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

Институт цифровых технологий
(наименование института полностью)

Департамент бакалавриата
(наименование)

09.03.03 Прикладная информатика
(код и наименование направления подготовки / специальности)

Разработка программного обеспечения
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка системы для автоматизации тестирования информационной системы»

Обучающийся Р.В. Рыжов
(Инициалы Фамилия) (личная подпись)

Руководитель к.п.н., доцент, О.В. Оськина
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2025

Аннотация

Тема бакалаврской работ «Разработка системы для автоматизации тестирования информационной системы».

Ключевые слова: разработка инструментария, автоматизированное тестирование, информационная система.

Современные информационные системы включают множество модулей, интеграций с внешними сервисами, работают в распределённых средах (облако, гибридные решения). Ручное тестирование таких систем становится дорогим и медленным, тогда как автоматизация ускоряет проверку и повышает её точность.

Создание системы для автоматизации тестирования информационной системы (САТ ИС) актуально и представляет практический интерес для широкого круга разработчиков ПО.

Объектом исследования бакалаврской работы является процесс тестирования ИС.

Предметом исследования является САТ ИС.

Цель выпускной квалификационной работы – разработка программного обеспечения САТ ИС.

Бакалаврская работа состоит из 43 страниц текста, 18 рисунков, 14 таблиц и 32 источников.

Оглавление

Введение

«Тестирование информационных систем – это критически важный этап разработки и сопровождения программного обеспечения информационной системы, который напрямую влияет на качество, безопасность и надежность последней» [31].

Современные информационные системы включают множество модулей, интеграций с внешними сервисами, работают в распределённых средах (облако, гибридные решения). Ручное тестирование таких систем становится дорогим и медленным, тогда как автоматизация ускоряет проверку и повышает её точность.

Помимо снижения трудозатрат, особенно при повторяющихся сценариях тестирования, автоматизированное тестирование снижает негативное влияние «человеческого фактора» тестировщиков на результаты рутинных проверок программного обеспечения в условиях DevOps и Agile-разработки, при которых релизы выходят часто, иногда несколько раз в день.

Вместе с тем, необходимо учесть, что разработка автоматизированных тестов для конкретного программного обеспечения (ПО) – это не просто дополнительная опция, а критически важная часть процесса разработки и сопровождения информационной системы.

Таким образом, создание системы для автоматизации тестирования информационной системы актуально и представляет практический интерес для широкого круга разработчиков ПО.

Объектом исследования бакалаврской работы является процесс тестирования информационной системы.

Предметом исследования является система для автоматизированного тестирования информационной системы.

Цель выпускной квалификационной работы – разработка программного обеспечения системы для автоматизации тестирования информационной системы.

«Для достижения данной цели необходимо решить следующие задачи:

– сформулировать требования и произвести постановку задачи на разработку программного обеспечения системы автоматизации тестирования информационной системы;
– выполнить проектирование программного обеспечения системы для автоматизации тестирования информационной системы;
– реализовать программного обеспечения системы для автоматизации тестирования информационной системы и провести его тестирование.

Методы исследования – метод автоматизированного тестирования информационных систем, методы и технологии разработки ПО.

Практическая значимость бакалаврской работы заключается в разработке ПО эффективной системы автоматизированного тестирования информационной системы, учитывающей архитектурные и функциональные особенности последней» [15].

Данная работа состоит из введения, трех глав, заключения и списка используемой литературы и источников.

Бакалаврская работа состоит из 43 страниц текста, 18 рисунков, 14 таблиц и 32 источников.

Глава 1 Постановка задачи на разработку программного обеспечения системы автоматизации тестирования информационной системы

1.1 Архитектурные и функциональные особенности средств автоматизированного тестирования информационных систем

Автоматизированное тестирование похоже на создание робота для тестирования программного обеспечения вместо того, чтобы тестировщик делал это вручную.

В таблице 1 представлены основные характеристики автоматизированного тестирования.

Таблица 1 – Основные характеристики автоматизированного тестирования

Преимущества	Недостатки
Высокая скорость выполнения	Высокие первоначальные затраты
Высокие первоначальные затраты	Высокая стоимость поддержки
Повторяемость и отсутствие человеческого фактора	Не может заменить творческое и исследовательское тестирование
Идеально для регрессионного тестирования	Сложность тестирования визуальной составляющей и UX
Возможность интеграции в CI/CD	Требует серьезных технических навыков
Быстрая обратная связь для разработчиков	Хрупкость (риск появления "flaky tests")

Автоматизированное тестирование – это не цель, а мощный инструмент.

Его главная особенность в том, что оно не заменяет ручное тестирование, а дополняет его, беря на себя повторяющиеся, трудоемкие и критически важные проверки.

Успех внедрения автоматизации зависит от понимания этих особенностей. Нужно осознанно подходить к тому, что автоматизировать, когда начинать и какие ресурсы на это выделять, чтобы сильные стороны перевесили слабые.

Автоматизированное тестирование использует программные

инструменты для выполнения тестов в другом программном приложении. Эти тесты пишутся заранее и могут имитировать взаимодействие пользователя с приложением. Затем инструменты автоматизации сравнивают фактические результаты тестов с ожидаемыми, выявляя любые несоответствия или ошибки [22].

Указанные инструменты являются системами автоматизированного тестирования (CAT) [16].

CAT предназначены для ускорения проверки качества программного обеспечения, уменьшения человеческих ошибок и повышения эффективности процессов тестирования. Их архитектура и функциональность варьируются в зависимости от типа тестирования (модульное, интеграционное, системное, UI-тестирование и т. д.), но можно выделить общие черты.

Современные CAT обладают гибкой архитектурой, позволяющей интегрироваться в CI/CD, поддерживать различные виды тестирования и масштабироваться. Выбор инструмента зависит от технологического стека, требований к тестированию и уровня автоматизации в проекте.

Архитектура типовой CAT показана на рисунке 1.

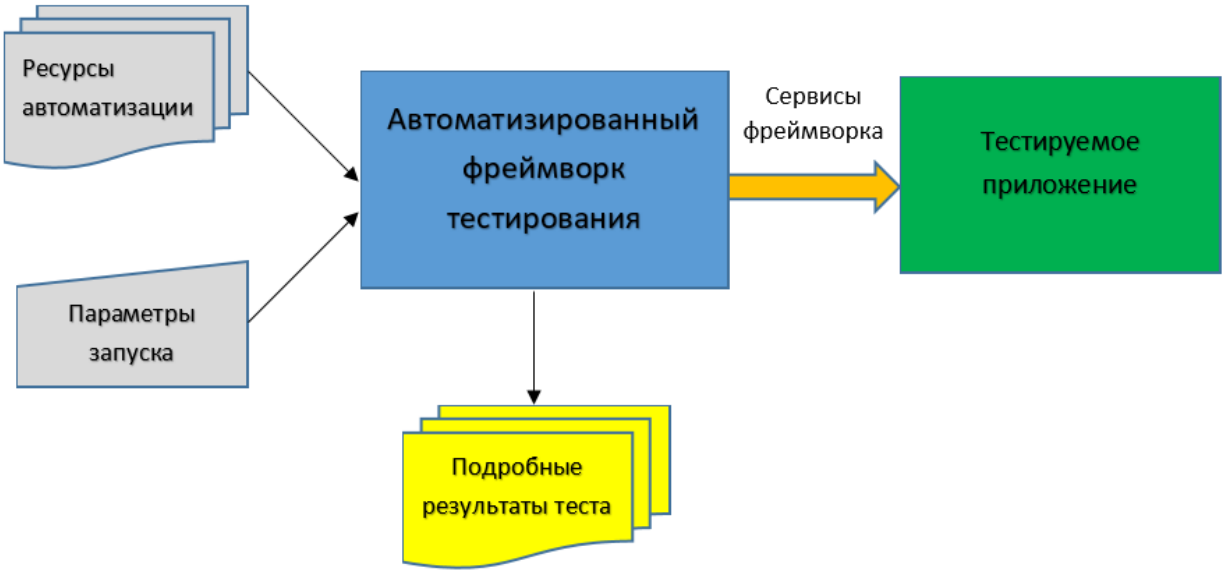

Рисунок 1 – Архитектура типовой CAT

Фреймворк автоматизации тестирования – это набор руководств, инструментов и практик, разработанных для поддержки автоматизированного тестирования. Он обеспечивает структурированный подход к написанию, управлению и выполнению автоматизированных тестов, обеспечивая согласованность и эффективность.

Рассмотрим архитектурные особенности CAT.

Типовые CAT имеют модульную архитектуру и обычно состоят из следующих взаимосвязанных компонентов:

- тест-раннер – исполняет тесты, управляет их порядком и параллелизацией (например, JUnit, TestNG, pytest);
- фреймворк тестирования – предоставляет API для написания тестов (Selenium, Cypress, Appium);
- оркестратор – управляет распределенным выполнением тестов (например, Jenkins, GitLab CI, Kubernetes);
- модуль формирования отчетности – собирает и визуализирует результаты (Allure, ExtentReports, ReportPortal);
- система управления тест-данными – генерирует и управляет тестовыми данными (Faker, SQL-скрипты);
- интеграции с CI/CD – связь с инструментами непрерывной интеграции (Jenkins, GitHub Actions).

Функциональность CAT предполагает поддержка различных видов тестирования [20]:

- модульное (Unit Testing) – JUnit, NUnit, pytest;
- интеграционное – Postman, RestAssured;
- UI-тестирование – Selenium, Playwright, Cypress;
- нагрузочное – JMeter, Gatling, Locust;
- тестирование безопасности – OWASP ZAP, Burp Suite.

На рисунке 2 представлена схема типового процесса автоматизированного тестирования приложения.

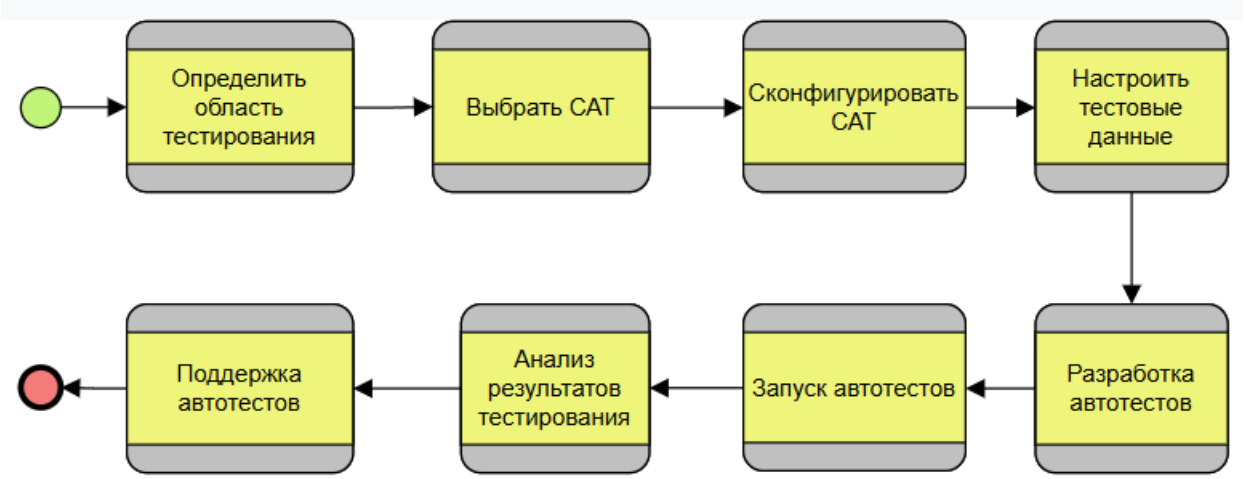

Рисунок 2 – Схема типового процесса автоматизированного тестирования приложения

Таким образом, необходимо разработать CAT информационной системы (ИС), архитектурные и функциональные особенности которой должны соответствовать подходам к построению систем данного типа, описанным в данном разделе.

1.2 Формирование требований на разработку системы автоматизированного тестирования информационной системы

Ключевые характеристики современной CAT приведены в таблице 2.

Таблица 2 – Ключевые характеристики современной CAT

Группа	Характеристика	Описание
Функциональность	Поддержка разных типов тестов	Позволяет закрыть все уровни «Пирамиды тестирования»
Операционность	Надежность. Параллельный запуск	Формирует доверие к результатам тестов. Команда начинает реагировать на

Продолжение таблицы 2

Группа	Характеристика	Описание
		падения, а не игнорировать их. Критически важно для скорости. Позволяет запускать тысячи тестов за минуты, а не часы
Сопровождение	Удобство поддержки и отчетность	Снижает затраты на изменение тестов и ускоряет анализ причин падения.
Интеграция	Интеграция с CI/CD и другими инструментами	Делает тестирование неотъемлемой частью процесса разработки, а не отдельной рутиной
Экономика	Приемлемая стоимость владения	Обеспечивает долгосрочную жизнеспособность и окупаемость системы

Для разработки требований к ПО САТ ИС используем методологию FURPS+.

FURPS+ – это удобный и структурированный подход к управлению требованиями, который помогает создавать более качественное и надежное ПО за счет комплексного анализа всех ключевых аспектов.

Этот метод особенно полезен, если САТ ИС разрабатывается «с нуля» или интегрируется в сложную DevOps-среду.

На основании проведенного анализа выработаны требования к ПО САТ ИС, представленные в таблице 3.

Таблица 3 – Требования к ПО САТ ИС

«Требование	Статус	Полезность	Риск	Стабильность
Functionality – Функциональные требования				
Поддержка различных типов тестирования	одобренное	средняя	средний	низкая
Интеграция с CI/CD	одобренное	критическая	средний	низкая
Управление тестовыми данными	одобренное	критическая	средний	низкая

Работа с тестовыми средами	одобренное	критическая	средний	низкая» [13]

Продолжение таблицы 3

«Требование	Статус	Полезность	Риск	Стабильность
Usability– Требования к удобству использования				
Интерфейс для написания тестов	одобренное	критическая	средний	низкая
Отчетность и визуализация	одобренное	критическая	средний	низкая
Простота отладки	одобренное	критическая	средний	низкая
Reliability– Требования к надежности				
Допустимая частота/периодичность сбоев: 1 раз в 300 часов	одобренное	важная	средний	средняя
Среднее время сбоев: 1 раб. день	одобренное	важная	средний	средняя
«Возможность восстановления системы после сбоев: 1 раб. день	одобренное	важная	средний	средняя
Режим работы: рабочий день	одобренное	важная	средний	средняя
Performance – Требования к производительности				
Допустимое количество одновременно работающих пользователей: 2	предложенное	важная	средний	средняя
Время реакции на возникновение аварийной ситуации: 1 с	предложенное	важная	средний	средняя
Время устранения критических проблем: в течение рабочего дня	предложенное	важная	средний	средняя
Оптимизация времени выполнения	предложенное	важная	средний	средняя]
Минимизация ресурсопотребления	предложенное	важная	средний	средняя
Проектные ограничения				
Отсутствие избыточного функционала	предложенное	важная	средний	средняя
Низкая совокупная стоимость владения	предложенное	критическая	средний	низкая» [13]

Указанная таблица требований является основой для разработки ПО САТ ИС.

1.3 Обзор и анализ аналогов программного обеспечения системы автоматизированного тестирования для информационных систем

Архитектура САТ для ИС, разработанных на платформе 1С: Предприятие 8.х, показана на рисунке 3.

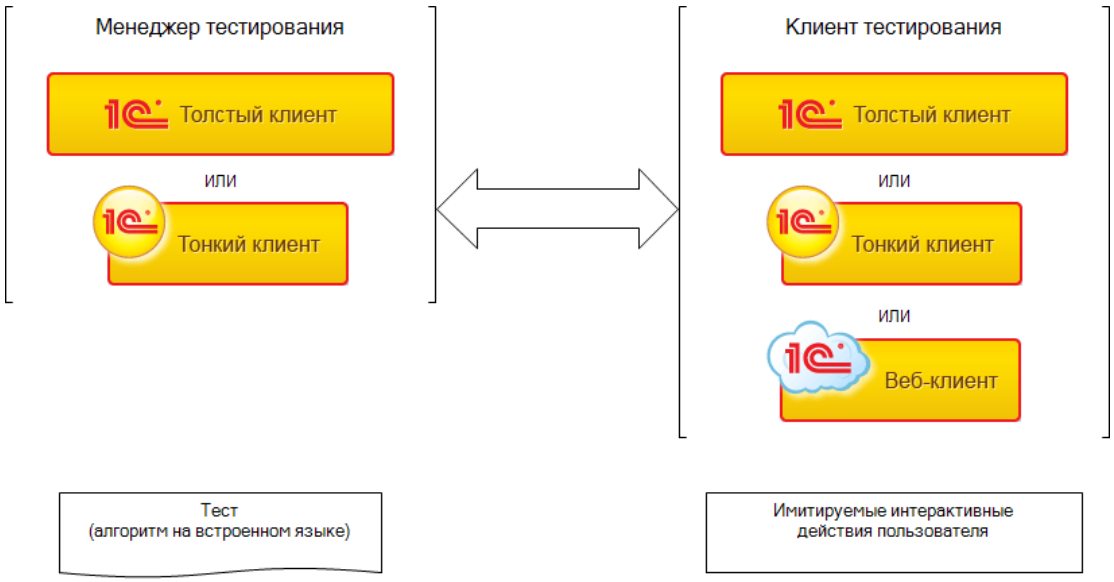

Рисунок 3 – Архитектура САТ для ИС, разработанных на платформе 1С: Предприятие 8.х

«Результат выполнения автотеста может контролироваться визуально, либо программно, путем сравнения полученных результатов с эталонными значениями» [4].

Пример автотеста и результат его выполнения для данной САТ показаны на рисунках 4 и 5 соответственно.

```
////////////////////////////////////////////////////////////
// *** Выполнить тест - добавить и записать новый товар.

// Открыть форму нового товара.
ГлавноеОкноТестКлиента.ВыполнитьКоманду("e1cib/command/Справочник.Товары.Команда.Создать");

// Если форма нового товара не открылась за 60 секунд, прекратить тестирование.
Если НЕ ТестКлиент.ОжидатьОтображениеОбъекта(Тип("ТестируемаяФорма"), "Товар*") Тогда
    Сообщить("Не удалось открыть форму нового товара в течение 60 секунд.");
    Возврат;

КонецЕсли;

// Получить форму нового товара.
ФормаНовогоТовара = ТестКлиент.НайтиОбъект(Тип("ТестируемаяФорма"), "Товар*");

// Заполнить поле "Наименование".
ПолеНаименование = ФормаНовогоТовара.НайтиОбъект(Тип("ТестируемоеПолеФормы"), "Наименование");
ПолеНаименование.ВвестиТекст("Новый товар (автотест)");

// Заполнить поле "Артикул".
ПолеАртикул = ФормаНовогоТовара.НайтиОбъект(Тип("ТестируемоеПолеФормы"), "Артикул*");
ПолеАртикул.Активизировать();
ПолеАртикул.ВвестиТекст("ПС-0001");

// Записать и закрыть новый товар, нажав на кнопку "Записать и закрыть".
КнопкаЗаписатьИЗакрыть = ФормаНовогоТовара.НайтиОбъект(Тип("ТестируемаяКнопкаФормы"), "Записать и закрыть");
КнопкаЗаписатьИЗакрыть.Нажать();

Сообщить("Тест выполнен.");
```

Рисунок 4 – Пример автотеста (язык 1С8)

```
<?xml version="1.0" encoding="UTF-8"?>
<uilog xmlns:d1p1="http://v8.1c.ru/8.3/uilog">
    <ClientApplicationWindow isMain="true">
        <Form title="Электронная почта">
            <close/>
        </Form>
        <Form title="Поиск в данных">
            <close/>
        </Form>
        <Form title="Товары">
            <activate/>
            <FormTable title="Список">
                <FormField title="Наименование">
                    <activate/>
                </FormField>
            </FormTable>
            <FormGroup title="Командная панель">
                <FormButton title="Создать">
                    <click/>
                </FormButton>
            </FormGroup>
        </Form>
    </ClientApplicationWindow>
    <ClientApplicationWindow caption="Товар (создание)">
        <Form title="Товар (создание)">
            <activate/>
            <FormField title="Наименование">
                <inputText text="Новый товар"/>
            </FormField>
            <FormField title="Артикул">
                <activate/>
                <inputText text="123456"/>
            </FormField>
            <FormGroup title="Командная панель">
                <FormButton title="Записать и закрыть">
                    <click/>
                </FormButton>
            </FormGroup>
            <close/>
        </Form>
    </ClientApplicationWindow>
</uilog>
```

Рисунок 5 – Результат выполнения автотеста (XML-файл)

Playwright – инструмент от Microsoft для кросс-браузерного

тестирования с поддержкой нескольких языков [29].

Playwright имеет очень простой метод выбора и работы с iframes: FrameLocator [2].

iframe, или встроенный фрейм, – это встроенный html-документ. В отличие от Shadow DOM, iframe – это полностью отдельный документ. Он имеет свои собственные теги <html>, <head> и <body> (рисунок 6).

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div>
        <iframe src="some.url">
            <html>
                <head>
                    <body>
                        <div aria-label="something">
                            Some content
                        </div>
                    </body>
                </head>
            </html>
        </iframe>
        <iframe src="none">
            <html>
                <body></body>
            </html>
        </iframe>
    </div>
</body>
</html>
```

Рисунок 6 – Пример iframe

Тестовый фреймворк Playwright более прост и легок, чем большинство альтернатив, включая Cypress и TestCafe , среди многих других. Чтобы придать ему импульс, он также предлагает тестирование API . Таким образом, это отличный выбор для сквозного тестирования.

Поддержка Playwright в PyCharm обеспечивается плагином Test Automation , и большинство функций, описанных в этом разделе, полагаются на него. Однако базовая функциональность, такая как запуск и отладка тестов Playwright, работает и без плагина.

Окна запуска и результатов тестирования показаны на рисунках 7 и 8.

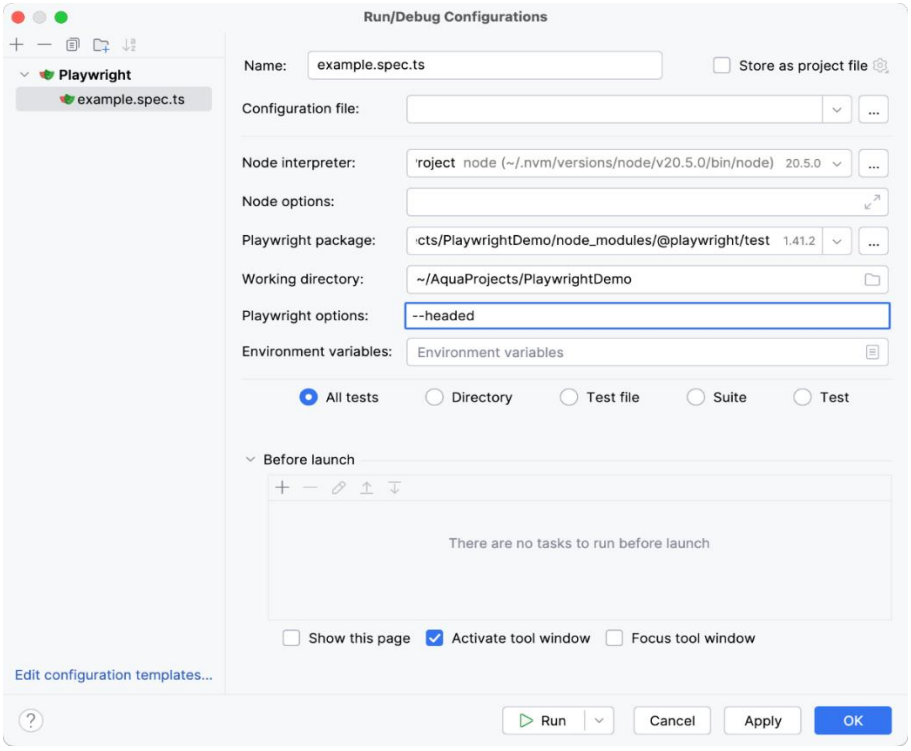

Рисунок 7 – Окно запуска теста

Рисунок 8 – Окно результатов тестирования

Selenium – это фреймворк с открытым исходным кодом,

предназначенный для автоматизации веб-браузеров . Он позволяет пользователям тестировать функциональность веб-сайта в разных браузерах, обеспечивая согласованность и совместимость [30].

Тесты в Selenium IDE записываются в архитектуру на основе таблиц HTML с использованием ключевых слов. Код может быть экспортирован на разных языках, таких как Ruby, Java, C# и т.д. [21].

Архитектура CAT Selenium IDE показана на рисунке 9.

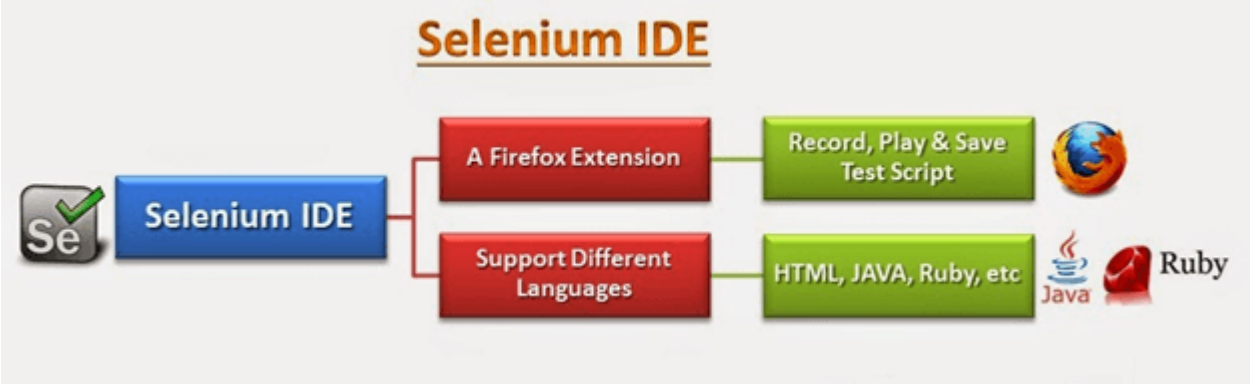

Рисунок 9 – Архитектура CAT Selenium IDE

Преимущества:

- Selenium IDE очень проста и удобна в использовании, а также гибка для пользователей;
- имеет возможность конвертировать тест на разные языки программирования;
- Selenium IDE обеспечивает возможности ведения журнала;
- в Selenium IDE пользователь может выполнять отладку и устанавливать точки останова;
- для использования этого инструмента не требуется никакого опыта программирования.

Для сравнения характеристик рассмотренных САТ разработана таблица 4.

Таблица 4 – Сравнение характеристик аналогов ПО САТ ИС

Критерий	Selenium IDE	Playwright	Менеджер автотестирования 1С8
Тип тестирования	Веб-UI (только браузерные тесты)	Веб-UI, API, мобильные эмуляции	Функциональное тестирование 1С-приложений
Поддержка языков	JavaScript (встроенный редактор)	JavaScript, TypeScript, Python, Java, C#	Встроенный 1С-скриптовый язык
Запись тестов	Да	Да	Да
Поддержка браузеров	Chrome, Firefox, Edge	Chromium, Firefox, WebKit	Только клиент 1С
Параллельный запуск	Нет	Да	Да
Интеграция с CI/CD	Частично	Да	Да
Кроссплатформенность	Только веб	Windows, Linux, macOS	Только в среде 1С:Предприятие
Работа с API	Нет	Да	Нет
Поддержка мобильных приложений	Нет	Да	Нет
Отчеты	Базовые (HTML, JSON)	Подробные (Allure, JUnit, JSON)	Встроенные отчёты 1С
Сложность изучения	Низкая	Средняя	Средняя
Лицензия	Open Source	Open Source	Проприетарный

Таким образом, выбор САТ зависит от задачи:

– если нужно тестировать 1С – только менеджер автотестирования;
– для веб-приложений – Playwright (гибкость) или Selenium IDE (простота);
– для API и мобильной эмуляции – Playwright.

В последнее время широкое распространение получили информационно-аналитические системы (ИАС), разработанные на языке Python.

Эти ИАС являются узкоспециализированными.

Разработка САТ для таких ИАС актуальна и представляет практический интерес [9].

Выводы к главе 1

Результаты выполнения первой главы позволили сделать следующие выводы:

– современные САТ обладают гибкой архитектурой, позволяющей интегрироваться в CI/CD, поддерживать различные виды тестирования и масштабироваться. Выбор инструмента зависит от технологического стека, требований к тестированию и уровня автоматизации в проекте;
– ИАС, разработанные на языке Python, являются узкоспециализированными.

Разработка САТ для таких ИАС актуальна и представляет практический интерес.

Глава 2 Проектирование программного обеспечения системы автоматизированного тестирования для информационно-аналитической системы

2.1 Выбор методологии проектирования программного обеспечения информационной системы

Проанализированы методологии разработки ПО для CAT Agile, DevOps, TDD, BDD, RUP, V-модель и Shift-Left Testing (таблица 5) [5].

Таблица 5 – Сравнение методологий разработки ПО для CAT

Методология разработки ПО	Достоинства	Недостатки
Agile (Scrum, Kanban)	Гибкость, быстрая реакция на изменения, вовлечение заказчика.	Требует высокой дисциплины команды, сложность оценки сроков и бюджета.
DevOps	Быстрое развертывание, высокая автоматизация, улучшенное качество кода.	Сложность внедрения, требует высокой культуры автоматизации.
TDD	Высокое покрытие тестами, меньше багов, улучшенная архитектура.	Замедляет начальную разработку, требует высокой квалификации.
BDD	Понятные спецификации, вовлечение бизнес-аналитиков.	Избыточность для простых проектов, требует дополнительных инструментов.
RUP	Гибкость в управлении требованиями**, р**анний акцент на архитектуре, итеративность, интеграция с UML.	Сложность управления, дороговизна.
V-модель	Раннее тестирование, четкая связь этапов.	Негибкость, сложность при изменениях.

Shift-Left Testing	Раннее выявление багов, снижение стоимости исправлений.	Требует перестройки процессов, вовлечения тестировщиков на ранних этапах.

По результатам сравнения преимуществ и недостатков представленных методологий выбираем для разработки ПО ИАС методологию RUP.

Главным преимуществом методологии RUP является итеративность и интеграция с UML [27].

Можно быстро начать тестирование новыми компонентами, получить обратную связь и оперативно внести правки. Риски снижаются, так как ошибки в архитектуре обнаруживаются на ранних этапах.

RUP идеально подходит для сложных, долгосрочных проектов по созданию САТ, где критически важны масштабируемость, надежность, поддержка и четкое соответствие бизнес-требованиям.

Все сценарии автоматизации четко привязаны к требованиям ПО, которое будет тестироваться. Это предотвращает создание «лишних» или пропуск критичных тестов, повышая покрытие и целесообразность.

2.2 Логическое моделирование программного обеспечения информационной системы для анализа медицинских данных

Одним из основных этапов начальной фазы проектирования RUP является разработка логической модели ПО.

Логическая модель ПО – это абстрактное представление структуры, поведения и взаимодействия компонентов программной системы без привязки к физической реализации (языку, фреймворкам, инфраструктуре).

Логическая модель описывает [10]:

– сущности (классы, модули, сервисы);

– их отношения (зависимости, потоки данных);

– правила работы (алгоритмы, бизнес-логику).

Ключевые компоненты логической модели ПО CAT представлены в таблице 6.

Таблица 6 – Ключевые компоненты логической модели ПО CAT

Компонент	Описание	Пример для CAT
Сущности (Entities)	Основные объекты системы.	Тест-кейс, Тестовый набор, Отчет, Тестовое окружение.

Продолжение таблицы 6

Отношения (Relationships)	Связи между сущностями.	Тест-кейс входит в Набор, Отчет содержит Результаты тестов.
Бизнес-правила (Business Rules)	Логика принятия решений.	Если тест упал 3 раза подряд, отправить уведомление в Slack.
Состояния (States)	Жизненный цикл объектов.	Тест-кейс: Draft → Active → Deprecated.
Взаимодействия (Interactions)	Сценарии использования.	Запуск тестов через CI/CD", Анализ отчетов тестировщиком.

Основной результат логического моделирования в RUP – функциональная модель ПО CAT, представленная в виде диаграммы вариантов использования.

Диаграммы вариантов использования – это мощный инструмент для анализа, проектирования и документирования, который обеспечивает ясность, снижает риски и способствует созданию ПО, ориентированного на потребности пользователей.

В процессе разработки диаграммы вариантов использования ПО CAT были выделены следующие акторы [8]:

– истема CI/CD.

Варианты использования ПО CAT представлены в таблицах 7-12.

Таблица 7 – Описание прецедента: Подготовка автотеста

«Элемент диаграммы	Описание
Прецедент	Подготовка автотеста
ID	1
Краткое описание	Подготовка автотеста
Главный актор	Тестировщик
Второстепенный актор	Нет
Предусловие	Нет
Основной поток	Тестировщик подготавливает автотест на выполнение
Постусловие	Нет
Альтернативные потоки	Нет» [8]

Таблица 8 – Описание прецедента: Анализ теста

«Элемент диаграммы	Описание
Прецедент	Анализ теста
ID	2
Краткое описание	Анализ теста
Главный актор	Тестировщик
Второстепенный актор	Нет
Предусловие	Нет
Основной поток	Тестировщик анализирует результаты выполнения автотеста
Постусловие	Нет
Альтернативные потоки	Нет» [8]

Таблица 9 – Описание прецедента: Настройка и администрирование CAT

«Элемент диаграммы	Описание
Прецедент	Настройка и администрирование CAT
ID	3
Краткое описание	Настройка и администрирование CAT
Главный актор	Администратор CAT
Второстепенный актор	Нет
Предусловие	Нет
Основной поток	Администратор выполняет настройку и администрирование CAT
Постусловие	Нет
Альтернативные потоки	Нет» [8]

Таблица 10 – Описание прецедента: Автозапуск теста

«Элемент диаграммы	Описание
Прецедент	Автозапуск теста
ID	4
Краткое описание	Автозапуск теста
Главный актор	Система CI/CD
Второстепенный актор	Тестировщик
Предусловие	Нет
Основной поток	Система CI/CD осуществляет запуск автотеста под контролем Тестировщика
Постусловие	Нет
Альтернативные потоки	Нет» [8]

Таблица 11 – Описание прецедента: Интеграция теста в систему CI/CD

«Элемент диаграммы	Описание
Прецедент	Интеграция теста в систему CI/CD
ID	5
Краткое описание	Интеграция теста в систему CI/CD
Главный актор	Разработчик
Второстепенный актор	Система CI/CD
Предусловие	Нет
Основной поток	Разработчик ПО выполняет интеграцию теста в в систему CI/CD
Постусловие	Нет
Альтернативные потоки	Нет» [8]

Таблица 12 – Описание прецедента: Просмотр теста

«Элемент диаграммы	Описание
Прецедент	Просмотр теста
ID	6
Краткое описание	Просмотр теста
Главный актор	Разработчик
Второстепенный актор	Нет
Предусловие	Нет
Основной поток	Разработчик просматривает результаты теста
Постусловие	Нет
Альтернативные потоки	Нет» [8]

Для разработки диаграмм UML использован онлайн-сервис Visual Paradigm [32].

Разработанная диаграмма вариантов использования ПО САТ ИАС изображена на рисунке 10.

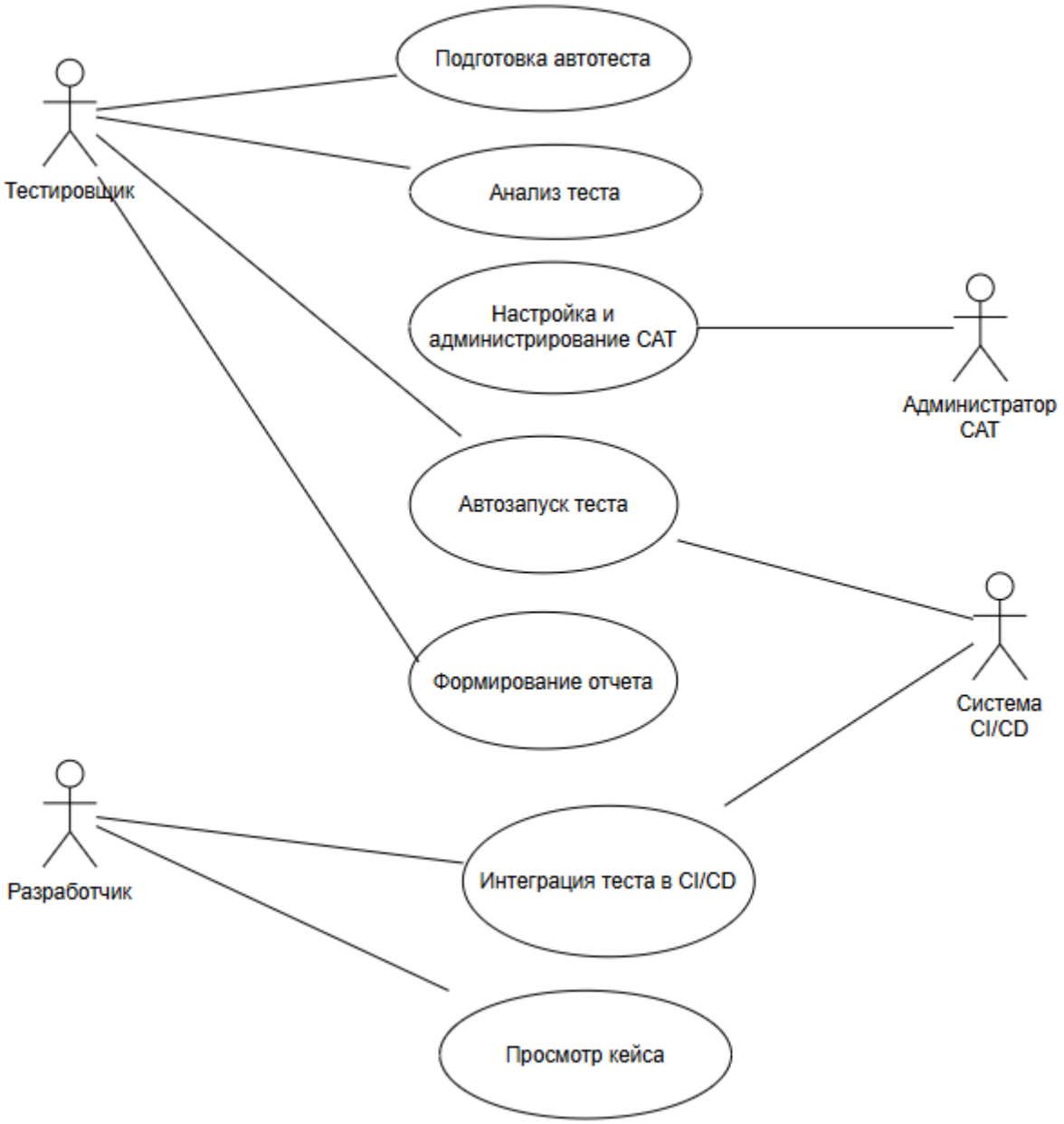

Рисунок 10 – Диаграмма вариантов использования ПО САТ ИАС

Диаграмма вариантов использования помимо фиксирования ключевых функций программы дает целостное, понятное заказчику и команде видение функциональности системы на самом старте проектирования ПО САТ ИАС.

Также разработана диаграмма деятельности ПО САТ ИАС, показанная

на рисунке 11.

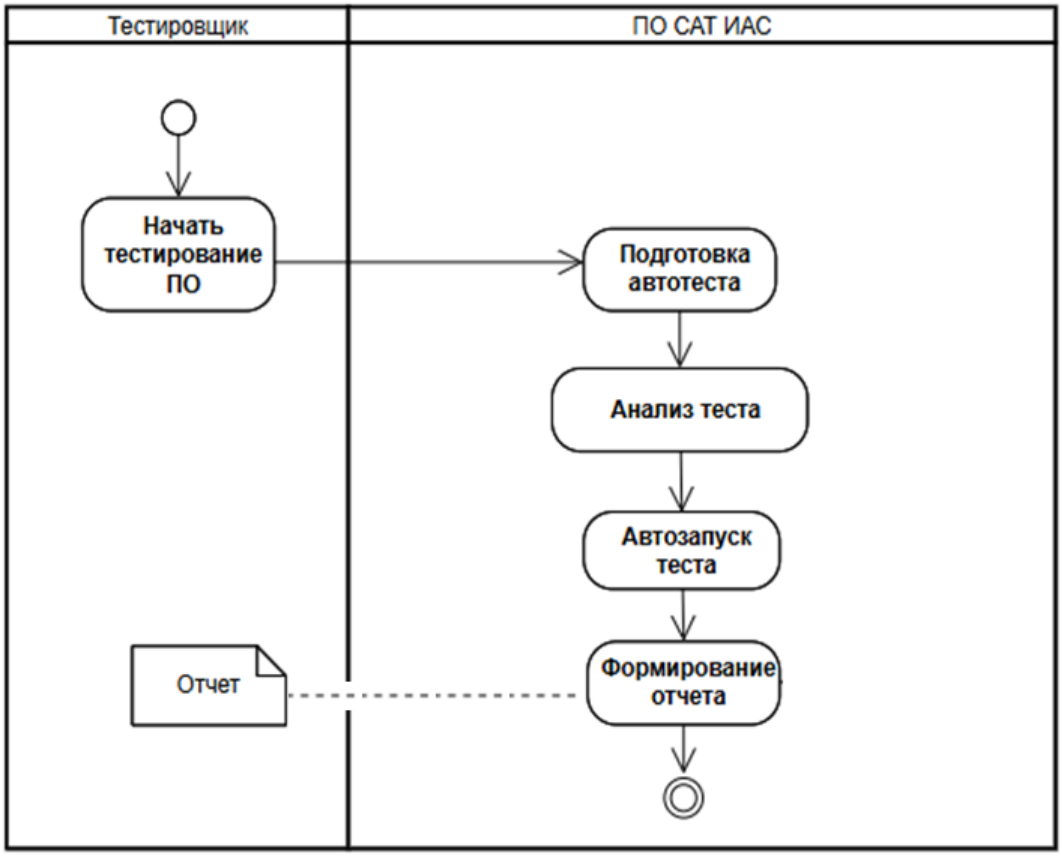

Рисунок 11 – Диаграмма деятельности ПО САТ ИАС

Выводы по главе 2

– выбрана методология проектирования RUP, главным преимуществом которой является интеграция с UML;
– разработана логическая модель ПО, которая играет ключевую роль для проектирования качественного ПО;

Диаграмма вариантов использования помимо фиксирования ключевых функций программы дает целостное, понятное заказчику и команде видение функциональности системы на самом старте проектирования ПО САТ.

Глава 3 Реализация и тестирование системы автоматизированного тестирования информационно-аналитической системы

3.1 Выбор средства разработки программного обеспечения системы автоматизированного тестирования информационно-аналитической системы

Для реализации ПО выбран язык Python.

Для выбора среды разработки сравним характеристики двух сред разработки: Jupyter Notebook и PyCharm [11].

Jupyter Notebook в основном используется для интерактивного анализа данных, визуализации и научных исследований. Он позволяет выполнять код по ячейкам, что удобно для экспериментов [24].

PyCharm – это полноценная IDE, предназначенная для разработки программного обеспечения, с упором на управление проектами, отладку и интеграцию с различными инструментами [26].

Для сравнения рассмотренных сред программирования используем таблицу 13.

Таблица 13 – Сравнение характеристик сред разработки на языке Pyhon

«Характеристика	Jupyter Notebook	PyCharm
Назначение	Интерактивный анализ данных, визуализация, обучение, прототипирование	Разработка ПО, управление проектами, отладка, полноценная среда для программирования
Интерфейс	Веб-интерфейс, линейное выполнение ячеек, минималистичный дизайн	Графический интерфейс (GUI) с множеством инструментов, вкладок, панелей управления
Интерактивность	Выполнение кода по ячейкам, мгновенный вывод результатов	Основной упор на полный запуск кода; интерактивность через

		консоль (например, Python Console)» [17]

Продолжение таблицы 13

Характеристика	Jupyter Notebook	PyCharm
Поддержка языков	Python (основной), R, Julia и другие через ядра	Python, JavaScript, HTML/CSS, SQL и другие (в Professional версии – Django, научные библиотеки)
Отладка	Базовая отладка через расширения (например, %debug).	Продвинутые инструменты: точки останова, пошаговое выполнение, просмотр переменных
Интеграция с инструментами	Подключение через плагины (например, Git, Matplotlib)	Глубокая интеграция: Git, Docker, базы данных, SSH, Jupyter Notebook (в Professional версии)
Совместная работа	Обмен файлами .ipynb; совместная работа через JupyterHub, Google Colab	Совместная работа через плагины (например, Code With Me) или внешние сервисы (Git, GitHub)
Производительность	Может замедляться при работе с большими данными из-за веб-интерфейса	Оптимизирован для больших проектов, но требует мощного железа
Использование ресурсов	Низкое (работает в браузере)	Высокое (требует больше оперативной памяти и CPU)
Стоимость	Бесплатный (open-source)	Community Edition – бесплатный; Professional Edition – платный (подписка)
Идеальные сценарии	Анализ данных, обучение, создание презентаций, быстрые эксперименты	Разработка приложений, командные проекты, работа с фреймворками (Django, Flask), отладка

По результатам сравнения и с учетом предпочтений разработчика выбираем среду разработки Jupyter Notebook.

Jupyter Notebook лучше подходит для исследовательской работы, анализа данных и интерактивного обучения.

В качестве библиотек тестирования Python используем pytest, doctest и

unittest [3].

Doctest – это модуль в Python, который позволяет проводить тестирование кода через примеры в документации.

Преимущества doctest в Python [25]:

- простота использования. Тесты пишутся прямо в docstring функций/классов, что минимизирует усилия на создание отдельных тестовых файлов. Примеры кода в документации сразу становятся тестами;
- интеграция с документацией. Примеры в docstring служат одновременно документацией и тестами, улучшая понимание кода и гарантируя актуальность примеров (если тесты проходят);
- экономия времени. Не требует написания сложного тестового кода – достаточно примеров использования. Идеально для быстрой проверки базовой функциональности;
- встроенная поддержка Python. Не нужны дополнительные установки (модуль в стандартной библиотеке), что упрощает начало работы и снижает зависимость от внешних пакетов;
- автоматизация тестирования. Легко интегрируется с unittest или pytest, позволяя запускать doctest вместе с другими тестами в CI/CD-процессах;
- проверка актуальности документации. Если примеры устарели, тесты не пройдут, что мотивирует поддерживать документацию в рабочем состоянии;
- наглядность. Тесты расположены рядом с кодом, что делает их более понятными и упрощает отладку;
- стимулирование документирования. Разработчики чаще пишут примеры использования, зная, что они будут автоматически проверяться.

Таким образом, Doctest – отличный инструмент для небольших

проектов, учебных материалов и быстрого тестирования, где важно совмещать документацию и проверку кода.

Unittest – это фреймворк для тестирования, вдохновленный JUnit. Он предоставляет больше возможностей для сложных тестов по сравнению с doctest.

Преимущества unittest в Python [23]:

- структурированный подход. Тесты организуются в классы и методы, что упрощает группировку и поддержку. Идеально для больших проектов;
- богатый набор ассертов. Встроенные методы для проверок: assertEqual, assertTrue, assertRaises и др. Упрощают валидацию сложных условий;
- фикстуры (setUp и tearDown). Позволяют настраивать предварительные условия и очищать ресурсы для каждого теста;
- интеграция с CI/CD и инструментами. Совместим с системами непрерывной интеграции (Jenkins, GitHub Actions). Результаты тестов можно экспортировать в XML/HTML;
- параметризация тестов. Возможность запускать один тест с разными входными данными через subTest или сторонние библиотеки (например, parameterized);
- моки и стабы. Встроенная поддержка мок-объектов (unittest.mock) для изоляции тестируемого кода от внешних зависимостей;
- генерация отчетов. Детализированный вывод о пройденных/проваленных тестах, включая трассировку ошибок;
- совместимость с другими фреймворками. Тесты unittest можно запускать через pytest или nose, расширяя функционал (например, параллельный запуск);
- поддержка сообщества. Большая база документации, примеров и готовых решений для сложных сценариев.

Unittest – мощный инструмент для комплексного тестирования, особенно в больших проектах. Он обеспечивает структуру, гибкость и надежность, но требует больше времени на настройку.

Преимущества pytest в Python [28]:

- минималистичный синтаксис. Тесты пишутся как обычные функции без необходимости создавать классы (в отличие от unittest);
- мощные фикстуры (fixtures). Гибкие фикстуры с настройкой областей видимости (function, class, module, session) и зависимостями;
- мощные фикстуры (fixtures). Гибкие фикстуры с настройкой областей видимости (function, class, module, session) и зависимостями;
- автоматическое обнаружение тестов. Pytest находит все файлы и функции, начинающиеся с test, без ручной настройки [6];
- расширенные возможности ассертов;
- богатая экосистема плагинов;
- детальные отчеты. Четкие сообщения об ошибках с контекстом, а также поддержка отладки через –pdb;
- совместимость с unittest и doctest. Можно запускать тесты, написанные для unittest или использующие doctest, через pytest;
- пропуск тестов и маркировка;
- гибкость для сложных сценариев.

Pytest – современный, гибкий и мощный фреймворк, который подходит для проектов любого масштаба. Он сочетает простоту написания тестов (как в doctest) с возможностями unittest, дополняя их фикстурами, параметризацией и экосистемой плагинов. Это де-факто стандарт для тестирования в Python-сообществе [18].

3.2 Реализация программного обеспечения системы автоматизированного тестирования информационно-аналитической системы

Для представления архитектуры ПО CAT разработана диаграмма компонентов UML.

Диаграммы компонентов в UML фокусируются на физической структуре системы, отображая компоненты, их интерфейсы и зависимости. Они особенно полезны в крупных проектах и распределенных системах. Вот их ключевые преимущества:

- визуализация архитектуры системы;
- упрощение проектирования распределенных систем;
- четкое определение интерфейсов;
- поддержка модульности и повторного использования;
- управление зависимостями;
- интеграция с DevOps и CI/CD;
- документирование системы;
- анализ масштабируемости;
- снижение рисков при интеграции;
- поддержка стандартов и compliance.

Диаграмма компонентов ПО CAT показана на рисунке 12.

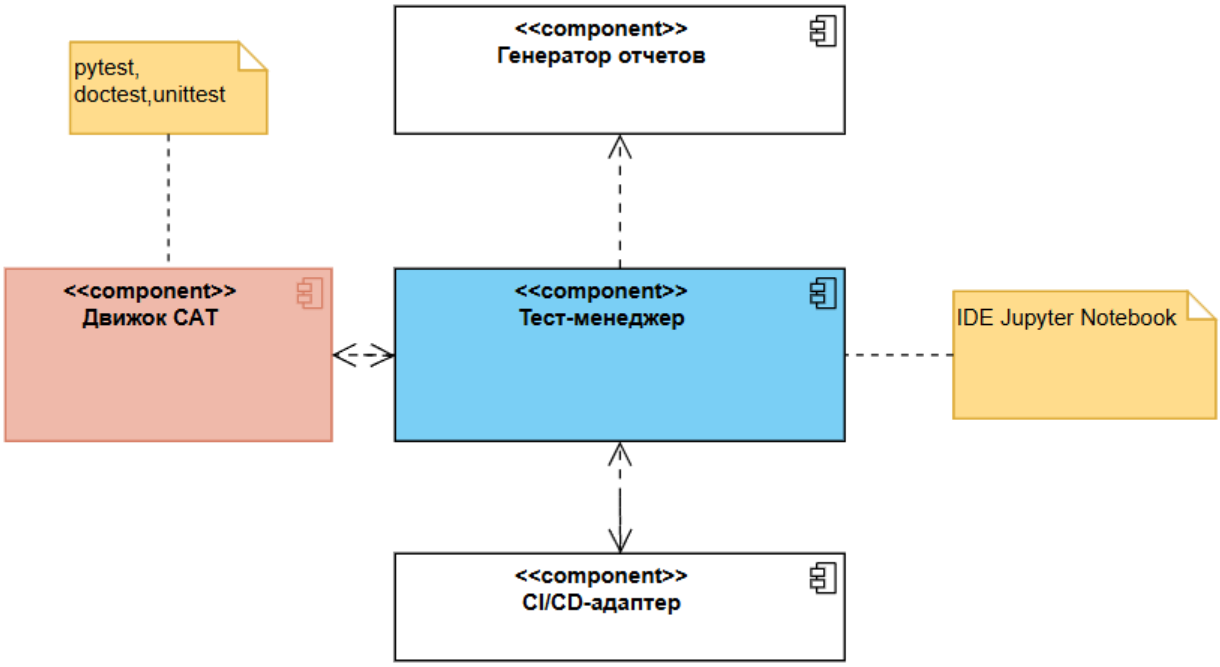

Рисунок 12 – Архитектура ПО CAT ИАС

Диаграммы компонентов – это инструмент для проектирования масштабируемой, модульной и поддерживаемой архитектуры. Они критически важны в DevOps-средах, микросервисах и системах с высокой степенью интеграции. Их использование снижает риски, улучшает коммуникацию и ускоряет разработку за счет четкого разделения ответственности между компонентами.

В среде Jupyter Notebook разработан код ПО CAT.

На рисунке 13 показан код модуля CAT с применением doctest [1].

```
"""
Модуль примера предоставляет одну функцию, factorial().
"""
def factorial(n):
    """Вернуть факториал n, точное целое число >= 0.
    Если результат достаточно мал, чтобы поместиться в int, вернуть int.
    Иначе вернуть long.
    ValueError: n должно быть >= 0
   Факториалы чисел с плавающей точкой допустимы, но число с плавающей точкой должно быть точным целым числом:
   >>> factorial(30.1)
    Traceback (последний вызов последний):
        ...
    ValueError: n должно быть точным целым числом
n также не должно быть слишком большим:
    """
    import math
    if not n >= 0:
        raise ValueError("n должно быть >= 0")
    if math.floor(n) != n:
        raise ValueError("n должно быть точно целым")
    if n+1 == n:  # catch a value like 1e300
        raise OverflowError("n слишком большое")
    result = 1
    factor = 2
    while factor <= n:
        result *= factor
        factor += 1
    return result
if __name__ == '__main__':
    import doctest #местирование с помощью doctest
    doctest.testmod(verbose=True)
```

Рисунок 13 – Код модуля CAT с применением doctest

Выполнена апробация прототипа программы с применением метода функционального тестирования.

Цель тестирования:

– проверить корректность работы функций, классов и модулей;

– обеспечить покрытие ключевых сценариев.

План тестирования:

– тестирование кода с применением doctest;

– тестирование кода с применением unittest;

– тестирование кода с применением pytest.

На рисунке 14 представлен результат тестирования с применением doctest.

```
Trying:
    factorial(5)
Expecting:
    120
ok
Trying:
    [factorial(n) for n in range(6)]
Expecting:
    [1, 1, 2, 6, 24, 120]
ok
Trying:
    [factorial(long(n)) for n in range(6)]
Expecting:
    [1, 1, 2, 6, 24, 120]
**********************************************************************
File "__main__", line 18, in __main__.factorial
Failed example:
    [factorial(long(n)) for n in range(6)]
Exception raised:
    Traceback (most recent call last):
      File "D:\anaconda3\Lib\doctest.py", line 1351, in __run
        exec(compile(example.source, filename, "single",
      File "<doctest __main__.factorial[1]>", line 1, in <module>
        [factorial(long(n)) for n in range(6)]
      File "<doctest __main__.factorial[1]>", line 1, in <listcomp>
        [factorial(long(n)) for n in range(6)]
                   ^^^^
    NameError: name 'long' is not defined
Trying:
    factorial(30)
Expecting:
    265252859812191058636308480000000L
**********************************************************************
```

Рисунок 14 – Результат тестирования кода с применением doctest

```
File "__main__", line 22, in __main__.factorial
Failed example:
    factorial(30L)
Exception raised:
    Traceback (most recent call last):
      File "D:\anaconda3\Lib\doctest.py", line 1351, in __run
        exec(compile(example.source, filename, "single",
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
      File "<doctest __main__.factorial[3]>", line 1
        factorial(30L)
                  ^
    SyntaxError: invalid decimal literal
Trying:
    factorial(-1)
Expecting:
    Traceback (most recent call last):
        ...
    ValueError: n must be >= 0
ok
Trying:
    factorial(30.1)
Expecting:
    Traceback (most recent call last):
        ...
    ValueError: n must be exact integer
ok
Trying:
    factorial(30.0)
Expecting:
    265252859812191058636308480000000L
**********************************************************************
```

```
File "__main__", line 34, in __main__.factorial
Failed example:
    factorial(30.0)
Expected:
    265252859812191058636308480000000L
Got:
    265252859812191058636308480000000
Trying:
    factorial(1e100)
Expecting:
    Traceback (most recent call last):
        ...
    OverflowError: n too large
ok
1 items passed all tests:
   1 tests in __main__
**********************************************************************
1 items had failures:
   4 of   8 in __main__.factorial
9 tests in 2 items.
5 passed and 4 failed.
***Test Failed*** 4 failures.
```

Продолжение рисунка 14

На рисунке 15 показан код модуля CAT с применением unittest.

```
#Тестирование с помощью unittest
def add(a, b):
    """Возвращает сумму a и b.
     >>> add(3, 4)
    7
    """
    sum = a + b
    #import pdb; pdb.set_trace() # для входа в дебагер
    return sum
```

```
import unittest
class TestAdd(unittest.TestCase):
    def test_add(self):
        self.assertEqual(add(2,3), 5)
```

```
def mult(a, b):
    """Возвращает произведение a и b..
     >>> mult(3, 4)
    12
    """
    return a*b
```

```
class TestMult(unittest.TestCase):
    def test_mult(self):
        self.assertEqual(mult(2,3), 6)
```

```
unittest.main(argv=[''], exit=False)
```

Рисунок 15 – Код модуля CAT с применением unittest

На рисунке 16 показан результат тестирования с применением unittest.

```
..
----------------------------------------------------------------------
Ran 2 tests in 0.002s

OK
<unittest.main.TestProgram at 0x20b2f344d10>
```

Рисунок 16 – Результат тестирования с применением unittest

На рисунке 17 показан код модуля CAT с применением pytest.

```python
#Тестирование с помощью pytest
import pytest
import ipytest
# Загружаем расширение pytest
%load_ext pytest
```

```
The pytest module is not an IPython extension.
```

```python
# Определяем тестируемую функцию
def add(a, b):
    return a + b
```

```python
# Пишем тест в отдельной ячейке с магической командой
#%%pytest

def test_add():
    assert add(2, 3) == 6
    assert add(-1, 1) == 0
    assert add(0, 0) == 0
```

```python
ipytest.run()
```

Рисунок 17 – Код модуля CAT с применением pytest

На рисунке 18 показан результат тестирования с применением pytest.

```
========================================= test session starts =========================================
platform win32 -- Python 3.12.4, pytest-7.4.4, pluggy-1.0.0
rootdir: C:\Users\Сергей
plugins: anyio-4.2.0
collected 1 item

t_36e741d1204d4258842f635eb179913b.py F                                                      [100%]

============================================== FAILURES ===============================================
_____________________________________________ test_add _____________________________________________

    def test_add():
>       assert add(2, 3) == 6
E       AssertionError

======================================= short test summary info =======================================
FAILED t_36e741d1204d4258842f635eb179913b.py::test_add - AssertionError
========================================== 1 failed in 0.25s ==========================================

<ExitCode.TESTS_FAILED: 1>
```

Рисунок 18 – Результат тестирования с применением pytest

В таблице 14 представлен протокол функционального тестирования ПО САТ ИАС.

Таблица 14 – Протокол функционального тестирования ПО САТ ИАС.

Действие	Результат	Комментарий
Тестирование модуля doctest	Успешно	Отсутствует
Тестирование модуля unittest	Успешно	Отсутствует
Тестирование модуля pytest	Успешно	Отсутствует

Результаты функционального тестирования подтвердили работоспособность разработанного ПО САТ ИАС [19].

Выводы по главе 3

Doctest – это модуль в Python, который позволяет проводить тестирование кода через примеры в документации.

Unittest обеспечивает структуру, гибкость и надежность, но требует больше времени на настройку.

Pytest – это де-факто стандарт для тестирования в Python-сообществе [14].

Заключение

Бакалаврская работа посвящена актуальной проблеме разработки ПО САТ ИС.

Современные информационные системы включают множество модулей, интеграций с внешними сервисами, работают в распределённых средах (облако, гибридные решения). Ручное тестирование таких систем становится дорогим и медленным, тогда как автоматизация ускоряет проверку и повышает её точность.

Помимо снижения трудозатрат, особенно при повторяющихся сценариях тестирования, автоматизированное тестирование снижает негативное влияние «человеческого фактора» тестировщиков на результаты рутинных проверок программного обеспечения в условиях DevOps и Agile-разработки, при которых релизы выходят часто, иногда несколько раз в день.

В результате выполнения бакалаврской работы были решены следующие задачи:

- «выполнена постановка задачи на разработку ПО для анализа и прогнозирования рынка недвижимости. современные САТ обладают гибкой архитектурой, позволяющей интегрироваться в CI/CD, поддерживать различные виды тестирования и масштабироваться. Выбор инструмента зависит от технологического стека, требований к тестированию и уровня автоматизации в проекте. ИАС, разработанные на языке Python, являются узкоспециализированными. Разработка ПО САТ для таких ИАС актуальна и представляет практический интерес;
- при проектировании ПО использована методология RUP, главным преимуществом которой является интеграция с UML. Построена диаграммы вариантов использования САТ ИАС – мощный инструмент для анализа, проектирования и документирования,

который обеспечивает ясность, снижает риски и способствует созданию ПО, ориентированного на потребности пользователей. Для представления алгоритма работы ПО разработана его диаграмма деятельности, на которой выделены 2 плавательные дорожки, изображающие зоны ответственности исполнителей: Тестировщик и ПО CAT ИАС;

– реализовано и протестировано ПО CAT ИАС. «По результатам сравнения и с учетом предпочтений разработчика выбираем среду разработки Jupyter Notebook» [7]. Jupyter Notebook лучше подходит для исследовательской работы, анализа данных и интерактивного обучения. Использованы библиотеки doctest, unittest и pytest. Последний модуль де-факто является стандартом для тестирования в Python-сообществе.

«Результаты функционального тестирования подтвердили работоспособность разработанного ПО CAT ИАС» [12].

Список используемой литературы и используемых источников

1. Автоматизация тестирования на Python. Шесть способов тестировать эффективно [Электронный ресурс]. URL: https://habr.com/ru/companies/otus/articles/560884/ (дата обращения: 29.09.2025).

2. Автоматизация тестирования с Playwright и Python: настройка, примеры и лучшие практики [Электронный ресурс]. URL: https://inzhenerka.tech/blog/tpost/4jkzty1dn1-avtomatizatsiya-testirovaniya-s-playwrig (дата обращения: 29.09.2025).

3. Автоматизируем тестирование приложений: три популярных фреймворка Python для тестировщиков [Электронный ресурс]. URL: https://practicum.yandex.ru/blog/fraymvorki-dlya-testirovaniya-na-python/ (дата обращения: 29.09.2025).

4. Архитектура платформы 1С: Предприятие (версия 8.3.27): автоматизированное тестирование [Электронный ресурс]. URL: https://v8.1c.ru/platforma/avtomatizirovannoe-testirovanie/ (дата обращения: 10.04.2025).

5. Базовые принципы разработки программного обеспечения : учебное пособие / В. И. Шипков, Т. Р. Захаренкова, А. А. Нечаев, А. С. Грицай. Омск : Омский государственный технический университет, 2023.116 с. URL: https://www.iprbookshop.ru/140826.html (дата обращения: 29.09.2025).

6. Библиотека Pytest: что это и как тестировать код в Python [Электронный ресурс]. URL: https://exolve.ru/blog/pytest-how-to-test-code-python/ (дата обращения: 29.09.2025).

7. Знакомство с Jupyter Notebook [Электронный ресурс]. URL: https://portal.tpu.ru/SHARED/j/JBOLOTOVA/academic/Tab/instruction_JN.pdf (дата обращения: 29.09.2025).

8. Использование диаграммы вариантов использования UML при проектировании программного обеспечения [Электронный ресурс]. URL: https://habr.com/ru/articles/566218/ (дата обращения: 10.04.2025).

9. Как использовать Python для автоматизации тестирования [Электронный ресурс]. URL: https://sky.pro/media/kak-ispolzovat-python-dlya-avtomatizaczii-testirovaniya/ (дата обращения: 29.09.2025).

10. Логическое моделирование [Электронный ресурс]. URL: https://intuit.ru/studies/courses/3440/682/lecture/14036 (дата обращения: 10.04.2025).

11. Основы PyCharm [Электронный ресурс]. URL: https://habr.com/ru/articles/720480/ (дата обращения: 29.09.2025).

12. Пишем АПИ автотесты на Python по шагам [Электронный ресурс]. URL: https://habr.com/ru/articles/765512/ (дата обращения: 29.09.2025).

13. Подходы к управлению требованиями в IBM OpenUP и FURPS+ [Электронный ресурс]. URL: https://business-analytics-russia.ru/requirements-in-ibm-openup-furps/ (дата обращения: 10.04.2025).

14. Руководство по Pytest: как тестировать код в Python [Электронный ресурс]. URL: https://skillbox.ru/media/code/rukovodstvo-po-pytest-kak-testirovat-kod-v-python/ (дата обращения: 10.04.2025).

15. Сайт ТГУ [Электронный ресурс]. URL: https://www.tltsu.ru (дата обращения: 10.04.2025).

16. Средства автоматизированного тестирования [Электронный ресурс]. URL: https://ibs-qa.ru/media/1952-sredstva-avtomatizirovannogo-testirovaniya/ (дата обращения: 10.04.2025).

17. Сузи Р. А. Язык программирования Python : учебное пособие / Р. А. Сузи. Москва: Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2024. 350 с. URL: https://www.iprbookshop.ru/142310.html.

18. Тесты в Python: все основные подходы, плюсы и минусы [Электронный ресурс]. URL:

https://habr.com/ru/companies/yandex/articles/517266/ (дата обращения: 29.09.2025).

19. Функциональное тестирование ПО: задачи, виды, методы проведения [Электронный ресурс]. URL: https://practicum.yandex.ru/blog/funkcionalnoe-testirovanie-po/ (дата обращения: 29.09.2025).

20. Функциональные характеристики системы автоматизации тестирования [Электронный ресурс]. URL: https://ibs-qa.ru/files/ckeditor/ (дата обращения: 10.04.2025).

21. Advanced usage of Selenium ide for web automated testing [Электронный ресурс]. URL: https://iwconnect.com/advanced-usage-of-selenium-ide-for-web-automated-testing/ (дата обращения: 10.04.2025).

22. Chatterjee A. What is Automation Testing? [Электронный ресурс]. URL: https://testrigor.com/blog/what-is-automation-testing/ (дата обращения: 10.04.2025).

23. How to use unittest-based tests with pytest [Электронный ресурс]. URL: https://docs.pytest.org/en/stable/how-to/unittest.html (дата обращения: 10.04.2025).

24. Jupyter Notebook [Электронный ресурс]. URL: https://jupyter-notebook.readthedocs.io/en/stable/ (дата обращения: 10.04.2025).

25. Mastering Python Testing: unittest, pytest, and doctest [Электронный ресурс]. URL: https://mysteryweevil.medium.com/mastering-python-testing-unittest-pytest-and-doctest-788660e8d31d (дата обращения: 10.04.2025).

26. PyCharm [Электронный ресурс]. URL: https://www.jetbrains.com/pycharm/ (дата обращения: 10.04.2025).

27. RUP Definition, Methodology & Examples [Электронный ресурс]. URL: https://study.com/academy/lesson/what-is-the-rational-unified-process-methodology-tools-examples.html (дата обращения: 10.04.2025).

28. Scientific-python/pytest-doctestplus [Электронный ресурс]. URL: https://github.com/scientific-python/pytest-doctestplus (дата обращения: 10.04.2025).

29. Scraping iframes Using Playwright for Python [Электронный ресурс]. URL: https://medium.com/@bytethewriter/playwright-for-python-and-iframes-f6e1dc4440a3 (дата обращения: 10.04.2025).

30. Selenium Testing [Электронный ресурс]. URL: https://www.browserstack.com/selenium (дата обращения: 10.04.2025).

31. Shrivastava A. Automation Framework Architecture for Enterprise Products: Design and Development Strategy [Электронный ресурс]. URL: https://www.oracle.com/technical-resources/articles/enterprise-architecture/shrivastava-automated-frameworks.html (дата обращения: 10.04.2025).

32. Visual Paradigm [Электронный ресурс]. URL: https://online.visual-paradigm.com/ (дата обращения: 29.09.2025).