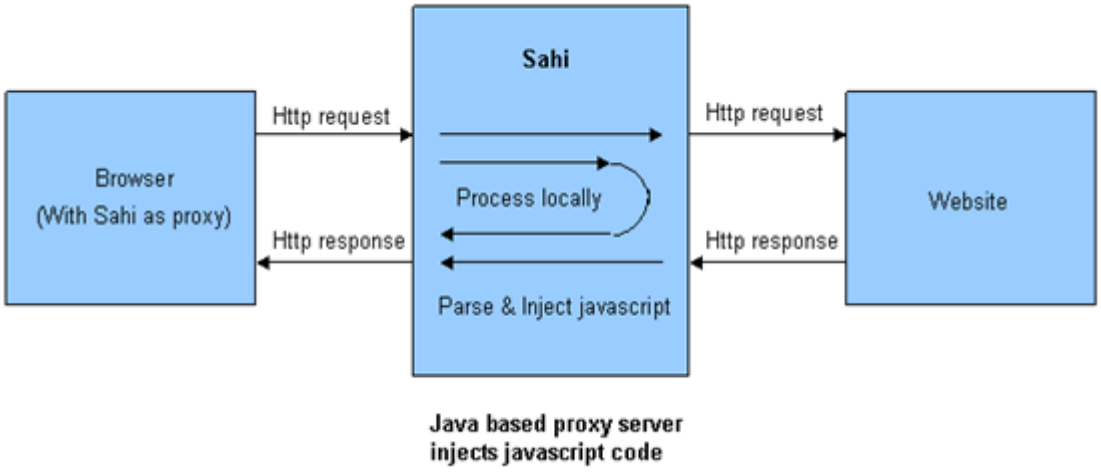

Рисунок 5 – Архитектура САТ Sahi Pro

Sahi Pro хорошо подходит для кросс-браузерного / мульти-браузерного тестирования сложных приложений Web 2.0 с большим количеством AJAX и динамического контента. Sahi Pro работает в любом современном браузере, поддерживающем javascript.

Sahi поддерживает модульное тестирование, ExtJS, ZK, Dojo, YUI и др. фреймворки.

Selenium IDE - простое в использовании расширение к браузеру Firefox, которое помогает разрабатывать тестовые сценарии веб-страниц [13].

Он записывает определенный сценарий поведения пользователя на сайте, а потом воспроизводит записанные действия в автоматическом режиме.

Selenium IDE (доступная версия 3.17) также обладает более 500-ми встроенными командами, которые позволяют зафиксировать практически любые ваши действия на сайте и воспроизвести их вновь или осуществить проверку какого-то элемента.

Selenium IDE можно расширить с помощью плагинов. Они могут вводить новые команды в IDE или интегрироваться со сторонней службой.

Поток работ в Selenium IDE показан на рисунке 6.

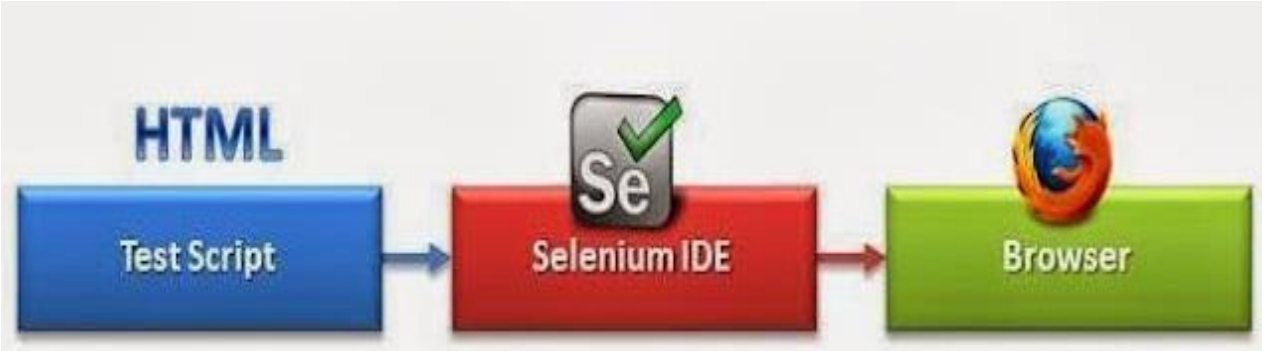

Рисунок 6 – Поток работ в Selenium IDE

Основные достоинства:

- простое готовое решение для быстрого создания надежных сквозных тестов. Готово к работе с любым веб-приложением;

- простая тестовая отладка с помощью богатых функций IDE, таких как установка точек останова и приостановка при исключениях;
- возможность параллельного запуска тестов в любой комбинации браузера / ОС, используя средство выполнения командной строки.

Недостатки:

- излишняя простота автотестов;
- ограниченное применение (для браузеров Firefox и Chrome).

Watir (последняя версия – 6.17) – это инструмент тестирования с открытым исходным кодом, разработанный с использованием Ruby, который используется для тестирования веб-приложений, разработанных на любом языке программирования [14].

Архитектура Watir показана на рисунке 7.

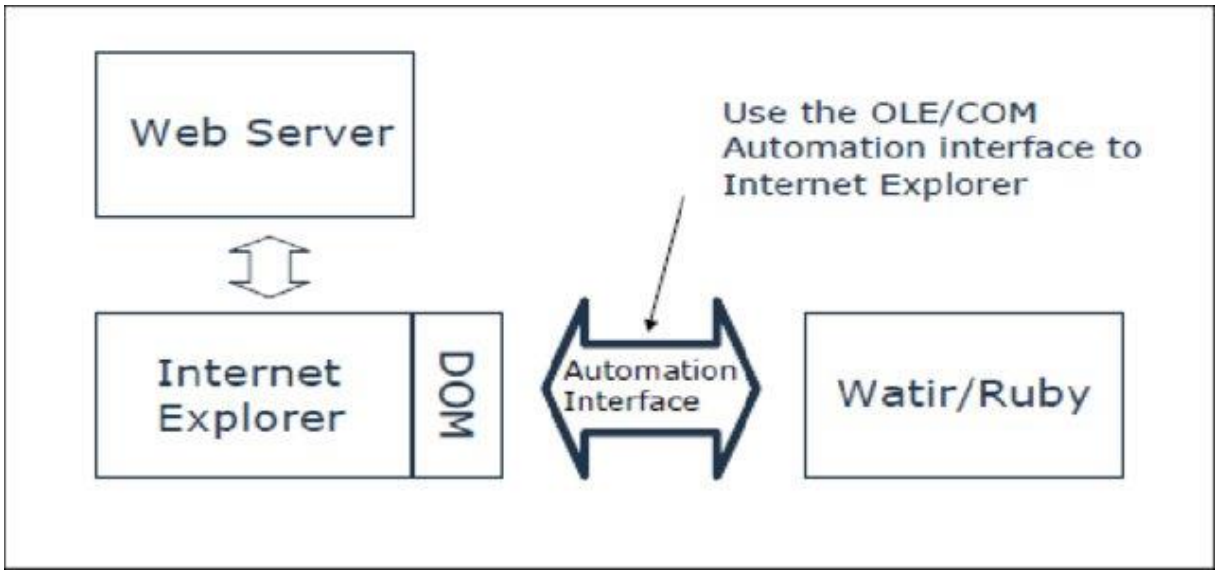

Рисунок 7 – Архитектура CAT Watir

Основные функции:

- тестирует языковое веб-приложение;
- кросс-браузерное тестирование;
- совместим с бизнес-инструментами разработки, такими как RSpec, Cucumber и Test / Unit;

− проверяет кнопки, формы, ссылки и их ответы на веб-страницах.

Основной недостаток - Watir поддерживает только среды тестирования Ruby и не может использоваться с другими системами тестирования.

Для сравнительного анализа рассмотренных выше решений САТ на предмет соответствия требованиям к ИСАТ ПО используем таблицу 2.

Таблица 2 – Сравнительный анализ аналогов САТ

Характеристика/балл (0-3)	Sahi Pro	Selenium IDE	Watir
«Обеспечение автоматизации тестов на 3-х уровнях тестирования веб-приложений	2	2	2
Применение современных ИТ	3	3	3
Низкая совокупная стоимость владения	2	3	3
Итого	7	8	8» [36]

Таким образом, ни одно из рассмотренных готовых решений САТ не соответствует всем установленным требованиям, что подтверждает целесообразность разработки ИСАТ ПО.

2.2 Выбор методологии проектирования интегрированной среды автоматизированного тестирования программного обеспечения

Согласно концепции гибкого проектирования (Agile), автоматизированное тестирование должно быть не изолированной задачей, а непрерывным процессом, неотъемлемо вписанным в жизненный цикл ПО.

В общем, стратегия автоматического тестирования включает в себя общий подход к тестированию и отчетности. Все процессы изложены и запланированы вместе с ожиданиями от цикла тестирования [21].

Стратегия автоматизации тестирования веб-приложений состоит из следующих этапов:

Этап 1. Выбор фреймворка тестирования для каждого уровня

автоматизации тестирования. Иными словами, необходимо выбрать и очертить диапазон используемых инструментов, а также их объем. При создании CAT разработчик должен обеспечить ее масштабируемость. Следует учесть, что фреймворк редко используется для одного проекта - вместо этого он представляет собой шаблон стратегии автоматизации тестирования, руководящих принципов и принципов, которые тестировщики должны уметь использовать для любого вида автоматизированного тестирования.

Этап 2. Разработка среды автоматизированного тестирования. Хорошая CAT должна, прежде всего, полностью контролироваться и предоставлять тестировщику свободу изменять все необходимые данные.

Этап 3. Настройка CAT. Это та часть, где тестировщик определяет объем теста, создает список действий и назначает членов команды для наблюдения за каждым кластером. Стандартный функциональный набор автоматизированного тестирования обычно выглядит следующим образом: «Инициирование», «Автоматическое планирование тестирования», «Выполнение», «Отчетность», «Завершение».

Этап 4. Внедрение CAT в процесс тестирования веб-приложений ИТ-компании.

Хотя структура стратегии в значительной степени зависит от структуры ИТ-компании, количества задействованных людей и методологии проектирования, принятой в ИТ-компании, количество этапов и действий, которые необходимо предпринять, остается довольно стабильным.

Рассмотрим и сравним известные фреймворки по уровням автоматизации тестирования веб-приложений.

2.2.1 Модульное тестирование программного обеспечения

Для выбора фреймворка для модульного тестирования сравним характеристики фреймворков Jasmine, Mocha и JUnit.

«Фреймворк Jasmine – это поведенческая среда разработки для тестирования кода JavaScript. Jasmine не зависит от других JavaScript-фреймворков и не требует DOM.

У него понятный синтаксис, позволяющий относительно просто писать тесты.

Jasmine чаще всего используется для тестинга асинхронного кода, но у него много возможностей. Работает на Node.js и составляет точные и гибкие отчеты, периодически запуская тест-кейсы» [3].

Преимущества:

- простота: Jasmine известен своим простым синтаксисом и простотой использования. Это отличный выбор для небольших наборов тестов, где простота имеет значение;
- разработка на основе поведения (BDD): Jasmine следует подходу BDD, позволяя разработчикам и тестировщикам писать выразительные тестовые примеры, очень похожие на естественный язык. Такая читабельность улучшает сотрудничество между членами команды;
- наборы тестов и спецификации. Жасмин объединяет тесты в наборы и спецификации. Такая иерархическая структура помогает поддерживать четкое разделение задач и упрощает управление и поиск конкретных тестов;
- интегрированное моделирование и утверждение. Хотя Jasmine требует внешних библиотек для создания макетов и утверждений, он обеспечивает прочную основу для написания тестовых примеров. Встроенная функция SpyOn позволяет имитировать вызовы, что делает ее полезной для тестирования асинхронного кода;
- поддержка сообщества. У Jasmine активное сообщество, а это значит, что вы можете найти множество ресурсов, учебных пособий и плагинов для улучшения вашего опыта тестирования.

Недостатки:

- внешние зависимости: Jasmine использует внешние библиотеки (например, Chai) для покрытия всех аспектов тестирования. Хотя эта модульность обеспечивает гибкость, она также означает

дополнительную настройку и обслуживание;

- ограниченные возможности. По сравнению с некоторыми другими фреймворками, такими как Jest, в Jasmine отсутствуют определенные встроенные функции. Например, Jest включает в себя такие функции, как отчеты о покрытии кода, тестирование моментальных снимков и параллельное выполнение тестов;
- медленное выполнение. Скорость выполнения тестов Jasmine может быть медленнее, чем у Jest, из-за того, что он зависит от внешних библиотек. Однако эта разница может быть незначительной для небольших проектов.

Подводя итог, можно сказать, что Jasmine — хороший выбор для простых наборов тестов и сценариев, где читаемость и принципы BDD имеют решающее значение. Однако рассмотрите другие платформы, такие как Jest или Mocha, для более сложных проектов или когда требуются дополнительные функции.

«Фреймворк Mocha - это многофункциональный тестовый фреймворк JavaScript, работающий на Node.js и в браузере, что делает асинхронное тестирование простым и увлекательным.

Тесты Mocha запускаются последовательно, обеспечивая гибкую и точную отчетность и отображая не перехваченные исключения в правильные тестовые случаи. Размещен на GitHub.

Mocha можно использовать с большинством известных библиотек утверждений JavaScript, включая Chai.

Тесты в Mocha имеют улучшенное качество трассировки исключений и могут прогоняться сериями» [2].

Преимущества:

- гибкость и настройка: Mocha предоставляет гибкую архитектуру, позволяющую разработчикам адаптировать свою среду тестирования в соответствии с потребностями конкретного проекта. Он не навязывает жесткой структуры, что делает его подходящим

для различных сценариев;

- внешнее тестирование: Mocha превосходно тестирует внутренний код. Его универсальность позволяет легко интегрировать его с другими библиотеками (такими как Chai для утверждений и Sinon для макетирования) для создания комплексного пакета тестирования;
- богатая экосистема: Mocha имеет обширную экосистему плагинов и расширений. Разработчики могут выбирать из множества генераторов отчетов, библиотек утверждений и других инструментов для улучшения своего опыта тестирования.

Недостатки:

- накладные расходы на настройку. В отличие от некоторых фреймворков, таких как Jest, которые имеют встроенные функции, Mocha требует дополнительной настройки. Разработчикам необходимо вручную настроить библиотеки утверждений и другие зависимости;
- кривая обучения. Гибкость Mocha достигается за счет более крутой кривой обучения. Новичкам на начальном этапе может быть сложно разобраться во всех тонкостях;
- сдвиг популярности. Рост популярности Jest повлиял на популярность Mocha. Jest, будучи единой интегрированной средой, набирает обороты, в то время как экосистема Mocha, хотя и обширна, может быть сложной.

Таким образом, выберите Mocha, если вы цените настройку и нуждаетесь в мощной среде тестирования для серверных проектов. Однако рассмотрите другие варианты, такие как Jest, для более простой настройки и более быстрого выполнения.

«Фреймворк JUnit — это инфраструктура модульного тестирования для языка программирования Java. Она играет решающую роль при разработке на основе тестов и представляет собой семейство платформ модульного

тестирования, известных под общим названием xUnit.

JUnit продвигает идею «сначала тестирование, а затем кодирование», которая делает упор на настройке тестовых данных для фрагмента кода, который можно сначала протестировать, а затем реализовать. Этот подход похож на «немного протестировать, немного кодировать, немного протестировать, немного кодировать». Это увеличивает производительность программиста и стабильность программного кода, что, в свою очередь, снижает нагрузку на программиста и время, затрачиваемое на отладку» [31].

Преимущества:

- открытый исходный код: JUnit — это платформа с открытым исходным кодом, что делает ее доступной для разработчиков без каких-либо затрат на лицензирование. Его развитие по инициативе сообщества обеспечивает постоянное улучшение;
- гибкое тестирование: JUnit позволяет писать тестовые примеры для различных сценариев, включая модульные тесты, интеграционные тесты и системные тесты. Его гибкость позволяет удовлетворить различные потребности в тестировании;
- читабельность и выразительность: JUnit следует простому синтаксису, что делает тестовые примеры читабельными и выразительными. Использование аннотаций (таких как @Test, @Before и @After) повышает ясность кода;
- автоматическое тестирование: JUnit автоматизирует выполнение тестов, обеспечивая быструю обратную связь во время разработки. Если тесты терпят неудачу, это четко указывает на проблему, позволяя разработчикам оперативно ее устранить;
- графический интерфейс и интерфейс командной строки: JUnit предлагает как текстовые интерфейсы командной строки, так и графические интерфейсы (AWT и Swing). Разработчики могут выбрать предпочтительный режим запуска тестов.

Недостатки:

- ограниченное тестирование зависимостей. В отличие от таких фреймворков, как TestNG, в JUnit отсутствует встроенная поддержка тестирования зависимостей. Разработчикам приходится управлять зависимостями вручную;
- не идеален для больших наборов тестов: JUnit, возможно, не лучший выбор для обширных наборов тестов. Его простота проявляется в небольших проектах, но для более крупных приложений рассмотрите альтернативы, такие как TestNG;
- нет группового тестирования. В JUnit отсутствует встроенная поддержка группировки тестов — функция, доступная в TestNG. Группировка помогает организовывать и выполнять связанные тесты вместе;
- отсутствие отчетов в формате HTML. JUnit не создает отчеты в формате HTML для тестовых случаев, что может быть полезно для визуального отслеживания результатов тестирования.

Подводя итог, можно сказать, что JUnit — надежный выбор для модульного тестирования на Java, особенно для небольших проектов. Однако рассмотрите другие платформы, если вам нужны более продвинутые функции или расширенное управление набором тестов.

Для сравнительного анализа характеристик инструментов автоматизации модульного тестирования веб-приложения используем таблицу 3, составленную на основе опроса разработчиков и тестировщиков.

Таблица 3 – Сравнительный анализ фреймворков модульного тестирования веб-приложений

Характеристика (0-5)	JUnit	Jasmine	Mocha
«Простота настройки	5	3	2
Простота освоения	3	5	4
Комьюнити	4	5	3
Функциональность	5	4	3» [36]

Продолжение таблицы 3

Характеристика (0-5)	JUnit	Jasmine	Mocha
«Документация	5	5	4
Производительность	5	2	2
Итого	27	24	18» [36]

Как показал анализ, наилучшие характеристики у фреймворка JUnit, что вполне объясняет его популярность среди тестировщиков. Еще одним важным преимуществом JUnit является простота интеграции в различные инструменты разработчика (например, IntelliJ).

2.2.2 Интеграционное тестирование программного обеспечения

Для выбора фреймворка для модульного тестирования сравним характеристики фреймворков JWebUnit, Cucumber и Protractor.

Фреймворк JWebUnit – это фреймворк тестирования на основе Java и одно из расширений JUnit, используемый для интеграционного, регрессионного и функционального тестирования. Он обертывает активные в данный момент фреймворки в простой тестовый интерфейс. Благодаря этому можно сразу проверить свои веб-приложения.

Преимущества:

- создание приемочных тестов: JWebUnit специально разработан для создания приемочных тестов для веб-приложений. Он возник из проекта, в котором разработчики использовали HttpUnit и JUnit для создания приемочных тестов. По мере написания тестов они постоянно подвергались рефакторингу для устранения дублирования и улучшения качества тестового кода;
- интеграция с HtmlUnit: JWebUnit использует HtmlUnit (из htmlunit.sourceforge.net) для веб-взаимодействия. HtmlUnit имитирует среду браузера, позволяя JWebUnit взаимодействовать с веб-страницами, отправлять формы и проверять ответы;
- читаемость и рефакторинг. Как и в случае с рефакторингом,

JWebUnit обеспечивает читаемый тестовый код. Устраняя дублирование и неприятные запахи, он способствует удобству и выразительности приемочных тестов.

Недостатки:

- внешние зависимости: JWebUnit использует внешние библиотеки, такие как HtmlUnit и JUnit. Хотя эта модульность обеспечивает гибкость, она также означает дополнительную настройку и обслуживание;
- кривая обучения. Разработчики, впервые работающие с JWebUnit, могут столкнуться с необходимостью обучения из-за его специфической интеграции с HtmlUnit и необходимости понимать концепции приемочного тестирования;
- сдвиг популярности: появление других сред тестирования (таких как Selenium и TestNG) повлияло на популярность JWebUnit. Рассмотрите альтернативы, основанные на требованиях проекта.

Таким образом, JWebUnit является подходящим выбором для приемочного тестирования веб-приложений Java, особенно когда желательна интеграция с HtmlUnit. Однако оцените другие платформы, исходя из ваших конкретных потребностей и контекста проекта.

Фреймворк Cucumber – это фреймворк, реализующий подход BDD/TDD.

Преимущества:

- разработка на основе поведения (BDD): Cucumber — это платформа BDD, которая поощряет сотрудничество между разработчиками, тестировщиками и заинтересованными сторонами, не являющимися техническими специалистами. Его файлы функций в виде простого текста позволяют четко сообщать о требованиях и ожидаемом поведении;
- читаемый и выразительный синтаксис. Сценарии Cucumber записываются в удобочитаемом формате с использованием языка Gherkin. Это упрощает понимание тестовых случаев даже для

нетехнических членов команды;

- повторное использование: Cucumber способствует повторному использованию, позволяя использовать определения шагов в разных сценариях. Такая модульность уменьшает дублирование и повышает удобство обслуживания;
- интеграция с различными языками: Cucumber поддерживает несколько языков программирования (таких как Java, Ruby и JavaScript), что делает его универсальным для различных стеков технологий;
- отчеты о тестировании и визуализация: Cucumber создает подробные отчеты о тестировании, включая информацию о пройденных и неудавшихся сценариях. Такие инструменты, как Cucumber Reports, улучшают визуализацию.

Недостатки:

- накладные расходы на настройку и кривую обучения**. Настройка Cucumber включает в себя настройку файлов функций, определений шагов и перехватчиков. Новичкам может показаться, что кривая обучения крутая из-за уникального синтаксиса и концепций;
- медленное выполнение. Сценарии Cucumber могут выполняться медленнее по сравнению с традиционными модульными тестами. Синтаксический анализ естественного языка и дополнительные уровни приводят к увеличению накладных расходов;
- проблемы обслуживания. По мере роста проекта обслуживание сценариев Cucumber усложняется. Поддерживать актуальность определений шагов и управлять файлами функций может быть непросто;
- ограниченное параллельное выполнение: Cucumber по своей сути не поддерживает параллельное выполнение сценариев. Командам необходимо внедрять индивидуальные решения для параллельного тестирования.

Подводя итог, можно сказать, что Cucumber ценен с точки зрения совместной работы, удобочитаемости и практики BDD. Однако при выборе его для нужд тестирования учитывайте трудоемкость установки и скорость выполнения. Вместе с тем, по мнению разработчиков, Cucumber является лучшим инструментом для интеграционного тестирования. Cucumber разработан для облегчения разработки, управляемой поведением.

«Фреймворк Protractor - это программа Node.js. Для запуска необходимо установить Node.js.

По умолчанию Protractor использует инфраструктуру тестирования Jasmine для своего интерфейса тестирования» [36].

Преимущества:

- создан на основе Selenium Webdriver**: Protractor работает поверх Selenium Webdriver, что означает, что он наследует все возможности Webdriver. Сюда входит взаимодействие с веб-элементами, обработка сеансов браузера и выполнение различных действий;
- функции, специфичные для Angular: Protractor специально разработан для тестирования приложений Angular. Он имеет встроенную поддержку для идентификации элементов, специфичных для Angular (таких как директивы и привязки), с использованием локаторов, таких как «модель», «повторитель» и «привязка»;
- легкое переключение между приложениями Angular и не Angular: Protractor легко обрабатывает как Angular, так и не Angular приложения. Разработчики могут переключаться между ними без серьезных изменений конфигурации;
- параллельное тестирование и поддержка кросс-браузерности: Protractor поддерживает параллельное выполнение тестов, что обеспечивает более быструю обратную связь. Это также облегчает кроссбраузерное тестирование в разных браузерах.

Недостатки:

- кривая обучения: Protractor требует обучения, особенно для новичков в тестировании Angular. Понимание концепций, специфичных для Angular, и уникальных функций Protractor может занять время;
- ограничено приложениями Angular. Несмотря на то, что Protractor превосходно тестирует приложения Angular, он может быть не лучшим выбором для проектов, отличных от Angular. Разработчикам, работающим над смешанными стеками технологий, следует рассмотреть альтернативы;
- ожидание Angular по умолчанию: Protractor ожидает стабилизации Angular перед выполнением действий. Хотя это полезно для приложений Angular, это может вызвать задержки при тестировании компонентов, отличных от Angular.

Подводя итог, можно сказать, что Protractor — мощный инструмент для тестирования Angular, но разработчикам следует оценивать его пригодность на основе требований проекта и знания концепций Angula.

Для удобства сравнения характеристики фреймворков сведены в таблицу 4.

Таблица 4 – Сравнительный анализ фреймворков интеграционного тестирования веб-приложений

Характеристика (1-5)	JWebUnit	Cucumber	Protractor
«Простота настройки	4	5	5
Простота освоения	4	4	4
Комьюнити	3	5	3
Функциональность	3	4	4
Документация	4	4	4
Производительность	4	4	5
Итого	22	26	25» [36]

Как показал анализ, наилучшими характеристиками для

интеграционного тестирования веб-приложений обладает фреймворк Cucumber.

2.2.3 E2E-тестирование программного обеспечения

Для выбора фреймворка для E2E-тестирования сравним характеристики фреймворков Selenium, Puppeteer и Cypress.

«Фреймворк Selenium Selenium WebDriver — это целый набор инструментов, позволяющий осуществлять браузерную автоматизацию. Отличительными чертами Selenium являются возможность написания сценариев на JavaScript, C#, Java, Ruby, Python и поддержки большинства современных браузеров (Chrome, Firefox, Safari, Edge).

Ключевым инструментов для работы с браузером является Selenium WebDriver.

Сценарии автотестов пишутся на одном из предпочитаемых языков, после чего Language-binding Selenium для конкретного языка транслирует команду в JSON и посылает ее (через HTTP) на Selenium server.

С помощью встроенного набора Browser Drivers осуществляет коммуникацию и контроль над браузером» [24].

Схема обмена данными Selenium WebDriver представлена на рисунке 8.

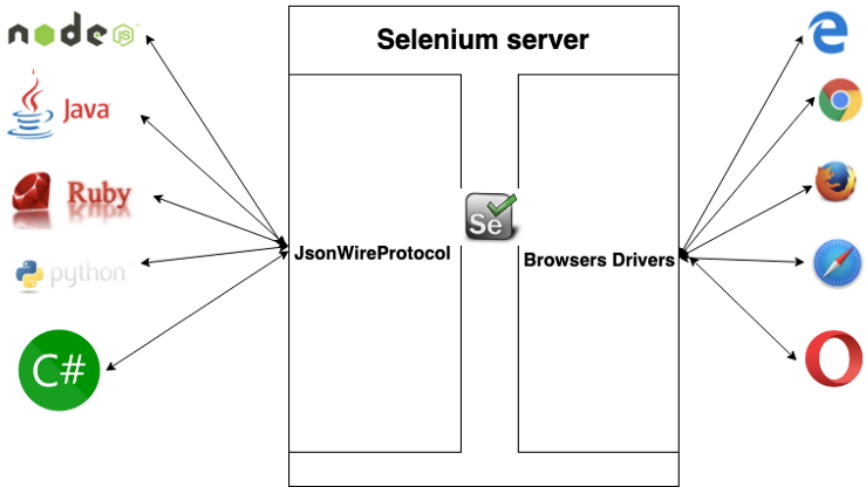

Рисунок 8 – Схема обмена данными в Selenium WebDriver

Непосредственно для написания сценариев тест-кейсов нужно

подключить предпочитаемую библиотеку, фреймворк для тестирования, в том числе Assertion Library и т.п.

Достоинства:

- «гибкость в использовании и выборе языка, платформы, браузера;
- Selenium Webdriver — де-факто стандарт индустрии, построенный на основе утвержденного веб-стандарта W3C WebDriver;
- большая поддержка сообщества;
- поддержка параллельного запуска тестов (Selenium Grid);
- поддержка различных плагинов (Selenium IDE как один из самых популярных, используемый для записи мануальных тестов, для создания автоматических);
- поддержка мобильных устройств» [24].

Недостатки:

- нетривиальная установка;
- нет встроенного хорошего инструмента для построения развернутого отчета об ошибках;
- нет встроенной возможности для сравнения изображений.

«Фреймворк Puppeteer – это open-source библиотека, которая предоставляет высокоуровневый API для запуска, контроля и управления браузера – Chromium.

В отличии от Selenium Webdriver, коммуникация происходит непосредственно с браузером, хотя и через тот же Chrome DevTools Protocol, который использует ChromeDriver Selenium. Особенность заключается в том, что Puppereer развивается и обновляется намного динамичнее, чем это происходит с ChromeDriver Selenium» [36].

Принцип работы фреймворка Puppeteer показан на рисунке 9.

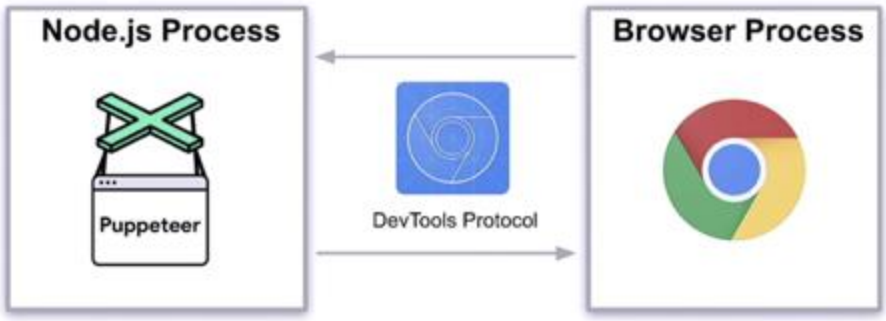

Рисунок 9 – Принцип работы фреймворка Puppeteer

Достоинства:

- богатый API для работы с сетевыми запросами и их перехватами;
- наличие возможности существенного ускорения тестов с помощью Puppeteer Browser Context;
- наличие API для тестирования приложения в режиме офлайн и др.

Недостатки:

- по факту поддержка работы только с одним браузером — Chomium;
- нет встроенного механизма параллелизации тестов;
- необходимость подключения дополнительных библиотек для полноценного, всеобъемлющего тестирования.

«Фреймворк Cypress – это open-source фреймфорк для E2E-тестирования. Это также, как и Puppeteer, относительно молодой инструмент, однако он вносит новые концепции и решения в способы осуществления автоматизации и тестирования.

Ключевой особенностью Cypress является то, что он исполняется внутри самого браузера. Это означает, что Cypress всегда отслеживает моменты вызова всякого рода событий в браузере и никогда не упустит любые манипуляции с элементами страницы, что намного уменьшает вероятность появления floating-тестов.

Достоинства:

- встроенный набор инструментов для тестирования (Mocha, Chai, Sinon);

- встроенный механизм автоматического ожидания;
- функция «Time machine», которая позволяет откатываться на определенные шаги в последовательности выполнения теста;
- исчерпывающая документация с большим набором примеров и др.» [15].

Недостатки:

- нет кросс-браузерной поддержки (только Chomium и Chrome);
- нет возможности создания дополнительных вкладок и окон;
- нет поддержки Native events браузера;
- не предназначен для тестирования производительности и др.

Для удобства сравнения характеристики фреймворков сведены в таблицу 5.

Таблица 5 – Сравнительный анализ фреймворков E2E-тестирования веб-приложений

Характеристика (1-5)	Selenium WebDriver	Puppeteer	Cypress
«Простота настройки	4	4	4
Простота освоения	4	4	4
Комьюнити	5	4	3
Функциональность	5	3	3
Документация	5	4	5
Производительность	4	4	4
Итого	27	23	23» [36]

Как показал анализ, наилучшими характеристиками для E2E тестирования веб-приложений обладает фреймворк Selenium.

С учетом вышеизложенного составлена таблица рекомендуемых фреймворков по уровням автоматизации тестирования веб-приложений (таблица 6).

Таблица 6 – Рекомендуемые фреймворки по уровням автоматизации тестирования веб-приложений

Уровень автоматизации тестирования	Фреймворк
«Модульное тестирование	JUnit
Интеграционное тестирование	Cucumber
E2E тестирование	Selenium» [36]

Выбранные фреймворки рекомендованы для использования при разработке моделей ИСАТ ПО.

Выводы по главе 2

В результате проделанной работы были сделаны следующие выводы:

- согласно концепции гибкого проектирования (Agile), автоматизированное тестирование должно быть не изолированной задачей, а непрерывным процессом, неотъемлемо вписанным в жизненный цикл ПО;
- с помощью сравнительного анализа по уровням тестирования выбраны фреймворки, которые в сообществе разработчиков считаются наилучшими.

Выбранные фреймворки рекомендованы для использования при разработке моделей ИСАТ ПО.

Глава 3 Разработка моделей и алгоритмов интегрированной среды автоматизированного тестирования программного обеспечения

3.1 Логическое моделирование интегрированной среды автоматизированного тестирования программного обеспечения

Рассмотрим процесс логического моделирования ИСАТ для веб-приложений.

«Для логического моделирования ИСАТ ПО используем методологию RUP (Rational Unified Process).

Методология RUP - это процесс разработки программного обеспечения для объектно-ориентированных моделей. Она также известна как унифицированная модель процесса» [34].

На начальной фазе логического проектирования определяются основные требования, ограничения и ключевая функциональность продукта. Создается базовая версия модели прецедентов [35].

Цель – определение основной функциональности ИСАТ ПО.

Ключевые участники – системный аналитик и тестировщик.

Входная информация: список функциональных требований к ИСАТ ПО, список рекомендуемых фреймворков по уровням автоматизации тестирования веб-приложений.

Результат – функциональная модель ИСАТ ПО.

Для определения основной функциональности ИСАТ ПО построим диаграмму вариантов использования UML.

«В рамках RUP модель бизнес-прецедентов предлагает синтаксис и семантику для представления бизнес-процессов организации на стратегических и тактических уровнях. уровни. Диаграмма вариантов использования поддерживается RUP и многими инструментами автоматизированной разработки программного обеспечения» [11].

Диаграммы вариантов использования обладают рядом преимуществ при

разработке программного обеспечения и проектировании систем:

- сбор функциональных требований. Диаграммы вариантов использования помогают фиксировать функциональные требования системы. Визуально представляя взаимодействие между участниками (пользователями или внешними системами) и системой, они дают четкое представление о том, как система должна вести себя;
- прослеживаемость. Варианты использования легко отслеживаемы. Каждый вариант использования соответствует определенной функциональности или взаимодействию с пользователем. Такая прослеживаемость гарантирует соблюдение требований во время разработки и тестирования;
- основа для оценки и планирования. Варианты использования служат основой для оценки, планирования и проверки затрат. Они помогают менеджерам проектов распределять ресурсы и планировать циклы разработки;
- эволюционное развитие. Варианты использования могут меняться на протяжении всего процесса разработки. Они начинаются как метод фиксации требований, затем служат руководством для программистов во время разработки, служат тестовыми примерами и, наконец, становятся частью пользовательской документации;
- альтернативные пути. Варианты использования позволяют фиксировать альтернативные пути или исключительное поведение. Эти пути повышают надежность системы за счет учета различных сценариев;
- удобное общение. Варианты использования легко понятны бизнес-пользователям. Они служат мостом между разработчиками программного обеспечения и конечными пользователями, способствуя эффективному общению;
- профессиональное моделирование. Вы можете создавать профессиональные модели вариантов использования с помощью

инструментов моделирования UML.

«На этапе управления требованиями RUP необходимо, чтобы все прецеденты и участники были определены, и было разработано большинство описаний прецедентов» [16].

Акторами процесса тестирования являются Тестировщик и фреймворки: JUnit, Selenium и Cucumber.

Варианты использования (прецеденты) представлены в таблицах 7-11.

Таблица 7 – Описание прецедента «Авторизация»

«Элемент диаграммы	Описание
Прецедент	Авторизация
ID	1
Краткое описание	Авторизация пользователя
Главный актор	Тестировщик
Второстепенный актор	Нет
Предусловие	Нет
Основной поток	Тестировщик выполняет авторизацию для доступа в ИСАТ
Постусловие	Нет
Альтернативные потоки	Нет» [10]

Таблица 8 – Описание прецедента «Модульное тестирование веб-приложения»

«Элемент диаграммы	Описание
Прецедент	Модульное тестирование веб-приложения
ID	2
Краткое описание	Модульное тестирование веб-приложения
Главный актор	Тестировщик
Второстепенный актор	фреймворк JUnit
Предусловие	Нет
Основной поток	Тестировщик выполняет модульное тестирование веб-приложения в ИСАТ
Постусловие	Нет
Альтернативные потоки	Нет» [10]

Таблица 9 – Описание прецедента «Интеграционное тестирование веб-приложения»

«Элемент диаграммы	Описание
Прецедент	Интеграционное тестирование веб-приложения
ID	3
Краткое описание	Модульное тестирование веб-приложения
Главный актор	Тестировщик
Второстепенный актор	фреймворк Cucumber
Предусловие	Нет
Основной поток	Тестировщик выполняет интеграционное тестирование веб-приложения в ИСАТ
Постусловие	Нет
Альтернативные потоки	Нет» [10]

Таблица 10 – Описание прецедента «E2E тестирование веб-приложения»

«Элемент диаграммы	Описание
Прецедент	E2E тестирование ПО
ID	4
Краткое описание	E2E тестирование ПО
Главный актор	Тестировщик
Второстепенный актор	фреймворк Selenium
Предусловие	Нет
Основной поток	Тестировщик выполняет E2E тестирование веб-приложения в ИСАТ
Постусловие	Нет
Альтернативные потоки	Нет» [10]

Таблица 11 – Описание прецедента «Формирование отчета»

«Элемент диаграммы	Описание
Прецедент	Формирование отчета
ID	5
Краткое описание	Формирование отчета тестирования ПО
Главный актор	Тестировщик
Второстепенный актор	Нет
Предусловие	Нет» [10]

Продолжение таблицы 11

«Элемент диаграммы	Описание
Основной поток	Тестировщик формирует отчет результатов тестирования ПО
Постусловие	Нет
Альтернативные потоки	Нет» [10]

На рисунке 10 представлена диаграмма вариантов использования ИСАТ веб-приложений.

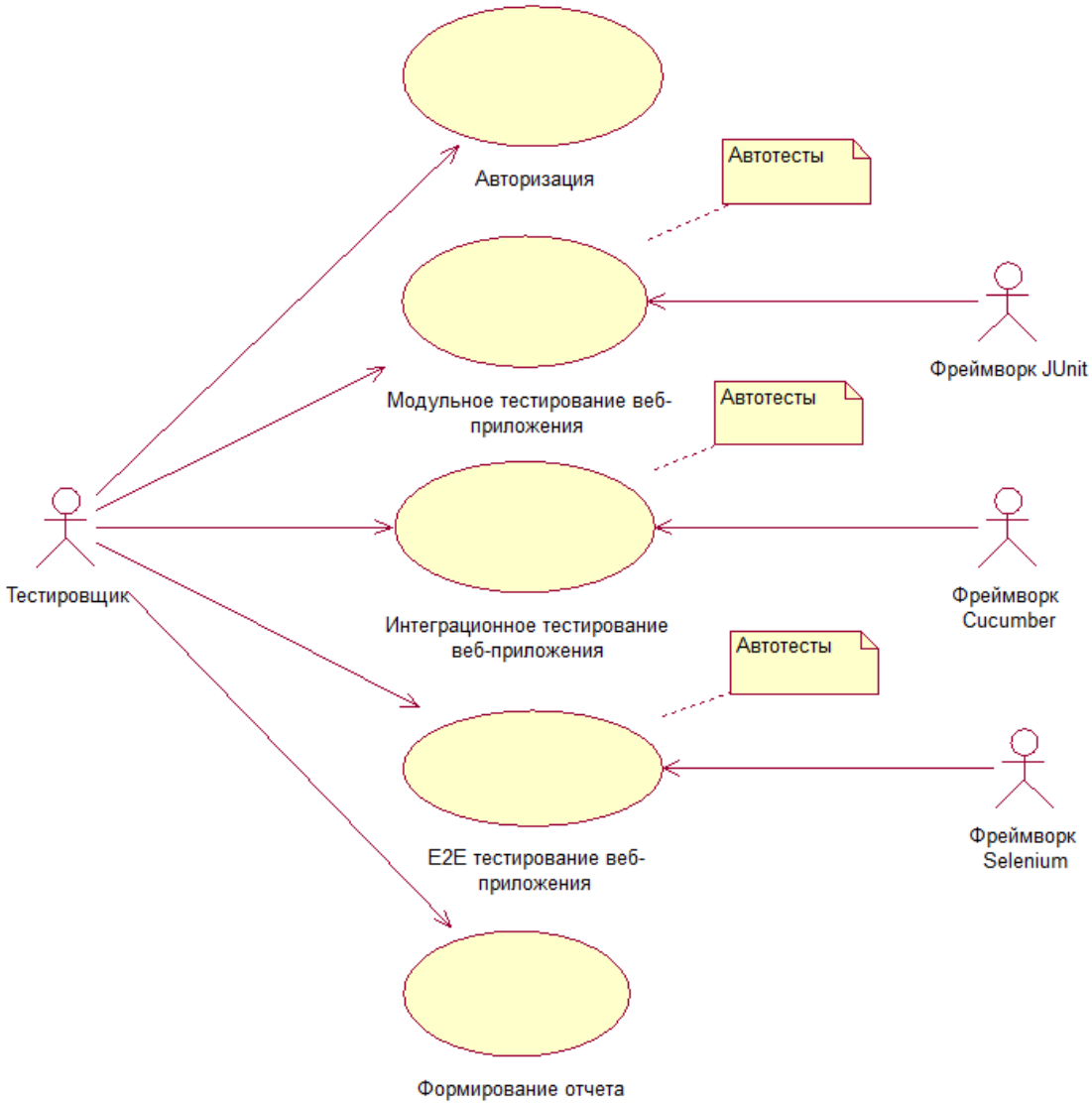

Рисунок 10 – Диаграмма вариантов использования ИСАТ веб-приложений

Диаграмма построена с точки зрения тестировщика.

Разработанная диаграмма вариантов использования отражает функциональный аспект и является функциональной моделью ИСАТ веб-приложений.

Для анализа предметной области автоматизации разработаем ее модель.

Модель предметной области (модель бизнес-объектов) - мощный механизм для описания важных терминов бизнеса, обеспечивающий единое определение терминов и их взаимосвязей, доступное для всех сотрудников проекта, от бизнес-менеджеров высокого уровня до инженеров низкого уровня.

Одним из преимуществ использования модели предметной области является то, что термины моделируются как элементы, что позволяет связывать их с другими элементами внутри самой модели предметной области или с элементами в других частях моделей [23].

Для разработки модели предметной области используем диаграмму классов UML.

Диаграмма классов – ценный инструмент при разработке программного обеспечения и проектировании систем.

Рассмотрим их преимущества:

- иллюстрация модели данных. Диаграммы классов позволяют иллюстрировать модели данных для информационных систем, независимо от их сложности. Они обеспечивают визуальное представление того, как классы, атрибуты и отношения сочетаются друг с другом;
- обзор системы. Создавая диаграммы классов, вы можете лучше понять общий обзор структуры приложения. На этих диаграммах показаны основные компоненты и их взаимодействие;
- коммуникация и распространение. Диаграммы классов помогают визуально выразить конкретные потребности системы. Вы можете распространить эту информацию по всей организации, гарантируя,

что все понимают структуру системы.

– подробное руководство по программированию. При создании диаграммы классов позволяют сгенерировать конкретный код, необходимый для реализации. Они направляют программистов во время разработки и следят за тем, чтобы система соответствовала намеченной структуре;
– описания, независимые от реализации. Диаграммы классов предоставляют независимое от реализации описание типов, используемых в системе. Эти типы впоследствии можно будет передавать между различными компонентами системы.

На рисунке 11 изображена диаграмма классов ИСАТ веб-приложений.

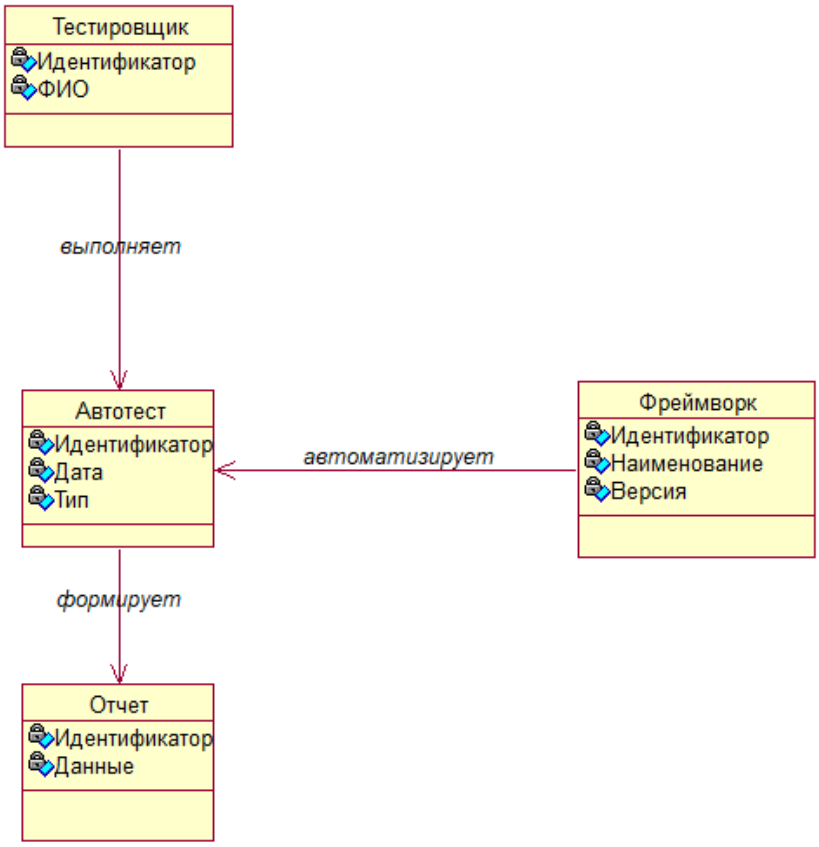

Рисунок 11 – Диаграмма вариантов использования ИСАТ веб-приложений

В таблице 12 представлена спецификация классов ИСАТ.

Таблица 12 – Спецификация классов ИС тестирования веб-приложений

«Класс	Описание
Тестировщик	Класс объектов, представляющих на логическом уровне тестировщиков ПО
Фреймворк	Класс объектов, представляющих на логическом уровне фреймворки для автоматизации тестирования
Автотест	Класс объектов, представляющих на логическом уровне автоматизируемые тесты
Отчет	Класс объектов, представляющих на логическом уровне отчеты результатов тестирования» [10]

«Для разработки модели программной архитектуры ИСАТ веб-приложений используем диаграмму компонентов UML» [37].

Диаграммы компонентов очень важны с точки зрения реализации. Эти диаграммы широко используется для иллюстрации программной архитектуры ИСАТ [17].

Таким образом, команда разработчиков приложения должна хорошо разбираться в деталях компонентов.

На рисунке 12 изображена программная архитектура ИСАТ веб-приложений, построенная в виде диаграммы компонентов UML.

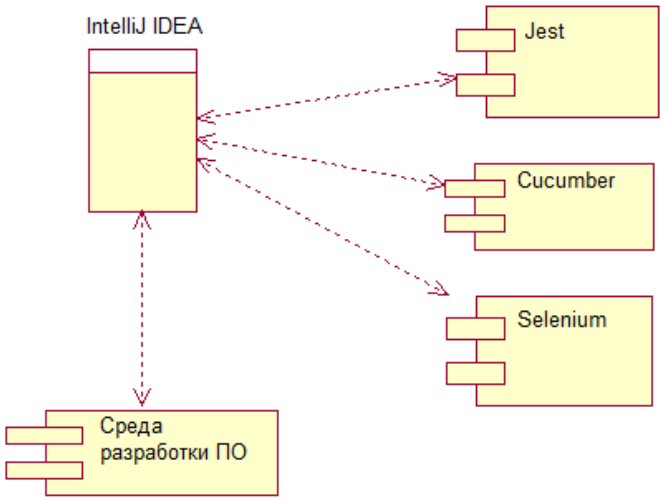

Рисунок 12 – Программная архитектура ИСАТ веб-приложений

Представленная программная архитектура отражает общие зависимости между программными компонентами ИСАТ веб-приложений.

3.2 Информационное моделирование интегрированной среды автоматизированного тестирования программного обеспечения

Информационная модель – это организационная структура, которую вы используете для категоризации своих информационных ресурсов. Эта структура помогает авторам и пользователям находить то, что им нужно, даже если их потребности существенно различаются и носят личный характер.

Создание информационной модели требует анализа, тщательного планирования и большого количества отзывов от вашего сообщества пользователей. Анализ переносит вас в мир тех, кто нуждается и использует информационные ресурсы каждый день.

Планирование означает общение с широким кругом заинтересованных сторон, включая отдельных лиц и группы, у которых есть информационные потребности и которые выиграют от сотрудничества в разработке информационных ресурсов.

Для получения обратной связи вам необходимо протестировать свою информационную модель с членами вашего сообщества пользователей, чтобы убедиться, что вы не упустили некоторые важные перспективы.

Для построения информационной модели ИСАТ ПО использован онлайн-сервис Visual Paradigm [39].

Информационная модель функции тестирования ПО изображена на рисунке 13.

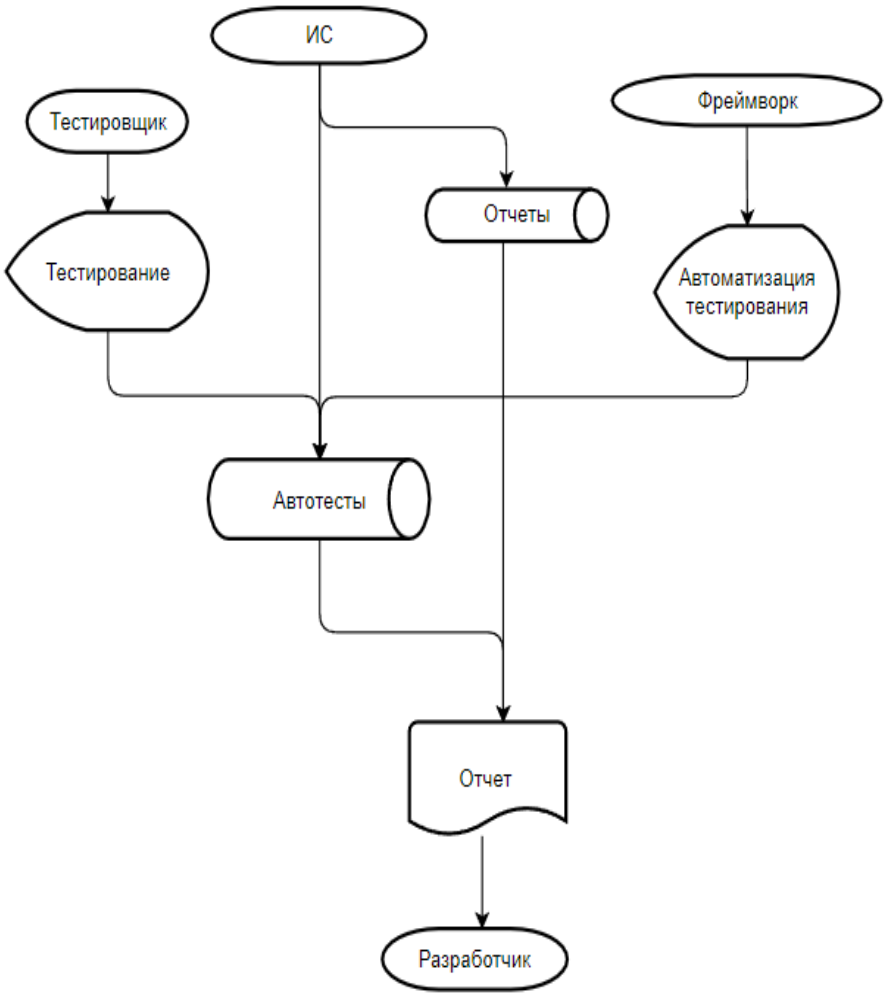

Рисунок 13 – Информационная модель функции тестирования ПО

Представленная информационная модель строится на основе модели бизнес-процесса тестирования ПО в ИТ-компании.

Для разработки базы данных (БД) ИСАТ ПО используется СУБД MySQL.

«Для разработки модели данных ИСАТ ПО используем CASE-средство MySQL Workbench, которое поддерживает стандарт IDEF1X.

Workbench предоставляет возможность моделирования данных, разработку SQL и комплексные инструменты администрирования для конфигурации сервера, администрирования пользователей, резервного копирования и т. д.

Workbench позволяет создавать физическую модель БД для СУБД

MySQL без предварительного логического моделирования, что позволяет существенно повысить производительность процесса» [33].

В процессе разработки ER-диаграммы ИС из ее диаграммы классов выделены следующие сущности:

- Пользователь;
- Автотесты;
- Фреймворк;
- Отчет.

На рисунке 14 показана модель данных ИСАТ веб-приложений.

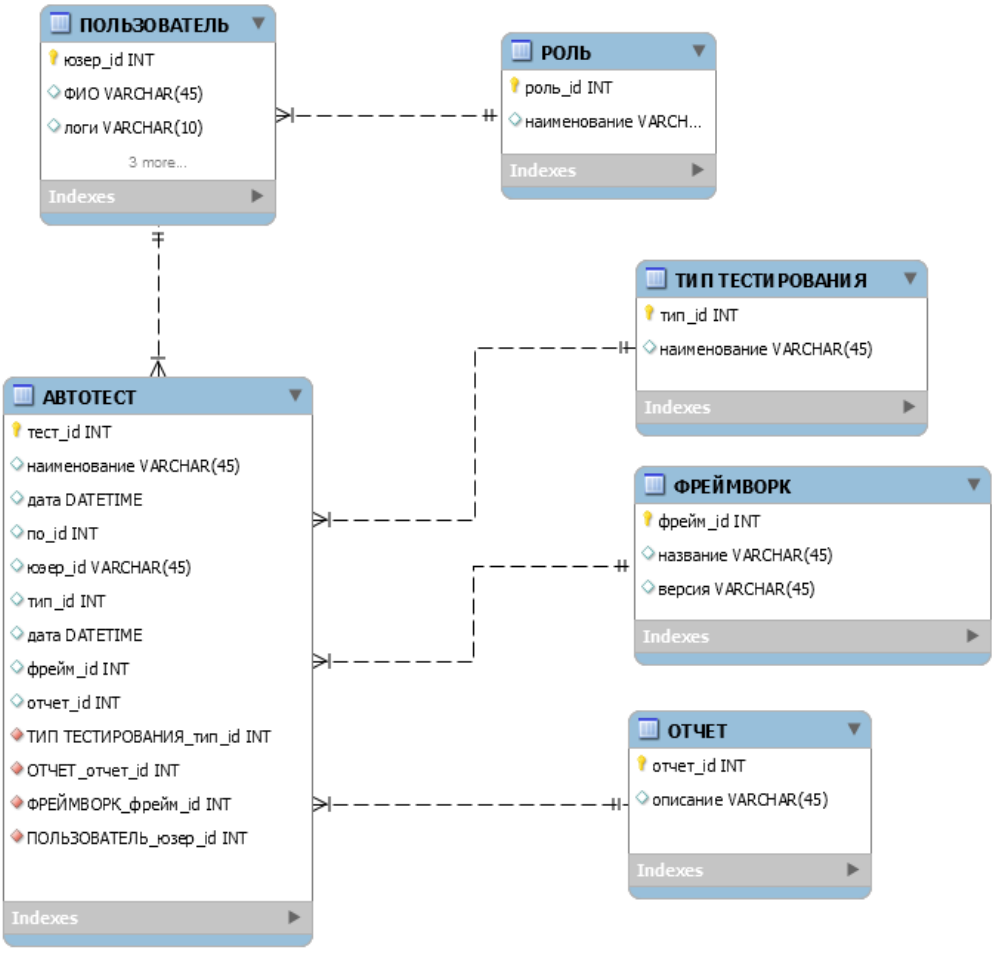

Рисунок 14 – Модель данных ИСАТ веб-приложений

Для приведения данных к третьей форме нормализации введены сущности Тип тестирования и Роль пользователя.

- Между сущностями установлены следующие связи:
- Пользователь-Автотест – «один ко многим»;
- Тип тестирования-Автотест – «один ко многим»;
- Тестируемое ПО-Автотест – «один ко многим»;
- Фреймворк-Автотест – «один ко многим»;
- Отчет-Автотест – «один ко многим»;
- Роль-Пользователь – «один ко многим».

Так как ИСАТ веб-приложений относится к OLTP-системам, все связи между сущностями неидентифицирующие.

3.3 Физическое проектирование интегрированной среды автоматизированного тестирования программного обеспечения

Для проектирования ИСАТ ПО используем подход, основанный на применении технологической платформы, в качестве которой используем Integrated Development Environment (IDE) или интегрированную среду разработки.

«IDE – это многофункциональная программа, которую можно использовать для различных аспектов разработки ПО.

Интегрированная среда разработки позволяет программистам объединять различные задачи написания компьютерной программы.

Интегрированные среды разработки повышают продуктивность программиста за счет объединения общих действий по написанию программного обеспечения в одном приложении: редактирование исходного кода, создание исполняемых файлов и отладка.

В состав типовой интегрированной среды разработки входят:

- текстовый редактор;
- транслятор – компилятор или интерпретатор;
- средства автоматизации сборки;
- отладчик» [8].

Рассмотрим и сравним характеристики интегрированных сред разработки, используемых для разработки веб-приложений: IntelliJ IDEA, NetBeans, Eclipse.

«IntelliJ IDEA – это ведущая, а по мнению многих отраслевых экспертов и Java-разработчиков, просто лучшая на сегодняшний день среда быстрой разработки приложений на языке Java.

IntelliJ IDEAпредставляет собой высокотехнологичный комплекс тесно интегрированных инструментов программирования, включая интеллектуальный редактор исходных текстов с развитыми средствами автоматизации, мощные инструменты рефакторинга кода, встроенную поддержку технологий J2EE, механизмы интеграции со средой тестирования Ant/JUnit и системами управления версиями, уникальный инструмент оптимизации и проверки кода Code Inspection, а также инновационный визуальный конструктор графических интерфейсов (рисунок 15)» [30].

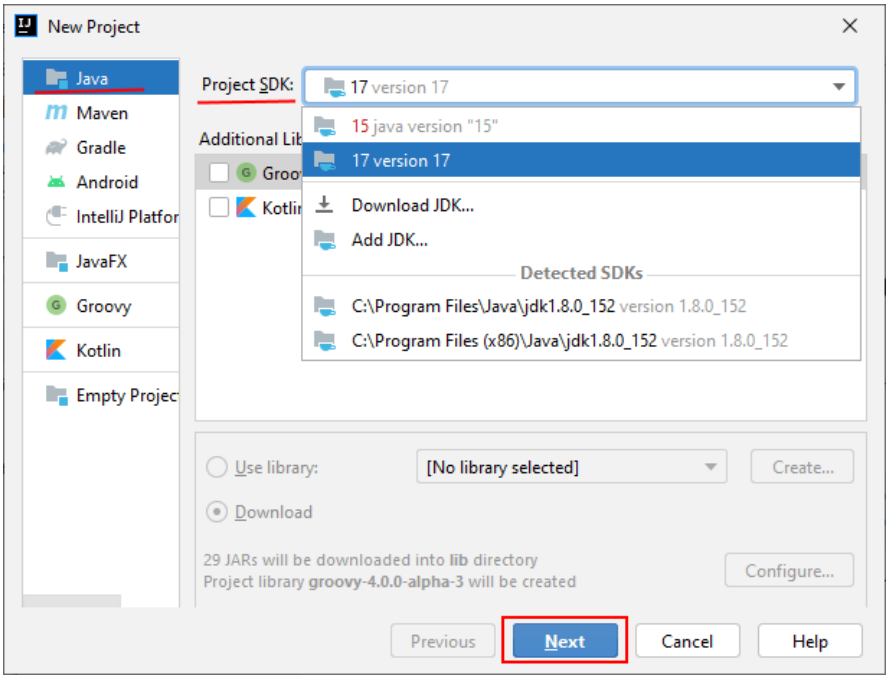

Рисунок 15 – Среда IntelliJ IDEA

Начиная с версии 9.0, среда доступна в двух редакциях: Community

Edition (бесплатная) и Ultimate Edition.

Бесплатная версия поддерживает Java, Kotlin, Groovy и Scala; Android; Maven, Gradle и SBT; работает с системами контроля версий Git, SVN, Mercurial и CVS.

NetBeans – это бесплатная интегрированная среда разработки с открытым исходным кодом для разработчиков ПО (рисунок 16).

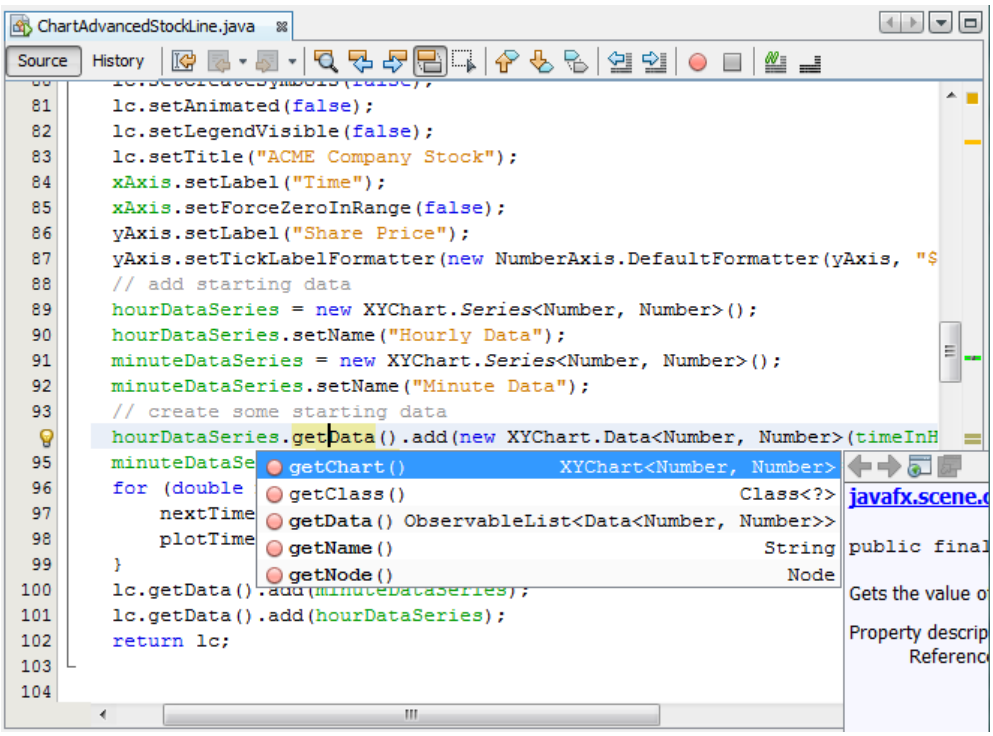

Рисунок 16 – Среда IDE NetBeans

«Среда NetBeans предоставляет все средства, необходимые для создания профессиональных десктоп-приложений, корпоративных, мобильных и веб-приложений на платформе Java, а также C++, PHP и других языках.

Среда NetBeans позволяет разрабатывать Java десктоп-приложения с профессиональными графическими интерфейсами пользователя, а также создавать веб- и корпоративные приложения в соответствии с современными стандартами» [20].

«Интегрированная среда разработки Eclipse является бесплатной

программной платформой с открытым исходным кодом, контролируется организацией Eclipse Foundation.

Среда написана на языке программирования Java.

Основной целью её создания является повышение продуктивности процесса разработки программного обеспечения» [25].

Фрагмент среды представлен на рисунке 17.

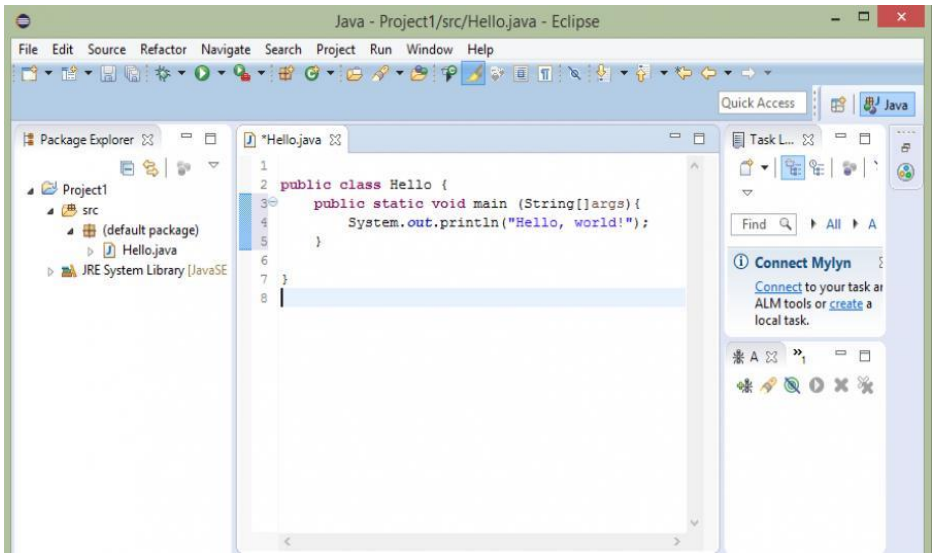

Рисунок 17 – Среда IDE Eclipse

«Основным компонентом является исполняемая среда – Eclipse Runtime, в которой выполняются коды расширений и модулей.

Она обеспечивает всю базовую функциональность – управление расширениями и обновлениями, взаимодействие с операционной системой, обеспечение работы системы помощи.

Следующим ключевым компонентом является собственно интегрированная среда разработки – она отвечает за управление элементами программы, управление проектами, отладку и сборку проектов, поиск по файлам и командную программу» [25].

Для выбора интегрированной среды разработки используем таблицу 12, составленную на основе анализа блогов по данной тематике.

Таблица 13 – Сравнительный анализ интегрированных сред разработки

Характеристика/балл	IntelliJ IDEA	NetBeans	Eclipse
Юзабилити	2	2	2
Простота интеграции с фреймворками для автоматизации тестирования	3	2	2
Продуктивность	3	2	2
Итого	8	6	6

С учетом результатов анализа выбираем IntelliJ IDEA в качестве среды для разработки ИСАТ веб-приложений.

С помощью диаграммы пакетов создается представление модульной конфигурации ПО.

На рисунке 18 изображена структурная схема ИСАТ веб-приложений в виде диаграмма пакетов.

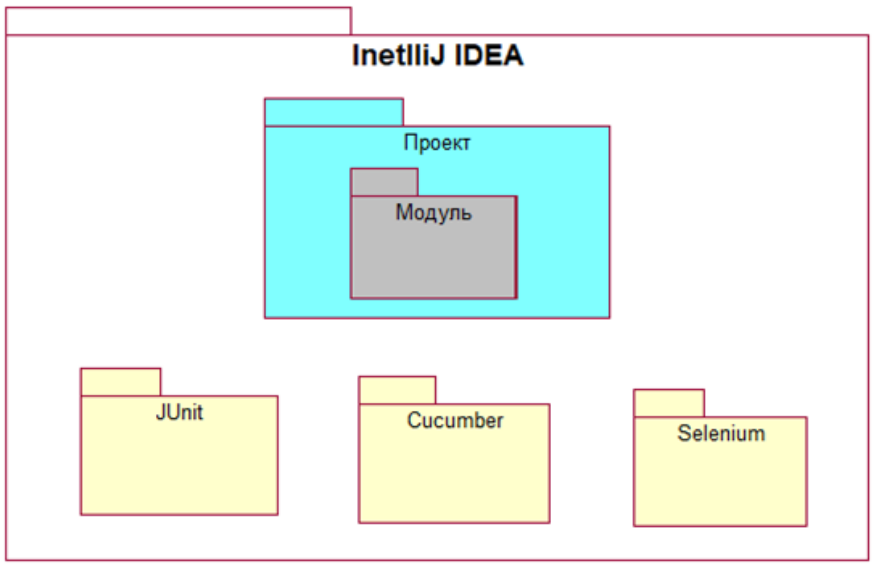

Рисунок 18 – Структурная схема ИСАТ веб-приложений

В таблице 14 дано описание функций модулей ИСАТ.

Таблица 14 – Описание функций модулей ИСАТ

Наименование модуля	Функции модуля
«IntelliJ IDEA	Содержит инструментарий для управления процессом тестирования веб-приложений
Фреймворк JUnit	Автоматизированная поддержка модульного тестирования веб-приложений
Фреймворк Cucumber	Автоматизированная поддержка интеграционного тестирования веб-приложений
Фреймворк Selenium	Автоматизированная поддержка E2E тестирования веб-приложений» [8]

Проект в IntelliJ состоит из модулей (например, java-модулей). Модули содержат программный код.

Таким образом, в концепции IntelliJ модули – это части проекта, которые можно компилировать, запускать, тестировать и отлаживать независимо друг от друга. Модули - это способ уменьшить сложность больших проектов при сохранении общей (проектной) конфигурации. Модули можно использовать повторно: при необходимости модуль можно включить более чем в один проект.

Проект может содержать несколько модулей. В настоящее время основные структуры крупномасштабных проектов представляют собой в основном многомодульные структуры.

3.4 Разработка алгоритмов тестирования веб-приложений

Диаграмма деятельности изображает поток управления от начальной до конечной точки, показывая различные пути принятия решений, которые существуют во время выполнения алгоритма.

На рисунке 19 показан алгоритм тестирования веб-приложений, построенный в виде диаграммы деятельности UML.

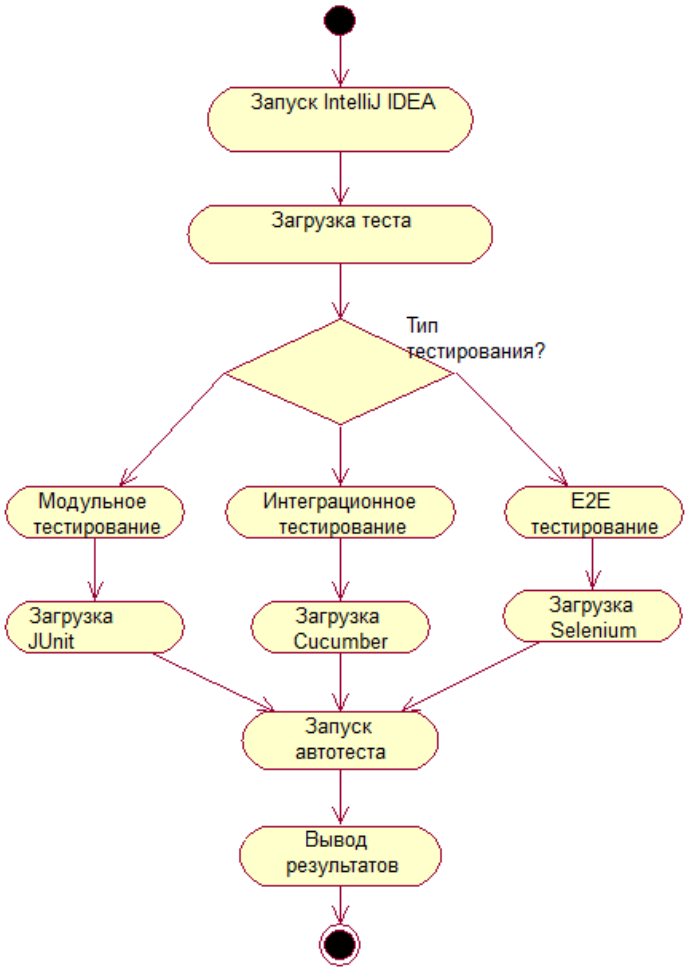

Рисунок 19 – Алгоритм тестирования веб-приложений по уровням тестирования

Существуют разные типы тестирования производительности веб-приложений, и один из них – нагрузочное тестирование. Нагрузочное тестирование проводится для определения производительности системы в реальных условиях.

Единственная причина проведения нагрузочного тестирования – проверить, какую нагрузку может выдержать система. Он подтверждает, может ли устройство или программное обеспечение справиться с нагрузкой в соответствии с требованиями пользователя.

Кроме того, нагрузочное тестирование используется для определения максимальной работоспособности любого приложения. С помощью нагрузочного тестирования вы можете определить, достаточна ли

существующая инфраструктура для запуска приложения. Наконец, тест определяет количество пользователей, которые могут одновременно работать над приложением.

Для нагрузочного тестирования сайта веб-приложения используется программа Apache JMeter [19].

Алгоритм нагрузочного тестирования JMeter показан на рисунке 20.

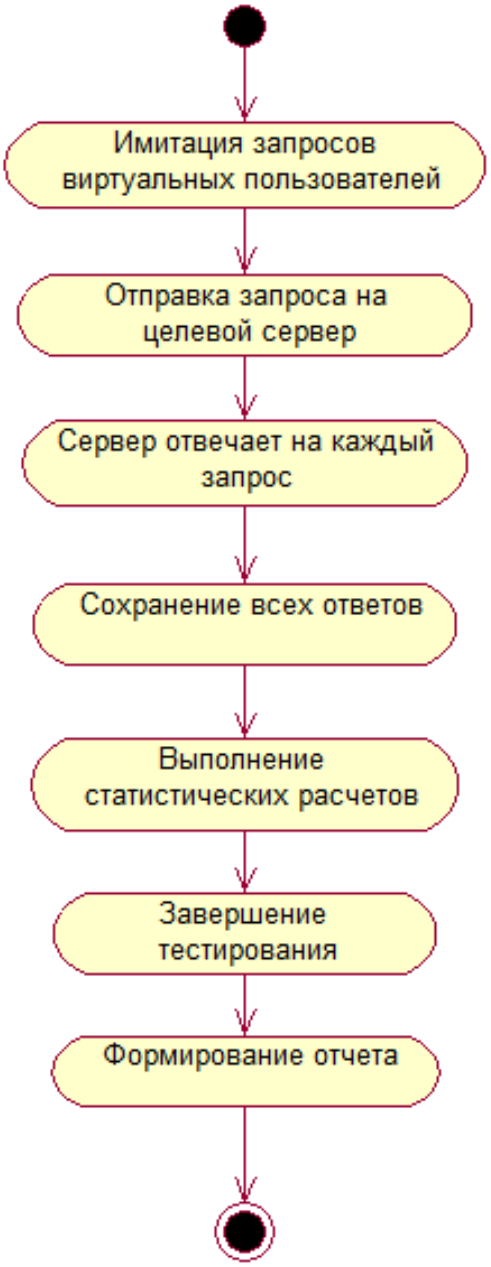

Рисунок 20 – Алгоритм нагрузочного тестирования веб-приложения с помощью JMeter

Представленные алгоритмы реализованы в прототипе ИСАТ ПО.

Выводы по главе 3

В результате проделанной работы были сделаны следующие выводы:

- на начальной фазе логического проектирования определяются основные требования, ограничения и ключевая функциональность продукта. Создается базовая версия модели прецедентов;
- для проектирования ИСАТ ПО применен подход, основанный на применении технологической платформы, в качестве которой используется IDE или интегрированная среда разработки;
- с учетом результатов анализа IntelliJ IDEA выбрана в качестве среды для разработки ИСАТ веб-приложений;
- диаграмма деятельности изображает поток управления от начальной до конечной точки, показывая различные пути принятия решений, которые существуют во время выполнения алгоритма;
- нагрузочное тестирование используется для определения максимальной работоспособности любого приложения. Для нагрузочного тестирования сайта веб-приложения используется программа Apache JMeter.

Представленные алгоритмы реализованы в прототипе ИСАТ ПО.

Глава 4 Апробация результатов исследования и оценка эффективности проектных решений.

4.1 Апробация результатов исследования

Прототип ИСАТ ПО разработан на базе интегрированной среды разработки IntelliJ IDEA.

Для представления иерархической структуры функций ИС служит диаграмма дерева функции, или иерархическая диаграмма.

На рисунке 21 изображено дерево функций ИСАТ ПО.

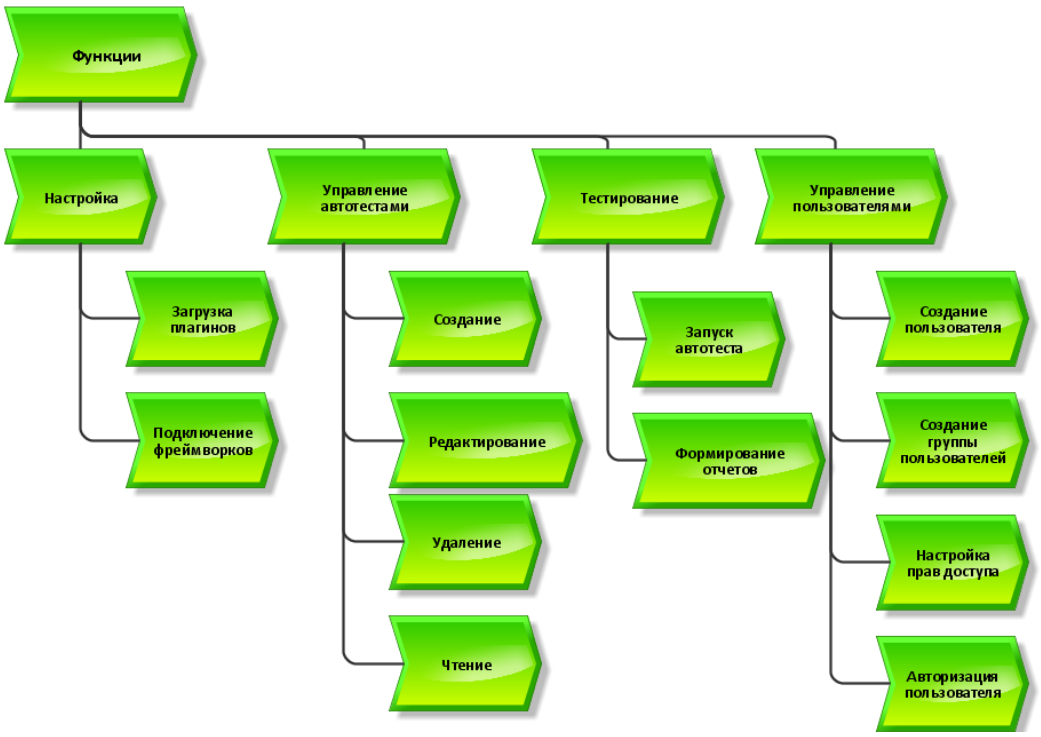

Рисунок 21 – Дерево функций ИСАТ ПО

Среда IntelliJ IDEA позволяет настраивать параметры на нескольких уровнях: на уровне модуля, на уровне проекта и глобально.

Глобальные настройки применяются ко всем проектам, которые открываются с определенной установкой или версией IntelliJ IDEA.

К таким настройкам относятся внешний вид среды (темы, цветовые

схемы, меню и панели инструментов), настройки уведомлений, набор установленных и включенных плагинов, настройки отладчика, завершение кода и т.д.

Чтобы настроить конфигурацию среды, выберите опцию «Файл| Настройки для Windows и Linux» (рисунок 22).

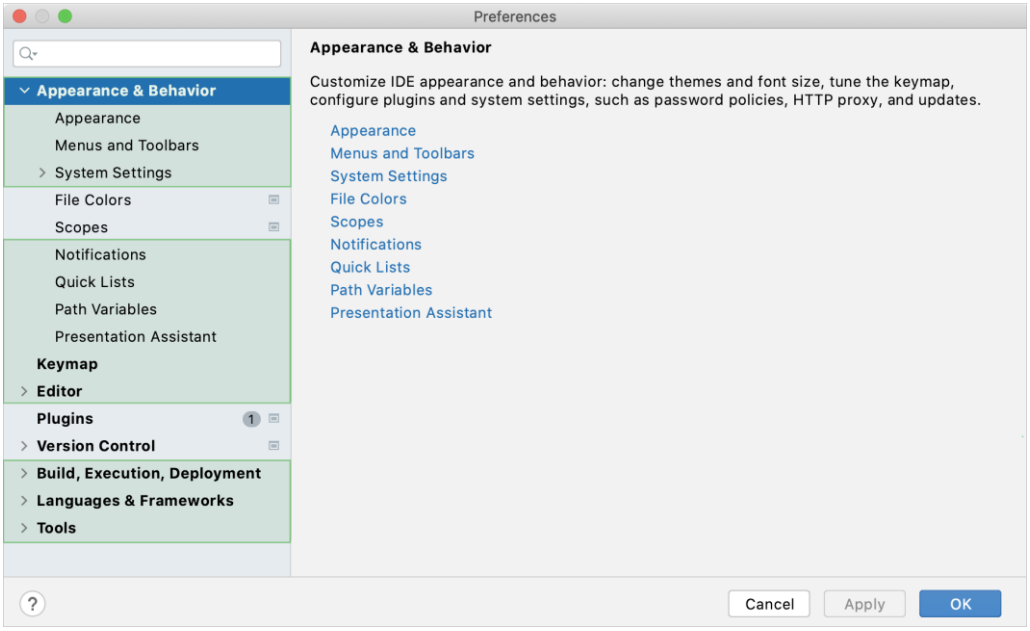

Рисунок 22 – Окно настройки конфигурации ИСАТ ПО

Настройки, которые не отмечены значком «Для текущего проекта» в диалоговом окне «Settings/Preferences», являются глобальными и применяются ко всем существующим проектам текущей версии IntelliJ.

Интеграция фреймворков в ИСАТ осуществляется через опцию «Плагины», последние версии которых можно скачать с сайтов разработчиков.

Процесс реализации прототипа ИСАТ веб-приложений представляет собой интеграцию среды разработки IntelliJ IDEA с фреймворками JUnit, Cucumber и Selenium.

Библиотека для JUnit поставляется с IntelliJ IDEA, но по умолчанию не включена в путь к классам проекта или модуля.

Следовательно, при создании тестового класса ссылки на класс TestCase

или текстовые аннотации не разрешаются.

«Чтобы добавить необходимую библиотеку в путь к классам, можно использовать общую процедуру добавления зависимости к модулю.

Соответствующая библиотека расположена в следующих каталоге:

JUnit библиотеки (junit.jar и junit-4.12.jar): <IntelliJ IDEA directory>\lib

IntelliJ IDEA может автоматически добавить необходимую библиотеку в путь к классам. Соответствующие функции доступны при создании теста для класса или при написании кода для теста из подменю «Testing-JUnit» (рисунок 23)» [30].

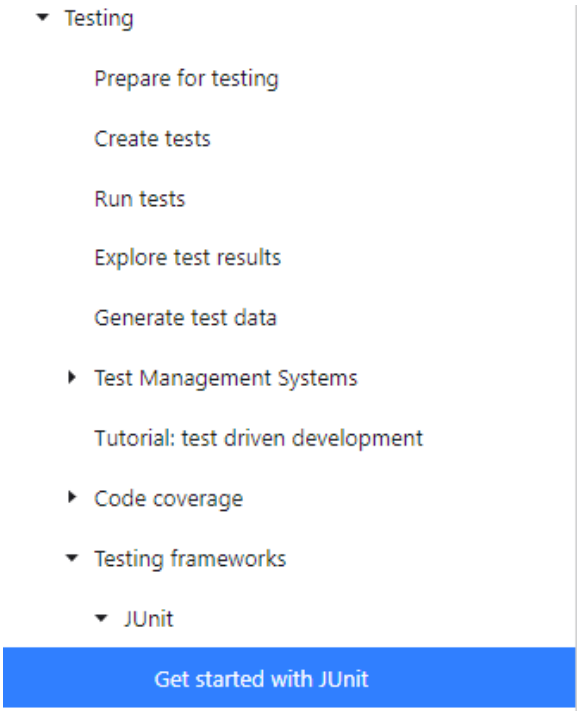

Рисунок 23 – Окно подменю JUnit-тестирования

Чтобы иметь возможность использовать фреймворк Cucumber в своем приложении, надо убедиться, что необходимые плагины включены, и добавить библиотеку Cucumber в свой проект.

В IntelliJ Ultimate необходимые плагины объединены и включены по умолчанию. Однако разработчики рекомендуют убедиться, что они включены.

В IntelliJ Community нужные плагины не связаны, поэтому необходимо

установить и включить их.

Для этого в диалоговом окне Settings/Preferences (Ctrl + Alt + S) выбираем Plugins.

Переходим на вкладку Installed и убеждаемся, что следующие плагины включены в указанном порядке:

- Gherkin;
- Cucumber for Java;
- Cucumber for Groovy (необязательно).

Если плагины не установлены, переходим на вкладку Marketplace, вводим их имена в поле поиска в указанном порядке и нажимаем на кнопку Install рядом с каждым из них.

Сохраняем изменения и закрываем диалог. Если будет предложено, перезапускаем среду IDE (рисунок 24).

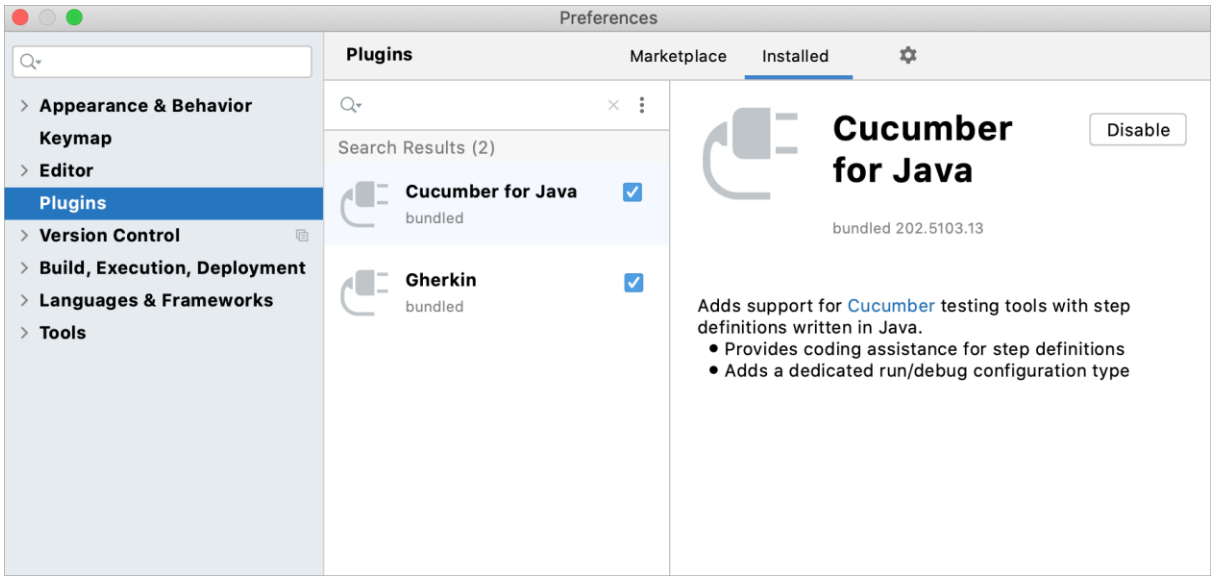

Рисунок 24 – Окно установки плагинов Cucumber

Далее необходимо подключить библиотеку Cucumber (рисунок 25).

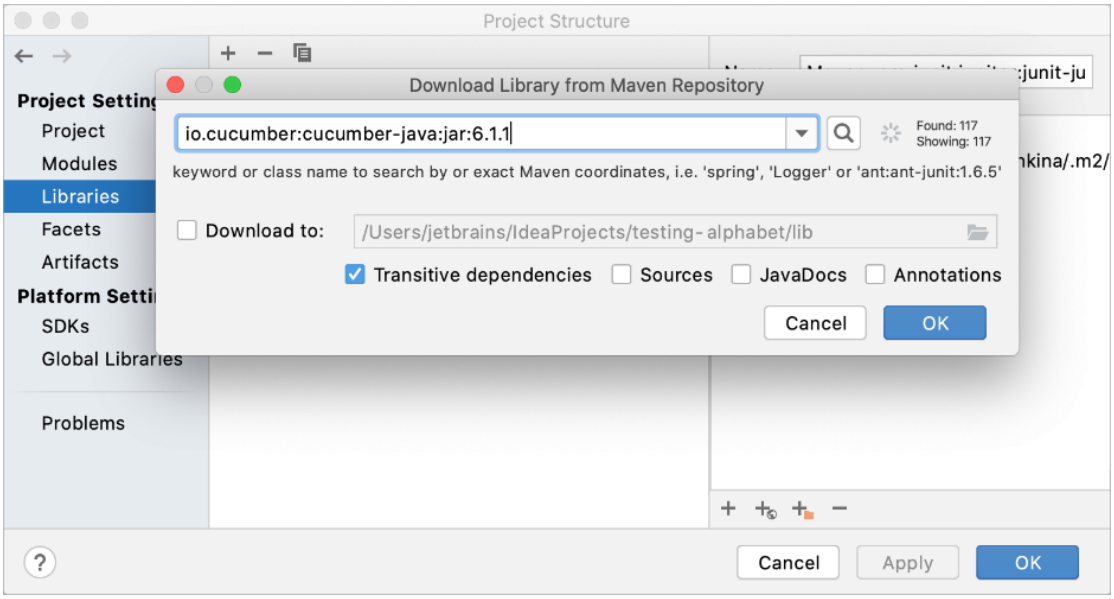

Рисунок 25 – Окно подключения библиотеки Cucumber

Для использования фреймворка Selenium необходимо установить его плагин.

Версия IntelliJ IDEA 2020.1 Ultimate представляет начальную поддержку Selenium с новым плагином Selenium UI Automation Testing.

Новый плагин поддерживает самые популярные платформы JVM для тестирования пользовательского интерфейса и библиотеки отчетов: Selenium, Selenide, Geb, Serenity BDD и Allure Framework.

Для установки этого плагина в диалоговом окне Settings/Preferences (Ctrl + Alt + S) выбираем Plugins. Переходим на вкладку Marketplace, вводим Selenium UI Testing и нажимаем на кнопку Install.

Сохраняем изменения и закрываем диалог. Если будет предложено, перезапускаем среду IDE (рисунок 26).

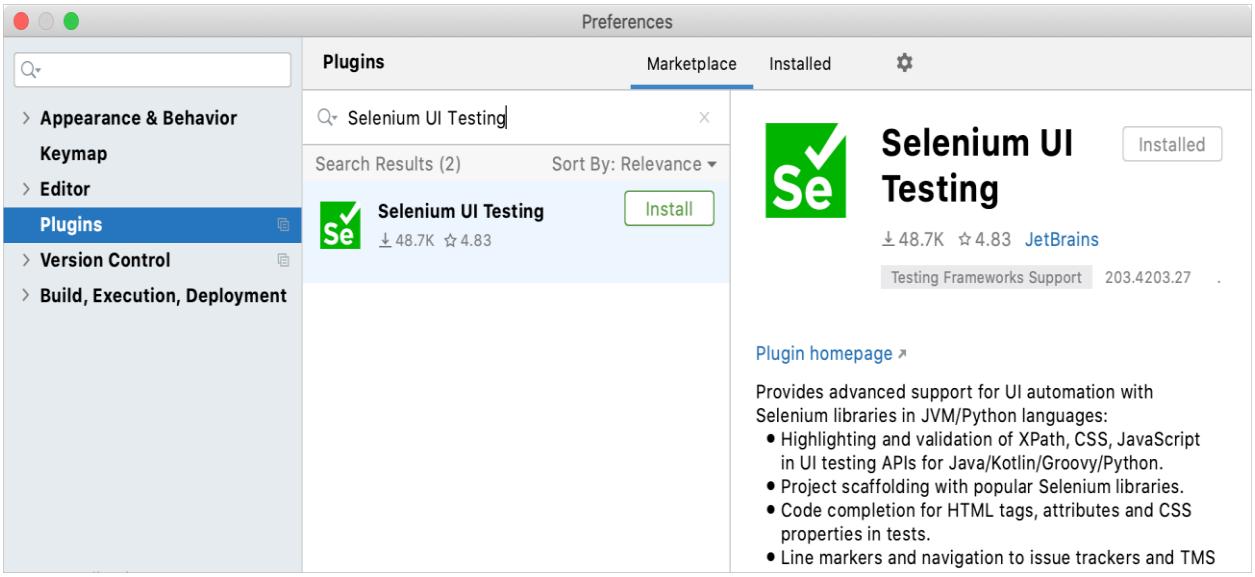

Рисунок 26 – Окно установки плагина Selenium UI Testing

Созданный прототип ИСАТ ПО представляет собой интегрированную САТ на платформе IntelliJ IDEA.

Для апробации результатов исследования проводилось автоматизированное тестирование веб-приложения в разработанной ИСАТ.

Рассмотрим реализацию функций типов тестирования веб-приложения.

Модульное тестирование веб-приложения осуществляется с помощью фреймворка JUnit.

Предварительно необходимо создать JUnit-проект (рисунок 27).

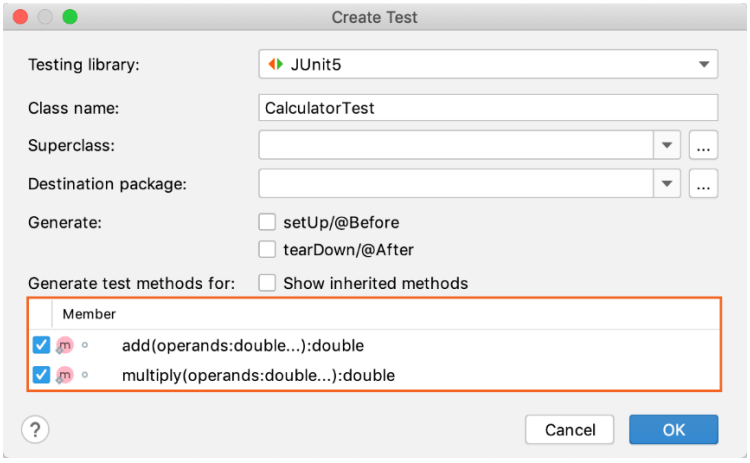

Рисунок 27 – Окно создания теста JUnit

После того, как настроен код для тестирования, можно запустить тесты и выяснить, правильно ли работают проверенные методы.

Чтобы выполнить отдельный тест, нажимаем на значок запуска конфигурации теста состояния запуска в поле и выбираем «Выполнить» (рисунок 28).

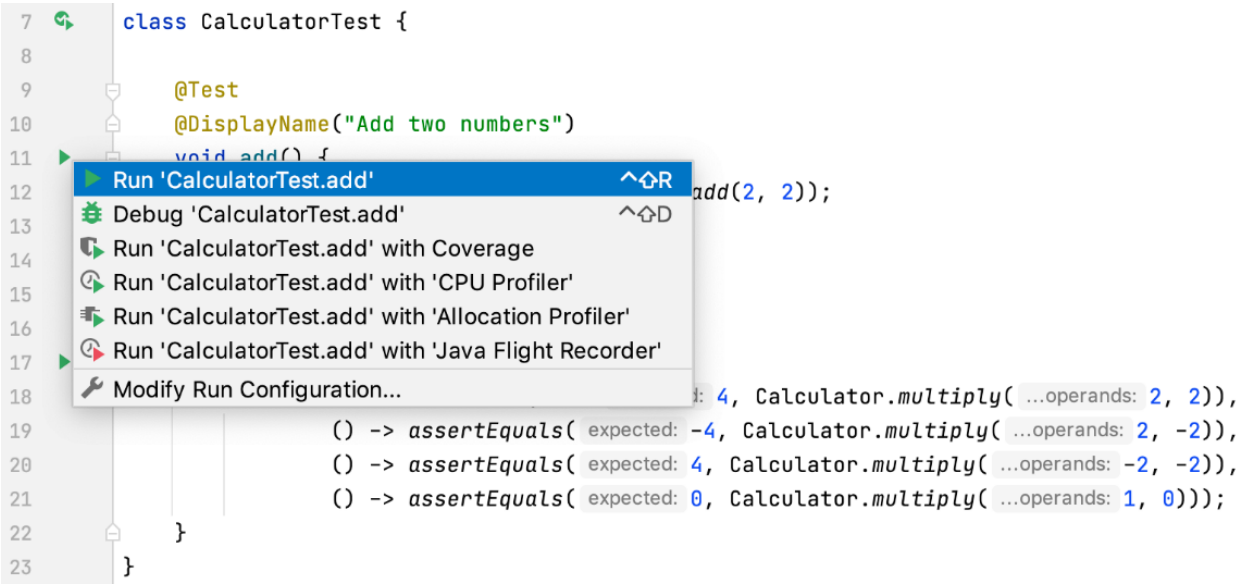

Рисунок 28 – Окно запуска JUnit-теста

Просмотреть результаты тестирования можно непосредственно в окне выполнения тестов (рисунок 29).

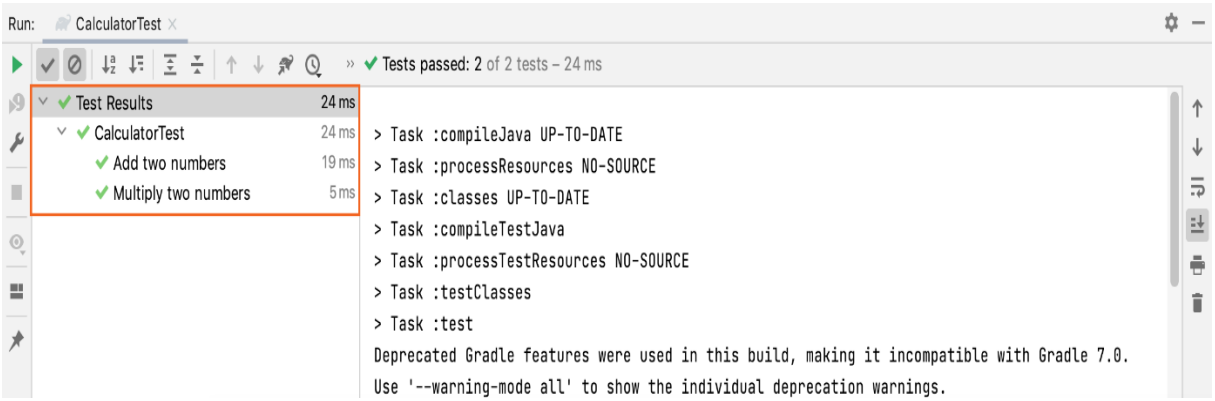

Рисунок 29 – Окно просмотра результатов выполнения JUnit-теста

Интеграционное тестирование веб-приложения осуществляется с помощью фреймворка Cucumber.

Предварительно необходимо создать структуру папок Cucumber-проекта, как показано на рисунке 30.

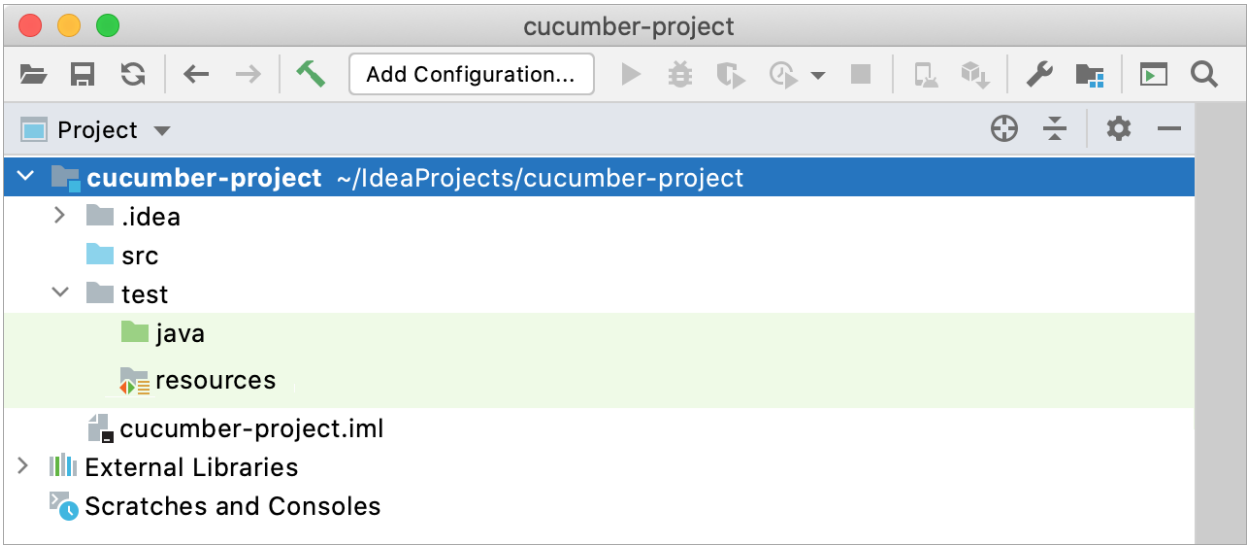

Рисунок 30 – Структура папок Cucumber-проекта

Самый быстрый способ запустить тесты Cucumber – использовать значки в поле рядом с необходимой функцией или сценарием. Значки меняются в зависимости от состояния теста.

На рисунке 31 показано окно выполнения тестового сценария. Для этого надо активизировать опцию «Запустить тест» в поле рядом со сценарием, который хотим запустить, и выбрать «Выполнить» «Сценарий: <имя>».

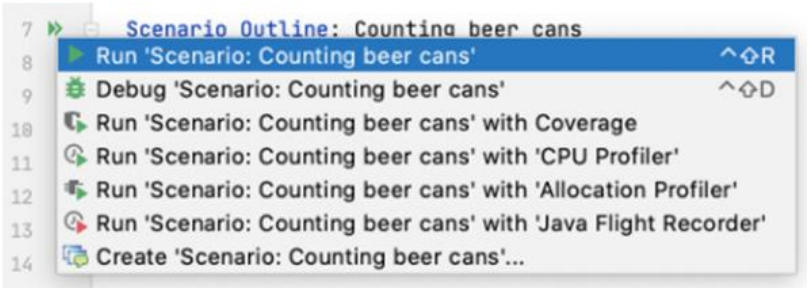

Рисунок 31 – Окно выполнения тестового сценария

На рисунке 32 показано окно выполнения функции.

Для этого надо активизировать опцию «Запустить тест» в поле рядом с функцией, которую вы хотите запустить, и выберите «Выполнить» «Функция: <имя>».

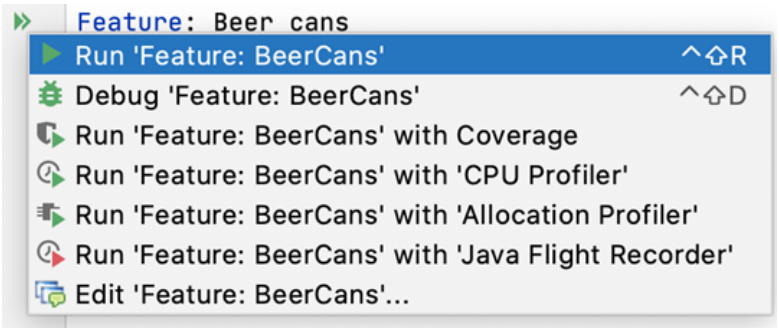

Рисунок 32 – Окно выполнения функции

При завершении выполнения тестов результаты отображаются на вкладке «Выполнение тестов» в окне инструмента «Выполнение» (рисунок 33).

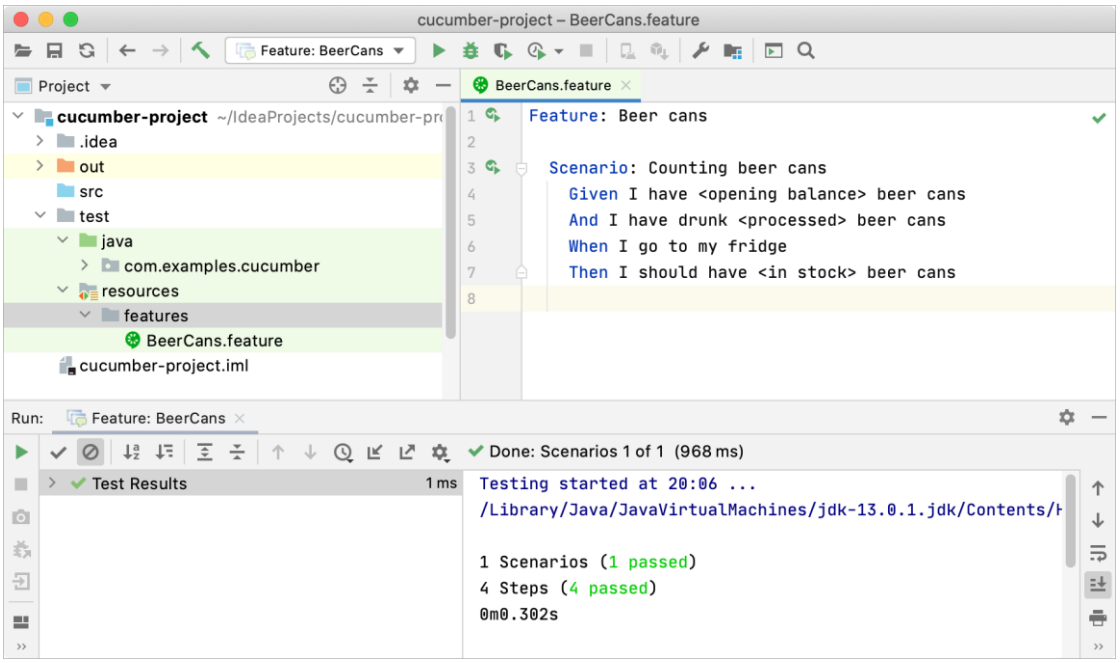

Рисунок 33 – Окно вывода результатов тестирования с помощью фреймворка Cucumber

На этой вкладке можно перезапустить тесты, экспортировать и импортировать результаты тестов, посмотреть, сколько времени ушло на запуск каждого теста и так далее.

E2E-тестирование веб-приложения осуществляется с помощью фреймворка Selenium.

Создаем новый Selenium-проект, как показано на рисунке 34.

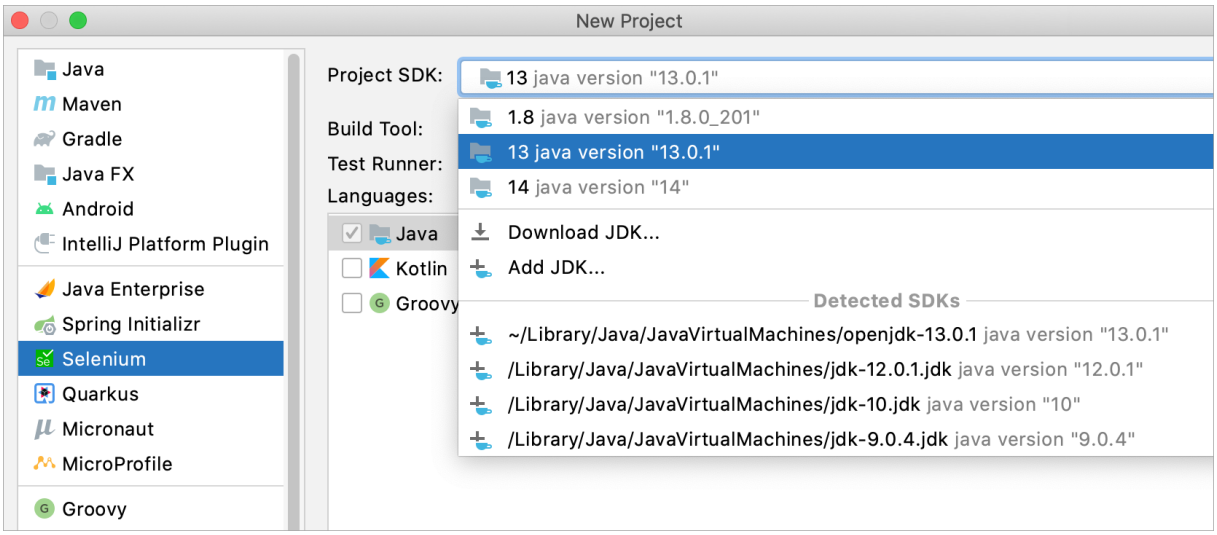

Рисунок 34 – Окно создания Selenium-проекта

Плагин Selenium UI Testing предоставляет помощь в написании кода для XPath и CSS, которые используются в Selenium API, а также во многих других библиотеках для тестирования пользовательского интерфейса.

Это включает подсветку синтаксических ошибок в селекторах по мере ввода и завершение кода для стандартных элементов CSS/HTML (рисунок 35).

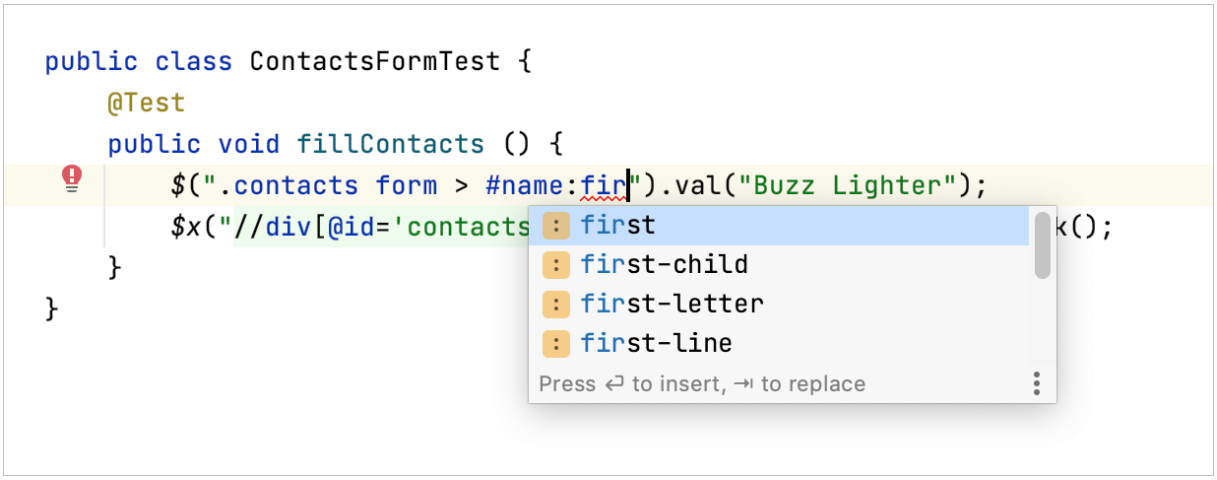

Рисунок 35 – Окно создания CSS/HTML-теста

Плагин Selenium UI Testing предоставляет инструмент на основе Chromium, который позволяет открывать веб-страницы и генерировать код, совместимый с шаблоном объекта страницы, прямо в среде IDE.

Это сокращает время, затрачиваемое на шаблонный код, и позволяет сосредоточиться на логике тестирования.

Для запуска теста надо поместить курсор на тестовый класс, чтобы запустить все тесты в этом классе, или на тестовый метод, и использовать комбинацию Ctrl+Shift+F10.

Можно также нажать на значок кнопки «Выполнить» рядом с тестовым классом или тестовым методом (рисунок 36).

Рисунок 36 – Запуск Selenium-теста

По завершении выполнения тестов результаты отображаются на вкладке «Выполнение тестов» в окне инструмента «Выполнение».

На этой вкладке можно перезапустить тесты, экспортировать и импортировать результаты тестов, посмотреть, сколько времени ушло на запуск каждого теста и т.д.

Апробация подтвердила работоспособность предлагаемого решения ИСАТ ПО всем уровням автоматизированного тестирования веб-приложений.

4.2 Оценка эффективности проектных решений

«Для оценки эффективности управления ИСАТ ПО используем формулу (1):

$$К_{эу} = \frac{\sum_{i=1}^{n} P_{yi}}{n} \quad (1)$$

где n - количество функций управления, реализуемых СФИОО;

P_{yi} - вероятность выработки СФИОО эффективного управляющего воздействия при реализации *i*-й функции управления» [4].

«Для управления тестированием ПО используются следующие функции:

- автоматизированное управление процессом тестирования ПО;
- оценка результатов тестирования тестировщиком.

Как показывает практика, на выполнение функции «Оценка результатов тестирования тестировщиком» может негативно повлиять человеческий фактор.

Пусть вероятность выработки эффективного управляющего воздействия для данной функции равна 0.5.

В этом случае значение показателя функциональной эффективности

управления ИСАТ ПО будет равно:

$$К_{эу} = 1.5/2 = 0,75$$

Таким образом, коэффициент эффективности управления предлагаемой СФИОО Кэу> 0,5, что свидетельствует о высокой функциональной эффективности ИСАТ ПО» [4].

Выводы по главе 4

Результаты проделанной работы позволили сделать следующие выводы:

- прототип ИСАТ ПО разработан на базе интегрированной среды разработки IntelliJ IDEA. Для представления иерархической структуры функций ИСАТ построена диаграмма дерева функций;
- процесс реализации прототипа ИСАТ ПО представляет собой интеграцию среды разработки IntelliJ IDEA с фреймворками JUnit, Cucumber и Selenium;
- интеграция фреймворков в ИСАТ осуществляется через опцию «Плагины», последние версии которых можно скачать с сайтов разработчиков;
- апробация подтвердила работоспособность предлагаемого решения ИСАТ ПО всем уровням автоматизированного тестирования веб-приложений;
- коэффициент эффективности управления предлагаемой ИСАТ Кэу> 0,5, что свидетельствует о высокой функциональной ее эффективности.

Таким образом, апробация и расчеты подтвердили работоспособность и высокую функциональную эффективность управления ИСАТ ПО.

Заключение

Магистерская диссертация посвящена актуальной проблеме разработки моделей и алгоритмов интегрированной среды автоматизированного тестирования программного обеспечения (ИСАТ ПО).

В процессе выполнения магистерской диссертации были решены следующие задачи:

- произведен анализ состояния исследований и разработок в области построения интегрированных сред автоматизированного тестирования ПО. Как показал анализ, проблематика автоматизированного тестирования ПО с помощью современных информационных технологий широко рассмотрена в работах отечественных и зарубежных ученых. Эффективная стратегия автоматизации тестирования требует обеспечение автоматизации тестов на трех разных уровнях: модульном, интеграционном и E2. Эффективность работы команды тестирования во многом зависит от того, какие именно задачи было решено автоматизировать и как эта автоматизация была проведена. Наиболее эффективным подходом к автоматизации тестирования веб-приложений является использование специализированных фреймворков. Необходимо констатировать недостаточность работ, посвященных разработке моделей и алгоритмов интегрированной среды автоматизированного тестирования ПО, что подтверждает актуальность темы исследования;
- произведен анализ методологий построения интегрированных сред автоматизированного тестирования программного обеспечения. Как показал анализ, согласно концепции гибкого проектирования (Agile), автоматизированное тестирование должно быть не изолированной задачей, а непрерывным процессом, неотъемлемо вписанным в жизненный цикл ПО. С помощью сравнительного анализа по

уровням тестирования выбраны фреймворки, которые в сообществе разработчиков считаются наилучшими. Выбранные фреймворки рекомендованы для использования при разработке моделей ИСАТ ПО;

- разработаны модели и алгоритмы ИСАТ ПО. На начальной фазе логического проектирования определяются основные требования, ограничения и ключевая функциональность продукта, и создается базовая версия модели прецедентов. Для проектирования ИСАТ ПО применен подход, основанный на применении технологической платформы, в качестве которой используется IDE или интегрированная среда разработки. С учетом результатов анализа IntelliJ IDEA выбрана в качестве среды для разработки ИСАТ веб-приложений. Разработаны алгоритмы тестирования веб-приложений;
- выполнены апробация и оценка эффективности проектных решений. прототип ИСАТ ПО разработан на базе интегрированной среды разработки IntelliJ IDEA. Для представления иерархической структуры функций ИСАТ построена диаграмма дерева функций;
- процесс реализации прототипа ИСАТ ПО представляет собой интеграцию среды разработки IntelliJ IDEA с фреймворками JUnit, Cucumber и Selenium. Интеграция фреймворков в ИСАТ осуществляется через опцию «Плагины», последние версии которых можно скачать с сайтов разработчиков. Апробация подтвердила работоспособность предлагаемого решения ИСАТ ПО всем уровням автоматизированного тестирования веб-приложений. Коэффициент эффективности управления предлагаемой ИСАТ Кэу> 0,5, что свидетельствует о высокой функциональной ее эффективности.

Гипотеза исследования подтверждена.

Список используемой литературы и используемых источников

1. Автоматизированное тестирование программного обеспечения – основные понятия [Электронный ресурс]. URL: http://www.protesting.ru/automation (дата обращения: 20.03.2024).
2. Автоматическое тестирование с использованием фреймворка Mocha [Электронный ресурс]. URL: https://learn.javascript.ru/testing-mocha (дата обращения: 20.03.2024).
3. Введение в Jasmine [Электронный ресурс]. URL: https://habr.com/ru/articles/167173/ (дата обращения: 20.03.2024).
4. Вдовин В.М., Суркова Л.Е., Шурупов А.А. Предметно-ориентированные экономические информационные системы. М.: Дашков и К, 2016. 388 с.
5. Гиматдинов Д.М., Герасимов А.Ю., Привалов П.А., Буткевич В.Н., Чернова Н.А., Горелова А.А. Автоматизированная система тестирования инструментов статического анализа кода // Труды Института системного программирования РАН. 2021. 33(3). С. 41-50.
6. Дастин Э., Рэшка Дж., Пол Дж. Автоматизированное тестирование программного обеспечения. Внедрение, управление и эксплуатация: Пер. с англ. –Москва: Лори. 2003. 289 с.
7. Дробинцев П. Д., Даишев М.Ш., Котляров В.П. Интегрированная среда автоматизации тестирования на основе технологии Eclips // Информатика, телекоммуникации и управление. 2010. №4 (103) [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/integrirovannaya-sreda-avtomatizatsii-testirovaniya-na-osnove-tehnologii-eclips (дата обращения: 20.03.2024).
8. Интегрированные среды разработки программ [Электронный ресурс]. URL: http://bourabai.ru/einf/ide.htm (дата обращения: 25.02.2024).
9. Коцюба И.Ю., Чунаев А.В., Шиков А.Н. Методы оценки и измерения

характеристик информационных систем: учебное пособие. -Санкт-Петербург: - Университет ИТМО, 2016. 264 с.

10. Леоненков А. В. Объектно-ориентированный анализ и проектирование с использованием UML и IBM Rational Rose : учебное пособие. М. : ИНТУИТ, Ай Пи Ар Медиа, 2020. 317 с. [Электронный ресурс]. URL: https://www.iprbookshop.ru/97554.html (дата обращения: 20.03.2024).

11. Поллиц Г. Использование RUP для небольших проектов: расширение экстремального программирования [Электронный ресурс]. URL: http://www.interface.ru/home.asp?artId=4720 (дата обращения: 20.03.2024).

12. Сайт Sahi Pro [Электронный ресурс]. URL: https://www.sahipro.com (дата обращения: 20.03.2024).

13. Сайт Selenium IDE [Электронный ресурс]. URL: https://www.selenium.dev/selenium-ide (дата обращения: 20.03.2024).

14. Сайт Watir [Электронный ресурс]. URL: http://watir.com (дата обращения: 20.03.2024).

15. Сквозное тестирование при помощи Cypress [Электронный ресурс]. URL: https://qarocks.ru/cypress-end-to-end-testing/ (дата обращения: 20.03.2024).

16. Сунгатуллина А. Т. Системный анализ и проектирование информационных систем на основе объектно-ориентированного подхода : учебно-методическое пособие по дисциплине «Методы и средства проектирования информационных систем». М. : Российский университет транспорта (МИИТ), 2020. 118 с. [Электронный ресурс]. URL: https://www.iprbookshop.ru/115990.html (дата обращения: 20.03.2024).

17. Трутнев Д. Р. Архитектуры информационных систем. Основы проектирования: Учебное пособие. Санкт-Петербург: Университет ИТМО, 2012. 66 с.

18. Фреймворк как программная платформа [Электронный ресурс]. URL: https://intellect.icu/frejmvork-kak-programmnaya-platforma-klassifikatsiya-i-vidy-frejmvorkov-framework-9515 (дата обращения: 20.03.2024).

19. Apache JMeter [Электронный ресурс]. URL: https://jmeter.apache.org (дата обращения: 20.03.2024).

20. Apache NetBeans [Электронный ресурс]. URL: https://netbeans.apache.org (дата обращения: 20.03.2024).

21. Automated Testing Strategy: How to Build & Examples [Электронный ресурс]. URL: https://performancelabus.com/automated-testing-strategy-how-to-build-examples/#2 (дата обращения: 20.03.2024).

22. Cohn M. The Forgotten Layer of the Test Automation Pyramid [Электронный ресурс]. URL: https://www.mountaingoatsoftware.com/blog/the-forgotten-layer-of-the-test-automation-pyramid (дата обращения: 20.03.2024).

23. Create a Domain Model [Электронный ресурс]. URL: https://sparxsystems.com/enterprise_architect_user_guide/15.2/model_domains/create_a_domain_model.html (дата обращения: 20.03.2024).

24. E2E автоматизация веб-приложений: выбери свой инструмент! [Электронный ресурс]. URL: https://medium.com/@themillenialscrolls (дата обращения: 20.03.2024).

25. Eclipse IDE [Электронный ресурс]. URL: https://www.eclipse.org/eclipseide (дата обращения: 20.03.2024).

26. Fowler M. Test pyramid. [Электронный ресурс]. URL: https://martinfowler.com/bliki/TestPyramid.html (дата обращения: 20.03.2024).

27. Garcıa B, Duenas J.C. “Automated Functional Testing based on the Navigation of Web Applications”, EPTCS 61, 2011, pp. 49-65.

28. Girgis M.R. et al. “An Automated Web Application Testing System”, International Journal of Computer Applications, vol. 99(7). 2014, pp. 37-44.

29. Hanna M. et al. “Automated Software Testing Framework for Web Applications”, International Journal of Applied Engineering Research, vol. 13(11). 2018, pp. 9758-9767.

30. JetBrains IntelliJ IDEA [Электронный ресурс]. URL: https://www.jetbrains.com/ (дата обращения: 20.03.2024).

31. JUnit 5 [Электронный ресурс]. URL: https://junit.org/junit5/ (дата

обращения: 20.03.2024).

32. Kalmo M. “Automated Testing of Java Web Applications”, Department of Computer Science and Engineering, University of Göteborg. 2009, 48 p.

33. MySQL Workbench [Электронный ресурс]. URL: http://www.mysql.com/products/workbench/features.html (дата обращения: 25.02.2024).

34. RUP Lifecycle [Электронный ресурс]. URL: https://swi.cs.vsb.cz/RUPLarge/core.base_rup/customcategories/rup_lifecycle_100BF298.html (дата обращения: 20.03.2024).

35. RUP методология разработки [Электронный ресурс]. URL: https://qaevolution.ru/metodologiya-menedzhment/rup (дата обращения: 20.03.2024).

36. Testing overview [Электронный ресурс]. URL: https://2018.stateofjs.com/testing/overview (дата обращения: 20.03.2024).

37. The component diagram [Электронный ресурс]. URL: https://developer.ibm.com/articles/the-component-diagram (дата обращения: 20.03.2024).

38. Types of Testing Environments [Электронный ресурс]. URL: https://www.testenvironmentmanagement.com/types-of-testing-environments (дата обращения: 20.03.2024).

39. Visual Paradigm Online [Электронный ресурс]. URL: https://online.visual-paradigm.com (дата обращения: 20.03.2024).