МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»
Кафедра математической кибернетики и компьютерных наук

РАЗРАБОТКА СИСТЕМЫ ПРЕОБРАЗОВАНИЯ ТЕСТ-КЕЙСОВ В СКРИПТЫ АВТОМАТИЗИРОВАННОГО ТЕСТИРОВАНИЯ НА ALT LINUX

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Иванова Александра Владимировича

Научный руководитель
к. ф.-м. н., доцент _____ А. С. Иванова

Заведующий кафедрой
к. ф.-м. н., доцент _____ С. В. Миронов

Саратов 2026

ВВЕДЕНИЕ

В условиях современной разработки программного обеспечения с непрерывными циклами выпуска обновлений обеспечение качества становится критически важным и сложным процессом. Традиционные методы ручного тестирования не успевают за темпами разработки, создавая узкие места в процессах выпуска программных продуктов.

Особую сложность представляет тестирование комплексных программных систем, состоящих из множества компонентов и требующих проверки на различных конфигурациях и окружениях. Ручной подход требует значительных временных затрат, а также высокой концентрации внимания от тестировщиков, что повышает риск пропуска ошибок, особенно при повторном выполнении одних и тех же тестовых сценариев после каждого обновления системы. В результате тестирование превращается в узкое место, ограничивающее скорость доставки новых функций и исправлений в продукт и увеличивающее общие трудозатраты команд разработки.

Автоматизация тестов позволяет существенно сократить время, затрачиваемое на тестирование, поскольку автоматически запускаемые сценарии могут выполняться в кратчайшие сроки, независимо от времени суток и участия человека.

Цель: разработать систему автоматической трансформации формализованных тест-кейсов в исполняемые скрипты для систем автоматизированного тестирования, позволяющей сократить время тестирования.

Задачи:

1. провести анализ существующих систем управления тестированием и автоматизированного тестирования;
2. исследовать архитектуру современных систем автоматизированного тестирования и определить требования к генерируемым скриптам;
3. разработать формализованное описание тест-кейсов, пригодное для автоматической трансформации в исполняемые сценарии;
4. спроектировать систему преобразования тест-кейсов в скрипты автоматизированного тестирования;
5. реализовать программный комплекс для автоматической генерации тестовых сценариев;
6. протестировать разработанную систему и оценить её эффективность.

Описание разработки системы

Архитектура системы представлена на рисунке 1.

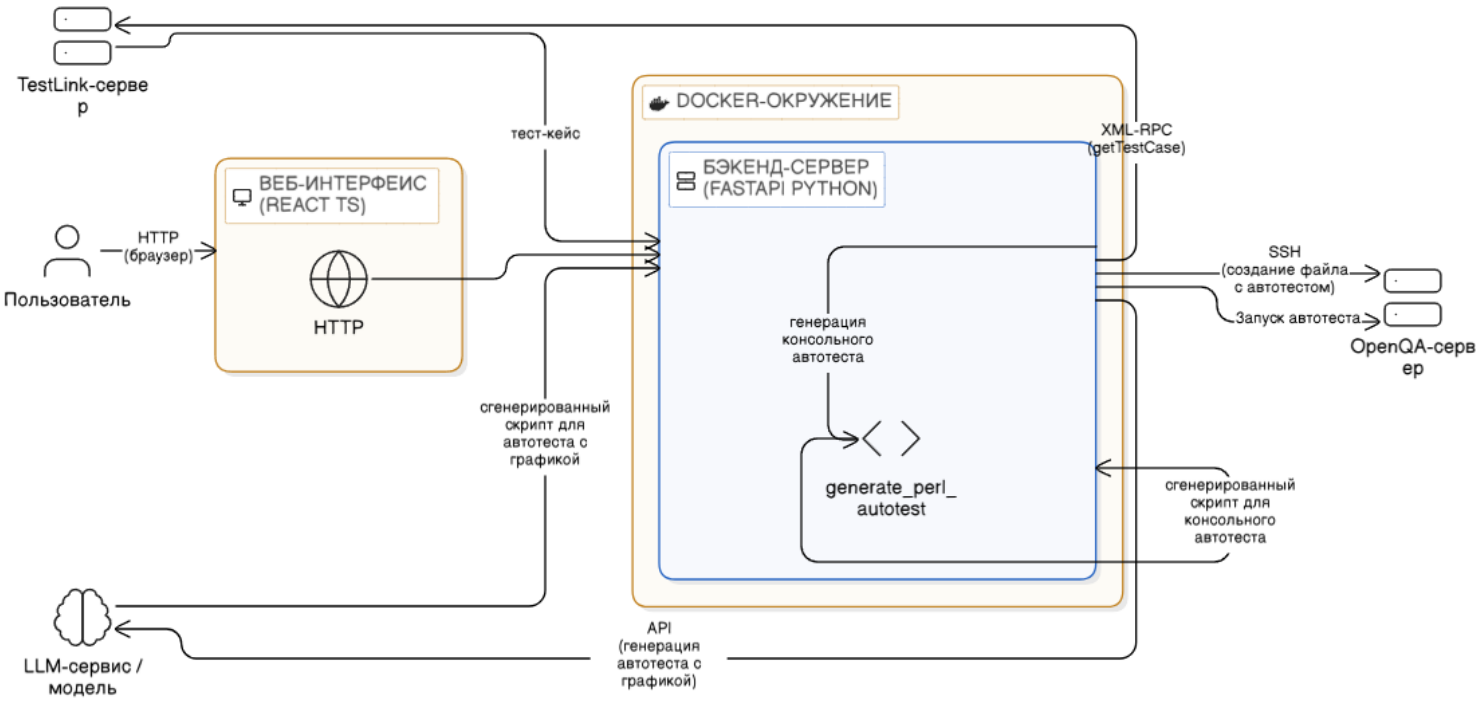

Рисунок 1 – Архитектура системы

Архитектура системы построена по клиент-серверному принципу и состоит из трех основных логических компонентов: клиентской части (Frontend), серверной части (Backend) и внешнего сервера автоматизированного тестирования (OpenQA-сервер).

Пользователь при взаимодействии с интерфейсом может ввести номер нужного тест-кейса и нажать кнопку `Запустить автотест` или `Запустить с LLM`, после чего веб-интерфейс отправляет запрос на серверную част приложения, затем происходит обращение к API TestLink для получения информации о тест-кейсе (шаги, ожидаемые результаты, предусловия, дата создания, дата изменения и т.д.). Затем, если пользователь запросил консольный автотест, произойдёт вызов функции `generate_perl_autotest`, в которой реализован алгоритм генерации консольных автотестов, если же пользователю нужно сгенерировать графический автотест, то backend-сервер будет обращаться к LLM-модели. После генерации скрипта автотеста происходит подключение к OpenQA-серве-

ру по протоклу SSH, на этом сервере создается файл с полученным скриптом, и с этого же сервера запускается выполнение автотеста.

В связи с тем, что тест-кейсы регулярно обновляются и изменяются, принято решение не использовать базу данных для того, чтобы хранить их локально. При использовании базы данных сервер все равно должен каждый раз обращаться к API TestLink, чтобы проверить его на актуальность, поэтому с целью экономии ресурсов решено не использовать базу данных для хранения тест-кейсов.

Клиентская часть представляет собой одностраничное веб-приложение, разработанное с использованием фреймворка React и языка TypeScript. Она обеспечивает взаимодействие пользователя с системой, позволяя ввести номер тест-кейса, выбрать целевую ветку дистрибутива ALT Linux (`p10` или `p11`) и образ операционной системы (`workstation`, `kworkstation`, `education-kde`, `education-xfce`), а также инициировать процесс генерации и запуска автотеста.

Серверная часть реализована на языке программирования Python с использованием высокопроизводительного асинхронного фреймворка FastAPI. Выбор Python обусловлен его богатой экосистемой библиотек для работы с API, сетевыми протоколами и обработки данных. FastAPI обеспечивает автоматическую валидацию входных данных, генерацию документации и высокую скорость обработки запросов. Серверная часть выполняет роль оркестратора: она получает запрос от клиента, взаимодействует с системой управления тестированием TestLink для получения данных тест-кейса, принимает решение о необходимости генерации нового автотеста или использования существующего, запускает процесс генерации кода и управляет развертыванием скрипта на OpenQA-сервере.

Для обеспечения изоляции и воспроизводимости среды выполнения серверная часть упакована в контейнер Docker. Это позволяет избежать конфликтов зависимостей и гарантирует идентичность поведения приложения в средах разработки и эксплуатации.

Взаимодействие с OpenQA-сервером осуществляется по защищенному сетевому протоколу SSH с использованием библиотеки Paramiko. SSH-ключи генерируются на уровне инфраструктуры и монтируются в Docker-контейнер бэкенда, что обеспечивает безопасную аутентификацию без использования паролей.

Первым этапом работы системы при поступлении запроса является импорт данных тест-кейса из платформы TestLink. Для этого используется специализированная Python-библиотека `testlink`, взаимодействующая с официальным XML-RPC API TestLink. Из всего массива данных тест-кейса система извлекает только необходимые для генерации поля: название тест-сьюта (используется для формирования пути к директории на OpenQA-сервере), предусловия , шаги выполнения и дату последнего изменения тест-кейса. Данные возвращаются в формате JSON. Объект, который возвращает реализованная функция для импорта тест-кейса:

```
    {
        "testcase_number": testcase_number,
        "test_suite_name": testcase_data[test_suite_name],
        "steps": testcase_data['steps'],
        "preconditions": testcase_data['preconditions'],
        "update_date": testcase_data['update_date']
    }
```

Важной архитектурной особенностью является отказ от использования локальной базы данных для кэширования тест-кейсов. Поскольку тест-кейсы регулярно обновляются, хранение их локальной копии потребовало бы постоянной синхронизации и проверки актуальности, что нивелировало бы выигрыш в производительности. Вместо этого система каждый раз обращается к API TestLink, получая наиболее актуальные данные.

После импорта данных тест-кейса система выполняет проверку наличия и актуальности автотеста на OpenQA-сервере. Алгоритм проверки выглядит следующим образом: При запуске автотеста для какого-либо тест-кейса он может быть уже сгенерирован, поэтому нужно рассмотреть все возможные случаи:

- если автотеста для тест-кейса нет на сервере, то он генерируется;
- если автотест для тест-кейса уже есть на сервере, но тест-кейс был обновлен, то автотест для него генерируется заново;
- если автотест для тест-кейса уже есть на сервере, и тест-кейс актуален, то перегенерация автотеста зависит от выбора пользователя.

Дата последнего обновления тест-кейса извлекается при его импорте из TestLink, а дата последнего изменения автотеста на OpenQA-сервере берется путем подключения к серверу по ssh с помощью библиотеки Paramiko.

Генерация автотестов — основная часть разработываемой системы, в которой на основе тест-кейса генерируется автотест на языке Perl. Автотесты представляют собой программный код, написанный на языке программирования Perl. На этом языке Perl реализован модуль `basetest`, который содержит множество полезных функций для выполнения команд автотестов. Некоторые из основых команд данного модуля:

- `assert_screen()` — проверяет, что текущее состояние экрана (или части экрана) совпадает с ожидаемым эталоном;
- `assert_and_click()` — проверяет, что текущее состояние экрана (или части экрана) совпадает с ожидаемым эталоном, и делает клик мышью по выбранной области на экране;
- `serial_terminal()` — функция для запуска терминала;
- `enter_cmd()` — отправляет команду в терминал / консоль / интерфейс устройств и имитирует нажатие Enter;
- `select_console()` — функция для переключения режима терминала, либо режим пользователя root, либо режим обычного пользователя;
- `assert_script_run()` — выполнение команды в терминале, функция завершается успехом, если команда, переданная в неё в качестве аргумента возвращает код ошибки 0 после выполнения.

Автотесты генерируются двумя способами: алгоритмическим и с использованием LLM.

С помощью алгоритма генерируются консольные автотесты, т.е. автотесты, в которых все действия выполняются в linux-терминале. Все linux-команды выполняются либо от обычного пользователя, либо от администратора (root). Алгоритм заключается в следующем:

1. для каждого шага тест-кейса сначала извлекается команда, все команды выделены жирным шрифтом и начинаются с символа ’#’ или ’$’;
2. если первый символ команды ’#’, то команда выполняется от пользователя root, а если же первый символ ’$’, то от обычного пользователя;
3. если команда выполняется от пользователя root, а предыдущая команды выполнялась от обычного пользователя, нужно вызвать `select_console()` с аргументом `'root-console'`, если же команда выполняется от обычного пользователя, а предыдущая от пользователя root, то `select_console()` вызывается с аргументом `'user-console'`, если команда выполняется от

того же пользователя, что и предыдущая, то переключать режим терминала не нужно, если же текущая команда первая в тест-кейсе, то аргумент функции `select_console()` зависит от того, какой пользователь выполняет данную команду;

4. для выполнения команды вызывается функция `assert_script_run()`, и аргументом в нее передается сама команда без первого символа (’#’ или ’$’);
5. шаги 1–4 выполняются для каждого шага тест-кейса, в том числе и для предусловий, которые выполняются перед основными шагами.

Для тест-кейсов, требующих взаимодействия с графическим интерфейсом пользователя, алгоритмический подход неприменим из-за высокой вариативности действий (клики мышью, проверка визуальных элементов, ввод текста). Для решения этой задачи в систему интегрирована большая языковая модель (LLM).

Изначально рассматривалась возможность использования модели `Qwen 3-32B-Coder-Instruct`, однако в ходе практической реализации было выявлено, что для дообучения такой объёмной модели не хватает мощности аппаратных ресурсов.

Для преодоления этого ограничения была выбрана более компактная модель `Omnicoder-2-9b` (9 миллиардов параметров), которая была дообучена на локальном оборудовании. Для расширения обучающей выборки сначала была использована базовая версия модели для генерации черновых вариантов автотестов по имеющимся тест-кейсам. Эти черновые варианты были вручную проверены и скорректированы, после чего добавлены в обучающий набор, что позволило увеличить объем данных и улучшить качество дообучения LoRA-адаптера.

Процесс генерации графического автотеста осуществляется путем отправки структурированного запроса в API модели через локальный сервер Ollama. Запрос жестко регламентирует формат вывода, запрещая использование Markdown-разметки, комментариев или поясняющего текста. Модель обязана выдать исключительно валидный Perl-код.

После успешной генерации Perl-кода автотеста (независимо от того, был ли он создан алгоритмически или с помощью LLM), система выполняет его развертывание и запуск на целевом OpenQA-сервере. Этот процесс полностью автоматизирован и состоит из трёх последовательных этапов:

1. подключение к OpenQA-серверу;
2. создание на OpenQA-сервере файла со сгенерированным автотестом;
3. запуск автотеста на OpenQA-сервере.

SSH-ключи генерируются на уровне инфраструктуры и монтируются в Docker-контейнер с бэкенд-сервером, из которого далее осуществляется подключение по SSH к OpenQA-серверу. Для использования протокола SSH используется python-библиотека `paramiko`.

Реализованная функция для подключения к OpqnQA-серверу:

```
    def _get_ssh_client(self):
        import paramiko
        ssh = paramiko.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh.connect(self.ssh_host, username=self.ssh_user,
        key_filename=self.ssh_key)
        return ssh
```

Затем нужно создать файл с автоетестом в нужной директории `/var/lib/openqa/tests/openqa-os-autoinst-distri-altlinux/tests/task/test_suite_name`, где `test_suite_name` — это название тест-сьюта, которому принадлежит тест-кейс, которое, в свою очередь совпадает с названием тестируемого пакета.

Часть кода, отвечающая за создание таких файлов:

```
    deploy_cmds = [
            f"mkdir -p {test_dir}",
            f"cat > {test_dir}/main.pm << 'EOF'\n{perl_content}\nEOF"
        ]
```

Linux-команда `mkdir` отвечает за создание директории, а команда `cat` создает файл и записывает в него сгенерированный автотест.

После создания файла с автотестом на OpenQA-сервере остается только запустить его. Запуск автотеста выполняется следующей командой:

```
    openqa-cli api --host http://$(hostname -i) -X POST jobs \\
            DISTRI=alt \\
            VERSION={branch} \\
            FLAVOR={iso_}{suffix}-hdd \\
            ARCH=x86_64 \\
            BUILD=11.1 \\
```

```
7               HDD_1=fixed/alt-{iso_}-{p}-x86_64{suffix}-upd.qcow2 \\
8               MACHINE=64bit \\
9               TEST={test_suite_name} \\
10              TESTING_DIRECTION=task
```

Эта команда представляет собой HTTP-запрос к REST-API OpenQA, который создаёт новое тестовое задание. Переменная `$(hostname -i)` хранит в себе ip-адрес OpenQA-сервера. Рассмотри передаваемые параметры:

- `DISTRI=alt` — дистрибутив, для которого запускается тест;
- `VERSION=branch` — ветка, на которой нужно запускать автотест;
- `FLAVOR=iso_suffix-hdd` — конкретный образ дистрибутива;
- `ARCH=x86_64` — архитектура, для которой выполняется автотест;
- `BUILD=11.1` — версия сборки образа;
- `HDD_1=fixed/alt-iso_-p-x86_64suffix-upd.qcow2` — путь к образу виртуального диска, примонтированного на виртуальной машине с развёрнутым OpenQA-сервером;
- `MACHINE=64bit` — тип аппаратной конфигурации;
- `TEST=test_suite_name` — название тест-сьюта, тест-кейс которого автоматизируется;
- `TESTING_DIRECTION=task` — номер задания, с которым пришёл на тестирование новый или обновлённый пакет.

Клиентская часть системы реализована как одностраничное приложение на стеке React и TypeScript. Использование TypeScript обеспечивает статическую типизацию пропсов, состояний и ответов от API, что минимизирует количество ошибок времени выполнения и упрощает поддержку кода. Ознакомиться с интерфейсом системы можно на рисунке 2.

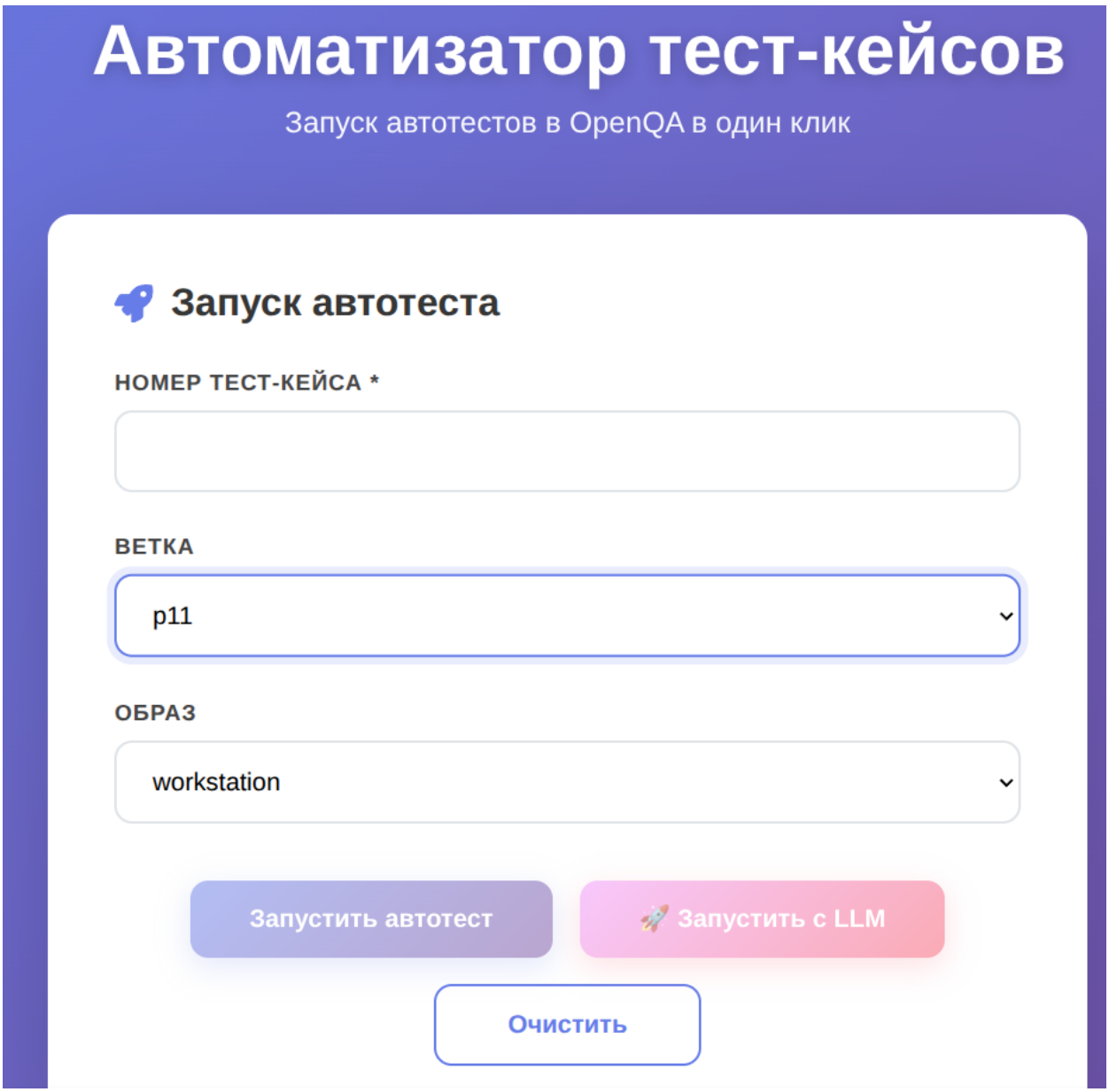

Рисунок 2 – Интерфейс системы

Интерфейс спроектирован максимально минималистичным и интуитивно понятным. Основная рабочая область содержит:

— поле ввода для номера тест-кейса;
— выпадающий список для выбора ветки дистрибутива ALT Linux;
— выпадающий список для выбора конкретного образа операционной системы;
— две основные кнопки действия: `Запустить автотест` — инициирует алгоритмическую генерацию консольного теста и `Запустить с LLM` — инициирует генерацию графического теста через языковую модель;
— кнопка `Очистить` для сброса формы.

Логика взаимодействия с пользователем включает обработку различных

сценариев состояния системы. При нажатии на кнопку запуска отправляется асинхронный запрос на бэкенд. В зависимости от ответа модуля проверки актуальности, интерфейс отображает соответствующие информационные сообщения или модальные окна:

— если автотест новый, отображается сообщение об успешной генерации и запуске;
— если автотест уже существует и тест-кейс не изменялся, появляется диалоговое окно с вопросом о перегенрации автотеста. Это дает пользователю контроль над процессом и позволяет сэкономить вычислительные ресурсы LLM при отсутствии необходимости в обновлении кода;
— при вводе несуществующего номера тест-кейса система перехватывает ошибку от API TestLink и выводит понятное сообщение об ошибке в интерфейсе, не допуская падения приложения.

После успешного запуска задания интерфейс предоставляет пользователю кнопку `Открыть OpenQA Job`, которая перенаправляет его на веб-страницу сервера OpenQA, где можно наблюдать за выполнением теста в реальном времени, просматривать логи, анализировать пошаговые скриншоты и изучать причины возможных сбоев.

ЗАКЛЮЧЕНИЕ

В ходе данной работы была разработана и протестирована система автоматической генерации автотестов на основе тест-кейсов, которая интегрируется с существующими инструментами тестирования. Решены задачи по анализу современных подходов к автоматизации тестирования, выбору и обоснованию архитектуры программной системы, реализации модуля взаимодействия с LLM-моделью Llama, а также построению логики обработки шагов тест-кейсов с учётом прав пользователя, состояния терминала и необходимости перегенерации автотестов.

Разработанная система позволяет автоматически генерировать автотесты как консольного, так и графического типа, в зависимости от типа тест-кейса, при этом выбор между перегенерацией и использованием существующего автотеста осуществляется с учётом актуальности тест-кейса и решения пользователя.

Для тетирования системы было отобрано 150 консольных и 150 графических тест-кейсов. Все консольные автотесты были сгенерированы идеально, а из 150 графических тест-кейсов безошибочно сгенерировалось 42 автотеста, 62 требуют небольшой доработки, 46 автотестов требуют глубокой оптимизации.

Полученные результаты подтверждают достижение поставленной цели и решённость всех задач, сформулированных во введении, а также показывают практическую применимость разработанной системы. В дальнейшем планируется улучшить систему с помощью дообучения более крупной модели на мощном оборудовании и интеграции с системой, позволяющей создавать и сохранять эталонные скриншоты на основании текущего состояния экрана, которая на данный момент находится в разработке.