Программные системы и вычислительные методы
Правильная ссылка на статью:
Макаров К.С., Фаткин Р.И. Скриншотное тестирование как многоаспектный вид автоматизированной динамической верификации веб-приложений // Программные системы и вычислительные методы. 2025. № 1. DOI: 10.7256/2454-0714.2025.1.73535 EDN: UVGEBC URL: https://nbpublish.com/library_read_article.php?id=73535

Скриншотное тестирование как многоаспектный вид автоматизированной динамической верификации веб-приложений

Макаров Константин Сергеевич

кандидат технических наук

зав. кафедрой; кафедра программного обеспечения и администрирования информационных систем; Курский государственный университет

305000, Россия, Курская обл., г. Курск, Центральный округ, ул. Радищева, зд 33

makarov_ks@kursksu.ru

Фаткин Руслан Игоревич

ORCID: 0009-0006-3481-6122

аспирант; кафедра программного обеспечения и администрирования информационных систем; Курский государственный университет

305014, Россия, Курская обл., г. Курск, Центральный округ, ул. Рябиновая, д. 26Б, кв. 96

ruslan4631@yandex.ru

Статья из рубрики "Показатели качества и повышение надежности программных систем"

DOI:

10.7256/2454-0714.2025.1.73535

EDN:

UVGEBC

Дата направления статьи в редакцию:

02-03-2025

Аннотация: Предметом исследования является многоаспектное скриншотное тестирование как современный метод автоматизированной динамической верификации веб-приложений, объединяющий функциональное тестирование и проверку пользовательского интерфейса (UI). Современные методы тестирования сталкиваются с проблемами высокой трудоемкости, ложноположительных срабатываний и низкой

масштабируемости, особенно в сложных проектах. Основная цель исследования — создать и внедрить метод, который позволяет повысить точность выявления дефектов, сократить время тестирования и снизить затраты на разработку тест-кейсов. В работе рассматриваются алгоритмы сравнения изображений, методы фильтрации динамических элементов и подходы к автоматизированному анализу интерфейсов для повышения эффективности и стандартизации в процессе верификации веб-приложений. В отличие от функционального и UI-тестирования по отдельности, предложенный метод позволяет анализировать несколько аспектов интерфейса и функциональности одновременно, что минимизирует трудозатраты и повышает надежность тестирования. Используется автоматическое сравнение эталонных и тестовых скриншотов на уровне пикселей, структурных элементов и содержимого с применением Python, Selenium, PIL и Pytest-xdist для параллельного выполнения тестов, что позволяет эффективно решать задачи верификации веб-приложений. Некоторые исследователи в области тестирования сходятся во мнении, что сам процесс тестирования мало стандартизирован и не имеет четких критериев оценки его эффективности. Предлагаемый нами метод позволяет добиваться решения поставленных задач верификации даже в условиях изменяющихся стратегий и подходов к оценке работоспособности системы за счет создания гибкой и точной системы проверки, которая объединяет различные типы тестирования в единую структуру, что делает данный метод подходящим для современных задач разработки программного обеспечения. Экспериментальная часть демонстрирует преимущества многоаспектного скриншотного тестирования по сравнению с другими методами, включая сокращение времени тестирования, повышение точности обнаружения дефектов и улучшение анализа получаемых отчетов. Данный подход может быть адаптирован к различным сценариям тестирования и выгоден для использования в высоконагруженных проектах, требующих регулярной регрессионной проверки.

Ключевые слова:

скриншотное тестирование, многоаспектное скриншотное тестирование, автоматизированное тестирование, динамическая верификация Веб-приложения, Python в тестировании, Selenium и PIL, UI тестирование, оптимизация тестирования, критерии тестирования, задачи верификации

Введение

В процессе разработки любого программного обеспечения (ПО) важным этапом является тестирование создаваемых решений. Современные приложения, особенно веб и мобильные, требуют регулярного обновления и доработки в условиях быстро меняющихся требований как со стороны рынка, так и со стороны внедрения новых технологий. Для обеспечения высокого уровня качества и поддержания стабильного функционирования важно регулярно проводить анализ состояния системы т. к. даже малозначительные ошибки могут привести к потере стабильности, ухудшению пользовательского опыта и т. д., а как следствие к экономическим издержкам. Так, ежегодные экономические потери из-за некачественного ПО только в США оцениваются в десятки миллиардов долларов [1, с. 14], что подчеркивает актуальность разработки надежных методов тестирования программного обеспечения.

Веб-приложения стали неотъемлемой частью жизни для миллиардов людей по всему миру. И большинство таких решений имеют коммерческую направленность. Для продукта, целью которого в конечном счете является получение прибыли, экономические потери,

связанные с некачественным ПО, могут стать одним из главных источников проблем т.к. любые ошибки, приводящие к ухудшению опыта использования, ведут к потерям в количестве реальных пользователей. Именно тестирование выпускаемого продукта призвано решить ряд таких проблем.

Скриншотное тестирование, ориентированное на автоматическое сравнение интерфейсов, активно применяется в динамической верификации ПО. Однако традиционные подходы, такие как функциональное или отдельное UI-тестирование, имеют ограничения: высокие трудозатраты, избыточное количество ложных срабатываний и низкую производительность при масштабировании задач. В случае с высоконагруженными и сложными веб-приложениями отслеживать все эти процессы в ручном режиме достаточно трудоемкая задача, которая чревата ошибками со стороны тестировщика.

Гурин Р. Е., Рудаков И. В., Ребриков А. В. рассматривают существующие подходы к верификации ПО и приводят анализ их ограничений и эффективности[2]. Проблемы, выделенные в статье, указывают на необходимость новых решений, способных преодолеть ограничения традиционных подходов.

Также, в исследованиях[3-5] отмечается, что верификация программного обеспечения имеет ряд ограничений, связанных с:

1. ограничением времени на тестирование и выделенным бюджетом;
2. человеческим фактором (субъект, проводящий тестирование, напрямую влияет на его результат);
3. изменение требований к работе ПО уже в процессе разработки или на этапе поддержки продукта;
4. формулировкой логики задач верификации напрямую влияет на получаемые результаты, если задача сформулирована неполно или некорректно, результат можно признать недействительным;
5. отсутствие современных метрик для оценки эффективности тестирования.

Проблемы, выделенные в исследованиях, указывают на необходимость новых решений, способных преодолеть ограничения традиционных подходов. Многоаспектное скриншотное тестирование предлагает комбинированный подход, объединяющий функциональное и UI-тестирование, что может повысить точность и производительность верификации ПО.

Кудрявцева Е. Ю., также отмечает, что использование автоматизированного тестирования в больших проектах наиболее обосновано и релевантно[6]. Таким образом, предлагаемая методика отвечает на вызовы и способствует дальнейшему развитию области верификации программного обеспечения.

Стоит также отметить, что в отечественной научной литературе на данный момент вопросы, связанные с верификацией ПО, в частности, общедоступного назначения, рассматриваются редко или ограничиваются обзорными статьями существующих методов. Ввиду этого область тестирования ПО развивается не столь стремительно, в отличие от разработки. Однако быстрые темпы развития области разработки ПО требуют от методов верификации такой же быстрой адаптации для решения актуальных задач. Так, целью данной работы было определено следующее — разработать метод многоаспектного

скриншотного тестирования, сочетающего функциональные и UI-подходы, для улучшения точности выявления дефектов и оптимизации ресурсов в процессе верификации веб-приложений. Для достижения цели предложено использовать язык программирования Python с библиотеками Selenium и PIL, что обеспечивает автоматизацию тестов и снижение трудозатрат.

1. Методы исследования

Методология исследования основана на разработке и применении многоаспектного скриншотного тестирования, которое представляет собой комбинацию функционального и UI-тестирования, позволяющую эффективно решать задачи верификации веб-приложений. Этот метод позволяет одновременно проверять несколько аспектов работы программного обеспечения в одном запуске тестового сценария. Такой подход призван снизить количество ложных срабатываний и увеличить точность тестирования.

Основная идея метода заключается в проведении параллельного анализа функциональных и визуальных характеристик веб-приложения путем автоматического сравнения эталонных и тестовых скриншотов. Это достигается за счет:

1. Создания эталонных версий скриншотов, которые фиксируют корректное состояние визуальных и функциональных элементов.
2. Разработки тестовых сценариев, направленных на автоматизированное сравнение текущего состояния приложения с эталонным по нескольким уровням:

- **Пиксельный уровень:** проверка различий между эталонным и тестовым скриншотами на уровне отдельных пикселей.
- **Уровень структурных элементов:** анализ корректности отображения интерфейсных элементов (размеры, расположение, отступы, шрифты).
- **Уровень содержимого:** проверка соответствия текстовой и графической информации эталону.

Методология ориентирована на минимизацию ложноположительных результатов, которые могут возникать при динамических изменениях интерфейса (например, всплывающих уведомлениях или анимациях). Для этого внедрены фильтры и маскирование динамических элементов, что позволяет исключить их влияние на итоговые результаты тестирования.

Экспериментальная часть методологии направлена на проведение сравнительного анализа многоаспектного скриншотного тестирования с традиционными методами, такими как функциональное и UI тестирование. Для этого были выделены ключевые метрики оценки:

- Количество выявленных дефектов;
- Точность тестирования (учитывая ложные срабатывания и пропущенные ошибки);
- Время выполнения тестов;
- Трудозатраты на разработку и поддержку тестов;
- Надежность и стабильность тестов.

Методология была протестирована на веб-приложении, разработанном в экспериментальных целях, и охватывала такие аспекты, как:

- Проверка отображения форм (регистрация, авторизация);
- Проверка работы элементов интерфейса (кнопки, поля ввода);

- Адаптивное отображение интерфейса в различных конфигурациях (тёмная/светлая темы, разрешение экрана).

Используемая методология обеспечивает более глубокий анализ взаимодействия элементов интерфейса, что позволяет выявлять дефекты, которые могли быть пропущены при применении других видов динамической верификации.

Так, многоаспектное скриншотное тестирование направлено на конфигурируемую проверку нескольких тестовых ситуаций для заданной верификационной задачи за один запуск автоматизированных тестовых сценариев. Каждый аспект сосредоточен на верификации определенных функциональных или визуальных элементов интерфейса и, в процессе прохождения теста, должны быть обнаружены все нарушения для проверяемых тестовых ситуаций. Тестовыми ситуациями называются те ситуации, в которых выполняется тестирование, а процедуры, описывающие процесс создания этих ситуаций и проверки, которые необходимо выполнить над полученными результатами, — тестами[7, С. 68]. В рамках верификации ПО путем многоаспектного скриншотного тестирования рассматриваются функциональные и UI характеристики объекта.

Объектом исследования в данной работе является веб-приложение, которое требует регулярной проверки для обеспечения стабильной работы и соответствия требованиям пользователей. Таким образом, был выбран интернет-магазин, разработанный на платформе WordPress, с акцентом на страницу «Мой аккаунт». Приложение является типичным представителем коммерческих веб-ресурсов, которые широко используются для предоставления онлайн-услуг. Оно содержит стандартные элементы пользовательского интерфейса, такие как формы для ввода данных, кнопки, чек-боксы, выпадающие списки и визуальные элементы (иконки, цветовые схемы). Страница авторизации и регистрации является критически важной для любого интернет-магазина, так как обеспечивает доступ к функционалу приложения и влияет на пользовательский опыт. В качестве тестовой среды использовалась имитация различных реальных условий эксплуатации: разные разрешения экрана, светлые и темные темы интерфейса, а также различные браузеры и устройства (десктопные и мобильные). Применение многоаспектного скриншотного тестирования для такого объекта позволяет эффективно продемонстрировать преимущества метода, включая выявление сложных ошибок, которые могли бы остаться незамеченными при использовании только функционального или UI-тестирования.

Таким образом, выбранный объект исследования — это универсальный пример, который позволяет оценить эффективность и применимость предложенного метода для более широкого спектра веб-приложений.

Для реализации предложенной методологии многоаспектного скриншотного тестирования использовался комплекс инструментов и программных библиотек, обеспечивающих автоматизацию процессов верификации, обработки изображений и анализа результатов. Для упрощения реализации многоаспектного подхода предложенный метод был интегрирован с языком программирования высокого уровня Python. Благодаря своей гибкости и обширной библиотеке инструментов для работы с изображениями и пользовательскими интерфейсами, язык предоставляет все необходимые средства для реализации многоаспектного подхода[8]. Таким образом, были использованы:

1. библиотеки Python, такие как Selenium для взаимодействия с веб-интерфейсами;

2. PIL для захвата и обработки скриншотов, включая сравнение эталонных изображений с текущими версиями;

3. Pytest-xdist для обеспечения параллельной обработки тестовых сценариев, позволяющие запускать тесты на нескольких конфигурациях устройств или браузеров одновременно.

4. JavaScript для маскирования динамических элементов интерфейса (например, анимаций или всплывающих окон), чтобы избежать ложных срабатываний при сравнении изображений.

Перечисленные технологии хорошо интегрируются в единую систему, поддерживают настройку под различные устройства, браузеры и темы интерфейса, а также позволяют параллельно выполнять тесты, что сокращает время на верификацию. Это позволило в рамках работы автоматизировать процесс скриншотного тестирования, который одновременно проверяет несколько аспектов ПО.

Также стоит отметить, что многоаспектное скриншотное тестирование повышает шансы на выявление потенциальных дефектов, которые могли бы остаться незамеченными при использовании других видов динамической верификации, например, функционального тестирования, где проверку проходит возможность взаимодействия с системой. Различные аспекты могут вести себя по-разному в зависимости от связи с другими элементами системы или условиями отображения. Например, при взаимодействии или наведении на кнопку ее цвет должен меняться, но функциональное тестирование призвано проверить только то, что с кнопкой можно взаимодействовать. Именно одновременная проверка позволяет выявлять и анализировать сложные взаимосвязи и взаимодействия между разными аспектами ПО, ошибки в которых могли бы остаться незамеченными при поэтапном тестировании отдельных аспектов. Например, при многоаспектном скриншотном тестировании веб-приложений можно одновременно проверять расположение кнопок, заполняемость инпутов, цветовые схемы и состояние различных элементов интерфейса, что делает тестирование более полным и менее подверженным ошибкам из-за невнимательности или кроссплатформенности.

2. Описание предлагаемого метода

Задача верификации была сформулирована так, что все функциональные и нефункциональные тесты были объединены в один поток, с помощью которого проведена проверка наибольшего числа аспектов ПО за наименьшее время прохождения тест-кейсов. Так, многоаспектное скриншотное тестирование даст возможность иметь сотни декомпозированных тест-кейсов в одном сценарии, что позволяет добиться хороших результатов производительности и экономической выгоды.

Каждый тест-кейс имеет свою задачу верификации, которая определяет корректное поведение. Так, путем сравнения скриншотов эталонной версии ПО с текущей версией, система анализирует интерфейс на предмет совпадения/несовпадения. В случае нахождения несовпадения тест-кейс отдает false и показывает метку ошибки в точке расхождения скриншота с ожидаемым поведением ПО. Все это фиксируется с помощью алгоритма вывода трасс-ошибок.

Трасс-ошибки (или trace-ошибки) — это последовательность шагов или операций, выполненных системой или программой, которые привели к возникновению ошибки. Они помогают локализовать сбои, что упрощает процесс диагностики и исправления ошибки.

Результат задачи верификации может иметь одно из следующих состояний:

- **Успешно (Passed):** функциональные и UI аспекты соответствуют ожидаемым результатам на всех скриншотах, что подтверждает корректность работы в рамках поставленной задачи.
- **Ошибка (Failed):** произошло несоответствие, и была зафиксирована ошибка. Трасс-ошибки позволяют определить, какие шаги привели к сбою, помогая быстрее устранить проблему.

Основные этапы процесса разработки и внедрения многоаспектного скриншотного тестирования можно представить в виде списка:

- **Подготовка тестовой среды**

На этом этапе необходимо определить, какие аспекты ПО будут тестироваться. В данном случае: переходы; заполнение форм; регистрация и авторизация; разрешение экрана; ориентация; темы (тёмная/светлая); браузеры или платформы (мобильные и десктоп устройства).

Также важно сформулировать задачу динамической верификации перед созданием тестовых сценариев. Например, проверка правильности отображения визуальных элементов, текста, иконок и т.д и возможности взаимодействия с ними. После этого стоит развернуть тестовую среду, которая будет максимально приближена к реальным условиям использования. Это включает настройку симуляторов устройств или браузеров, использование различных конфигураций экранов, загрузку необходимых шрифтов и библиотек.

Для каждого из аспектов, задействованного в скриншотном тестировании, можно использовать разные уровни проверки. Например, можно выполнить более глубокую проверку только тех элементов, которые уже были идентифицированы как потенциально проблемные на начальном этапе анализа. Это позволяет уменьшить избыточные затраты ресурсов.

- **Создание эталонных скриншотов**

Под эталонными скриншотами мы понимаем те изображения, которые отражают желаемое или правильное состояние интерфейса. Именно они будут служить основой для последующего сравнения. Скриншоты делаются на основе той версии программы, которую мы принимаем за эталонную (она либо ранее была протестирована и отлажена, либо используется дизайн-макет системы). При создании скриншотов предлагается опираться на те технические требования к средам и устройствам, которые были определены на предыдущем шаге. В рамках задачи верификации при подготовке тест-кейсов можно выделить несколько конкретных аспектов ПО, которые будут требовать проверки по различным параметрам. В рамках многоаспектного скриншотного тестирования параметры можно разделить следующим образом:

- Глобальные отклонения – это значительные изменения в ожидаемом поведении ПО (например, исчезновение элементов, серверные ошибки).

- Детальные отклонения – небольшие изменения, такие как изменение цвета или незначительное смещение элементов.

- Функциональные отклонения – ошибки в работе элементов, с которыми можно взаимодействовать.

- **Автоматизированное снятие тестовых скриншотов**

После сбора технических требований и подготовки эталонных версий скриншотов, запускаются автоматизированные тестовые сценарии, которые делают снимки текущего состояния ПО в тех же условиях, в которых были сделаны эталонные. Для многоаспектного скриншотного тестирования важно предусмотреть создание определенных условий для проверки. Например, проверку разных тем оформления, языков локализации, устройств с различными экранами и разрешениями. Это позволяет расширить покрытие тестирования и убедиться в корректной работе ПО в различных ситуациях.

- **Многоаспектное сравнение**

После автоматизированного снятия тестовых скриншотов на текущей версии ПО, автоматизированные тестовые сценарии начинают сравнивать два имеющихся в библиотеке скриншота на соответствия/несоответствия. Сравнение проходит на трех уровнях:

- пиксельный уровень, на котором определяются любые различия между эталонными и тестовыми скриншотами (изменение позиций, шрифтов, цветов).

- уровень проверки по структурным элементам позволяет проверить корректность отображения элементов интерфейса. Это могут быть размеры маржинов/паддингов, кнопок и инпутов, их расположение во вьюпорте.

- уровень проверки по содержимому, при котором проверяется не только соответствие размеров/цветов шрифта, но и корректность самого содержания. Помимо текстовой информации, можно также проверить, что все графические элементы интерфейса отображаются корректно и находятся на своих местах.

- **Отчетность и анализ результатов**

После завершения тестов необходимо задать параметр запуска с использованием фреймворка для формирования отчетности с описанием всех негативных и позитивных исходов. Эти отчеты могут включать скриншоты с подсвеченными проблемными участками, что упрощает анализ и понимание проблемы со стороны как тестировщика, так и разработчика.

На основе полученных отчетов команды могут принимать решения о необходимости исправления тех или иных проблем. Это может быть анализ вручную для сложных случаев или автоматическое принятие решения для очевидных отклонений.

Для сокращения ложноположительных срабатываний может потребоваться настройка условий фильтрации полученных результатов. Так, все найденные отклонения должны быть классифицированы по их критичности. Например, исчезнувшие элементы верстки или неработающая форма регистрации могут быть намного важнее, чем незначительные отклонения в оттенке кнопки или ее сдвиг на несколько пикселей. Также стоит обратить внимание на динамически изменяющиеся элементы: всплывающие окна, анимации, номера телефонов с автозаменой и т.д. В предлагаемом методе для визуальной проверки интерфейса используется библиотека PIL для обработки изображений, а Javascript помогает маскировать динамические элементы, такие как всплывающие уведомления или анимации, которые могут меняться между запусками и мешать корректному сравнению. Это обеспечивает более детализированное и точное сравнение, исключая нежелательные ошибки, которые не относятся к функциональным изменениям.

Таким образом, для минимизации ложноположительных срабатываний нужно настраивать

уровни чувствительности тестов или проработать алгоритм обхода анализа нежелательных к сравнению элементов. Это поможет сосредоточиться на критических багах и избежать отражения в отчетах незначительных проблем, которые не влияют на функционирование системы.

- **Рефакторинг и оптимизация тестов**

Если приложение изменилось в соответствии с новыми требованиями, старые эталонные скриншоты могут стать неактуальными. В таком случае их нужно обновить, чтобы будущие тесты были релевантны текущему состоянию ПО. Это важный процесс, который помогает избежать ложных срабатываний или несоответствия поставленных задач верификации. Также может потребоваться оптимизация тестовой системы. Постоянное улучшение алгоритмов сравнения скриншотов и инструментов автоматизации — это ключевой момент для повышения эффективности тестирования. Например, можно улучшить систему фильтрации результатов или внедрить более точные методы анализа. Оптимизация позволяет ускорить тестирование и сократить время на ручную проверку результатов.

Также в некоторых случаях дополнительно можно реализовать итерационный процесс верификации, при котором проверка правильности или соответствия системы, программы, либо ее отдельных компонентов выполняется многократно после каждого этапа разработки. Например, одна из команд разработки занимается новой функцией, которую планируют интегрировать в уже созданный продукт, в таком случае нет нужды ждать, пока новая функция появится в конечном продукте — можно начинать тестировать ее сразу. Это позволит выявлять ошибки или несоответствия требованиям на ранних стадиях и вносить необходимые изменения перед переходом к следующей фазе работы или интеграцией с основным проектом. В таком случае, при последующем запуске многоаспектного скриншотного тестирования сократится количество ошибок, которые можно было выявить раньше, что позволит сосредоточиться на проверке взаимодействия новых и старых элементов в системе.

3. Экспериментальная апробация многоаспектного скриншотного тестирования

В рамках работы был проведен эксперимент, целью которого была проверка следующий гипотезы: многоаспектное скриншотное тестирование является более эффективным видом верификации веб-приложений по сравнению с функциональным и UI тестированием, благодаря гибридному анализу всех аспектов ПО в рамках выбранного тестового сценария за один запуск тестов.

Цель эксперимента: доказать, что многоаспектное скриншотное тестирование является более эффективным видом верификации.

В рамках эксперимента был создан сайт интернет-магазина на WordPress для облегчения документирования и анализа полученных результатов.

Задача верификации была поставлена следующим образом: необходимо проверить работоспособность веб-страницы интернет-магазина «Мой аккаунт», которая предназначена для прохождения авторизации и регистрации пользователей. Под работоспособностью мы понимаем следующее: корректность отображения страницы согласно эталонным версиям ПО, возможность функционального взаимодействия с различными элементами, корректная передача информации (логин/пароль).

Оценка эффективности проводилась методом сравнительного анализа на примерах функционального, ui и многоаспектного скриншотного тестирования.

Критерии оценки эффективности: эффективность созданных решений оценивалась по следующим критериям:

- Количество выявленных дефектов: оценка эффективности тестирования на основе количества обнаруженных отклонений, включая графические артефакты и функциональные ошибки. **Метрика:** Общее количество обнаруженных дефектов каждым из видов тестирования. Сравнение между видами в процентном соотношении.
- Время выполнения тестов: анализ времени, затраченного на проведение тестов каждым из методов. **Метрика:** Среднее время, затраченное на выполнение каждого тестового сценария в рамках метода.
- Трудозатраты на разработку и поддержку тестов: оценка сложности разработки тестов и их поддержки на каждом этапе. **Метрика:** Количество часов, затраченных на разработку и поддержку тестов в разрезе каждого метода. Измерялось в процентном соотношении.
- Точность тестирования: сравнение точности обнаружения ошибок при каждом методе тестирования, включая ложные срабатывания и пропущенные дефекты. **Метрика**: Процент ложных срабатываний и пропущенных дефектов
- Надежность: оценка стабильности тестов, измеряемая количеством непредсказуемых отказов тестов.

В рамках верификационной задачи были определены следующие тестовые сценарии:

- для функционального тестирования — 2 тестовых ситуации (Табл. 1);
- для UI тестирования — 16 тестовых ситуаций (Табл. 2);
- для многоаспектного скриншотного тестирования — 4 тестовых ситуации (Табл. 3).

Рассмотрим их подробнее:

Таблица 1

Тестовая модель функционального тестирования

Номер кейса	**Название теста**	**Шаги**	**Ожидаемый результат**
1	Проверка формы регистрации. Регистрация пользователя	1. Перейти на страницу http://study-test.local/my-account/	Страница успешно открыта, отображается форма авторизации
		2. Выбрать в меню "Зарегистрироваться"	Отображается форма регистрации
		3. Заполнить Поля: Поле почты ФИО Пароль Подтверждение пароля Чекбоксы Кнопка регистрации	Поля успешно заполнены
		4. Нажать на кнопку Зарегистрироваться	Регистрация выполнена успешно
2	Проверка формы авторизации. Логин	1. Перейти на страницу http://study-test.local/my-account/	Страница успешно открыта, отображается

	пользователя		форма авторизации
		2. Заполнить поля Имя пользователя	Имя успешно введено
		3. Заполнить поле Пароль	Пароль введён
		4. Клик на кнопку Войти	Вход успешно произведён

Таблица 2

Тестовая модель UI тестирования

Номер кейса	**Название теста**	**Шаги**	**Ожидаемый результат**
1	Проверка формы регистрации (UI)	1. Перейти на страницу http://study-test.local/my-account/	Страница успешно открыта, отображается форма регистрации
		2. Выбрать в меню "Зарегистрироваться"	Отображается форма регистрации
		3. Проверка полей в форме регистрации	Отображаются: Поле почты ФИО Пароль Подтверждение пароля Чекбоксы Кнопка регистрации
2	Проверка формы авторизации UI	1. Перейти на страницу http://study-test.local/my-account/	Страница успешно открыта, отображается форма авторизации
		2. Проверка отображений меню для входа	Отображается Почта или логин Пароль Ссылка Забыли пароль Чекбокс Запомнить меня Вкладка Вход
3-10	Проверка формы регистрации (UI). Проверка свойств	1. Перейти на страницу http://study-test.local/my-account/	Страница успешно открыта, отображается

	полей: Поле почты Фамилия Имя Пароль Подтверждение пароля Чекбокс 1 и Чекбокс 2 Кнопка регистрации		форма регистрации
		Проверка свойств css поля почты	Свойства выбранного элемента отображаются корректно относительно заданных

Таблица 3

Тестовая модель многоаспектного скриншотного тестирования

Номер кейса	**Название теста**	**Шаги**	**Ожидаемый результат**
1	Проверка формы регистрации (UI)	1. Перейти на страницу http://study-test.local/my-account/	Страница успешно открыта, отображается форма авторизации
		2. Выбрать в меню "Зарегистрироваться"	Отображается форма регистрации
		3. Проверка полей в форме регистрации	Отображаются: Поле почты ФИО Пароль Подтверждение пароля Чекбоксы Кнопка регистрации
2.	Проверка формы регистрации (UI) - регистрация	1. Перейти на страницу http://study-test.local/my-account/	Страница успешно открыта, отображается форма авторизации
		2. Выбрать в меню "Зарегистрироваться"	Отображается форма регистрации
		3. Заполнить Поля: Поле почты ФИО	Поля успешно заполнены

		Пароль Подтверждение пароля Чекбоксы Кнопка регистрации	
		4. Нажать на кнопку Зарегистрироваться	Регистрация выполнена успешно
3.	Проверка формы авторизации UI	1. Перейти на страницу http://study-test.local/my-account/	Страница успешно открыта, отображается форма авторизации
		2. Проверка отображений меню для входа	Отображается Почта или логин Пароль Ссылка Забыли пароль Чекбокс Запомнить меня Вкладка Вход
4.	Проверка формы авторизации. Логин пользователя	1. Перейти на страницу http://study-test.local/my-account/	Страница успешно открыта, отображается форма авторизации
		2. Заполнить поля Имя пользователя	Имя успешно введено
		3. Заполнить поле Пароль	Пароль введён
		4. Клик на кнопку Войти	Вход успешно произведён

Ниже на рис. 1 представлена блок-схема работы предлагаемого метода многоаспектного скриншотного тестирования.

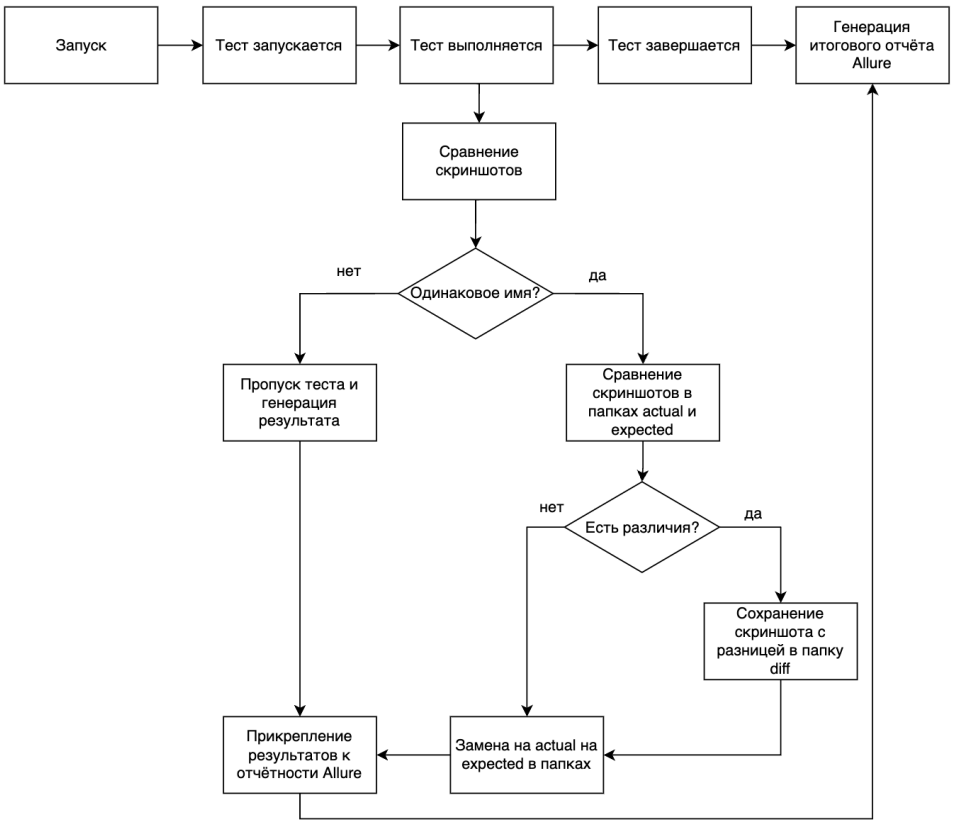

Рис. 1 — Блок схема выполнения тестов

Проверка страницы авторизации и регистрации

Шаг 1. Первый запуск тестирования проводился на эталонной версии ПО для оценки и отладки работоспособности созданного кода. В рамках этого шага все результаты должны быть pass. На рис. 2 приведены результаты тестирования эталонной версии ПО.

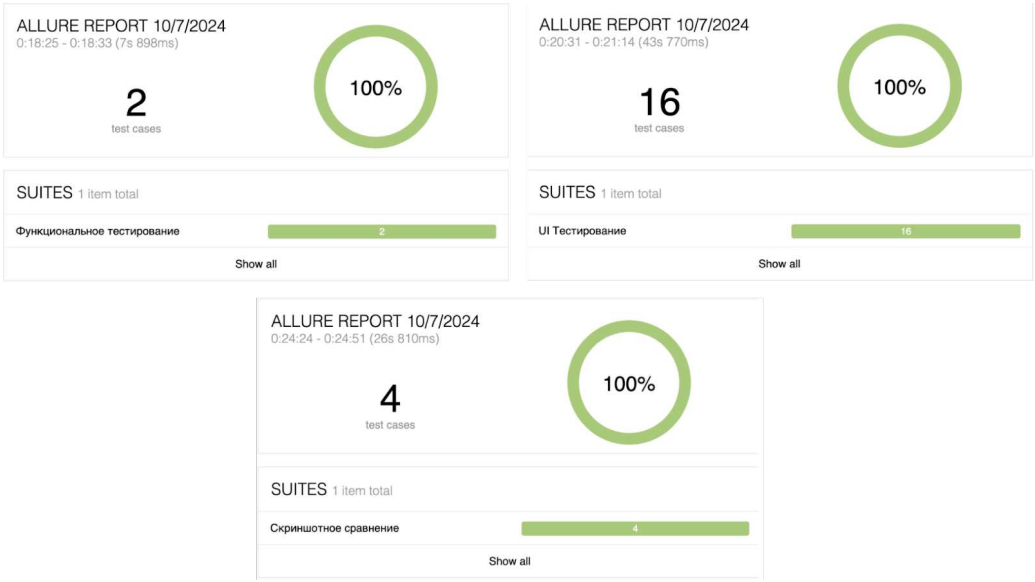

Рис. 2 — результаты функционального, UI и многоаспектного скриншотного тестирования эталонной версии ПО

Шаг 2. Для оценки результатов по выбранным метрикам необходимо внести изменения в тестируемую модель, чтобы оценить способность созданного решения выявлять ошибки. Изменения вносились в следующие аспекты: ширина полей формы авторизации изменена с 50 пикселей на 60 пикселей, белая заливка формы заменена с белой на

серую. Результаты показаны на рис. 3 и рис. 4.

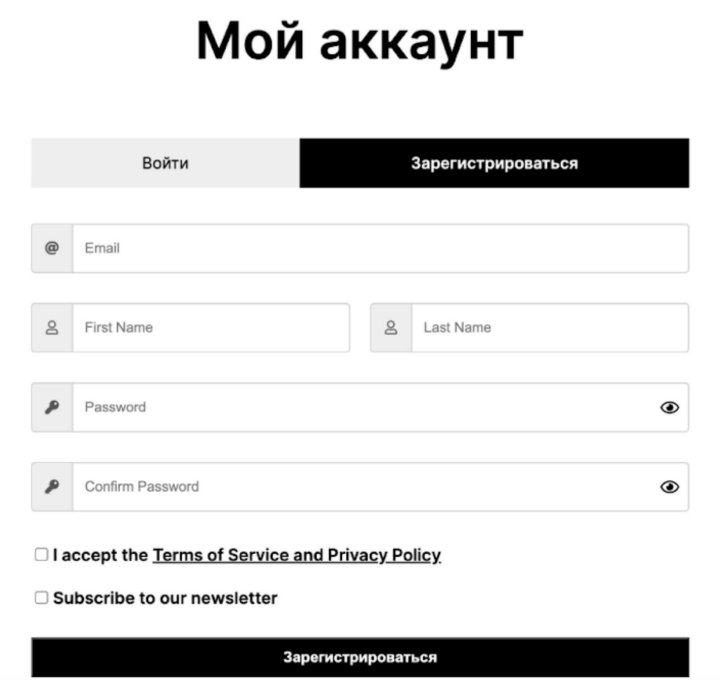

Рис. 3 — визуальная демонстрация эталонной версии ПО

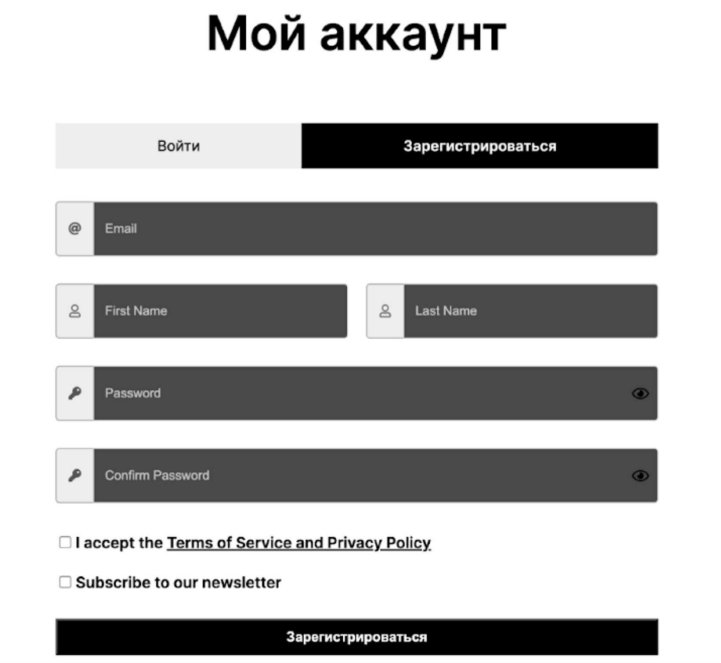

Рис. 4 — визуальная демонстрация версии ПО с имитацией ошибок

По результатам тестирования после внесения изменений (Рис. 5), имитирующих возникновение реальных ошибок на тестируемой модели, были получены следующие результаты:

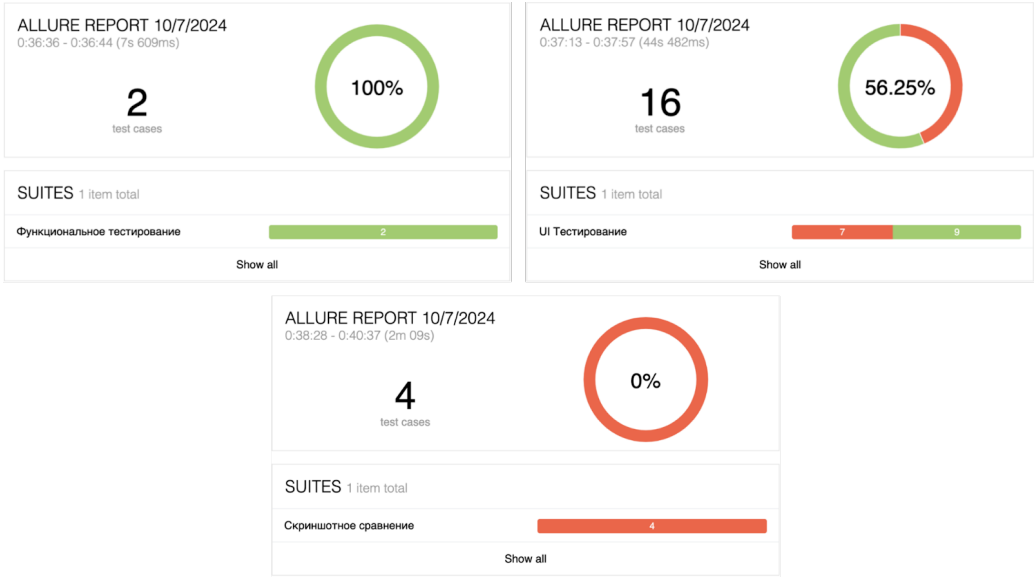

Рис. 5 — результаты функционального, UI и многоаспектного скриншотного тестирования версии ПО с внесенными ошибками

На рис. 5 показано, что функциональное тестирование не смогло полностью выполнить задачу верификации, так как его возможности не охватывают все аспекты, требующие проверки. В свою очередь, UI-тестирование выявило большее количество ошибок, что связано с более широким набором тестовых случаев, предусмотренных для проверки пользовательского интерфейса. Однако стоит отметить, что это тестирование не охватывало проверку функциональности, и поэтому не может служить полноценной заменой функциональному тестированию. В отличие от этого, многоаспектное скриншотное тестирование показало наилучший результат, выявив все ошибки и задокументировав их.

При нахождении ошибки многоаспектное скриншотное тестирование выводит в отчет сообщение, содержание которого зависит от проверяемого аспекта. Если ошибка была в визуальной составляющей, сообщение будет выглядеть следующим образом: «Failed: Ошибка при сравнении скриншотов: Разница найдена в скриншотах: 419178 пикселей отличаются от предыдущего скриншота: Проверка формы авторизации (UI). Наличие всех элементов_название файла.png», где «название файла.png» указывает на полученный после сравнения скриншот. Это упрощает процесс анализа и документации ошибок после проверки как для тестировщика, так и для разработчика.

Функциональное и UI тестирование на возникновение ошибки реагирует выводом статуса «AssertionError: <текстовое указание ошибки>», где <текстовое указание ошибки> может указывать, например «Не перешли на страницу успешной авторизации» или «Неверный color: rgba(160, 160, 160, 1)», что не столь очевидно и требует дополнительной работы для подготовки отчетности.

Результат наложения эталонной версии скриншота и той версии, которая снимается во время тестирования, показан на рис. 6. В таком виде после прохождения тестов формируются отчеты.

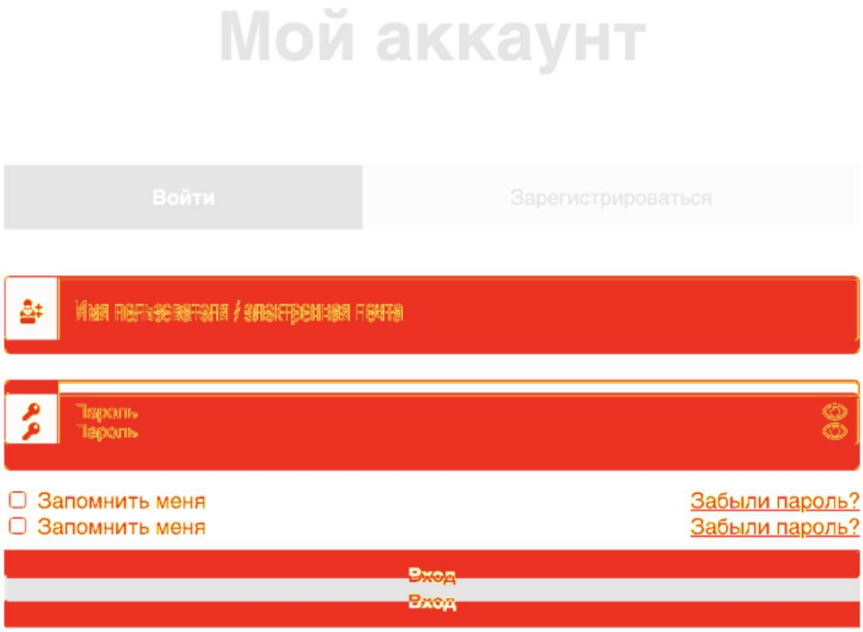

Рис. 6 — Пример итогового скриншота по результатам тестирования

По итогу проведенного эксперимента были получены следующие результаты:

Точность тестирования:

- **Функциональное тестирование**: выявлено 0 ложных срабатываний, пропущено 2 дефекта.
- **UI тестирование**: выявлено 7 дефектов, было 3 ложных срабатывания, пропущен 1 дефект.
- **Многоаспектное скриншотное тестирование**: выявлено 4 дефекта, 0 ложных срабатываний, пропущено 0 дефектов.

Для оценки точности проверяемых видов тестирования использовалась следующая формула:

$$x = \frac{y - z - f}{y + f} \times 100$$

Где x – точность, y - общее количество выявленных дефектов, z – количество ложных срабатываний, f – пропущенные дефекты.

Таким образом, получены следующие результаты:

- Точность функционального тестирования равна **100%**, однако на странице присутствовали UI дефекты, которые данным видом тестирования определяться не должны. С точки зрения поставленной задачи верификации функциональное тестирование покрывает только часть ошибок.

- Точность UI тестирования равна **37.5%**, так как было выявлено 3 ложных срабатывания и была пропущена 1 ошибка.

- Точность многоаспектного скриншотного тестирования равна **100%** так как были выявлены все дефекты без ложных срабатываний и пропущенных ошибок.

Этот расчёт подтверждает, что многоаспектное скриншотное тестирование выигрывает по

точности по сравнению с функциональным и UI тестированием, что делает его более эффективным методом для выявления дефектов в системе.

Стабильность тестовых сценариев:

Стабильность тестовых сценариев можно оценить на основе количества тестовых запусков, которые успешно проходят за несколько итераций, по сравнению с общим количеством тестовых запусков. Формула для расчёта стабильности использовалась следующая:

$$x = \frac{y}{z} \times 100$$

Где x – стабильность тестов, y – количество успешных тестовых запусков, z – общее количество тестовых запусков.

- **Успешные тестовые запуски** — это те тесты, которые прошли без сбоев или ошибок.
- **Общее количество тестовых запусков** — это общее число запусков тестов за несколько итераций.

Для получения данных по стабильности, тестовые запуски проводились несколько раз, после чего фиксировались результаты.

- **Успешные прогоны**: сколько раз тестовые запуски завершились успешно.
- **Неуспешные прогоны**: сколько раз тестовые запуски вышли из строя по причине непредсказуемых факторов (например, сбои в среде, проблемы с сетью).

Были получены следующие результаты:

1. Функциональное тестирование: 10 успешных запусков из 10. Стабильность равна 100%;
2. UI тестирование: 5 успешных запусков из 10, стабильность равна 50%;
3. Многоаспектное скриншотное тестирование: 9 успешных запусков из 10, стабильность равна 90%

Этот расчет показывает, что функциональное тестирование в рамках эксперимента показало наиболее стабильные результаты работы. Однако стоит принять во внимание то, что тестовые сценарии в рамках поставленной верификационной задачи были пройдены со значительными ошибками.

Сводную таблицу полученных результатов можно представить следующим образом:

Таблица 4

Полученные результаты в ходе эксперимента

Анализируемые положения для сравнения	**Функциональное тестирование**	**UI тестирование**	**Скриншотное тестирование**
Количество выявленных	0 из 2	7 из 16	4 из 4

дефектов			
Время выполнения тестов	11 секунд	45 секунд	28 секунд
Трудозатраты на разработку и поддержку тестов	Достаточно быстрые решения	Времязатратно, т.к. каждый аспект интерфейса требуется отдельно покрывать стилями и свойства	Чуть дольше чем функциональное, но быстрее UI тестирования
Точность тестирования, включая ложные срабатывания и пропущенные дефекты	0%	37.5%	100%
Оценка стабильности тестов	100%	50%	90%

Таким образом, гипотеза подтверждается, многоаспектное скриншотное тестирование является более эффективным видом верификации веб-приложений по сравнению с функциональным и UI тестированием.

Обсуждение результатов

Предложенный метод проходил апробацию на искусственно созданном веб-приложении с ограничением проверки только критически важной составляющей в рамках четко сформулированной ограниченной задачи верификации. Предложенный метод верификации показал свою состоятельность, однако требует дополнительной проверки при интеграции в более сложные системы и постановки развернутой задачи верификации, которая будет требовать не только поверки критически важного функционала одной страницы, но и менее критичных, однако важных критериев верификации веб-приложения.

Стоит обратить внимание, что реализация многоаспектного скриншотного тестирования может быть сложной и требовательной к ресурсам задачей. При увеличении количества проверяемых аспектов возрастает вероятность столкновения с проблемой экспоненциального роста сложности, что может привести к затруднениям в обработке всех возможных состояний системы. В контексте многоаспектного скриншотного тестирования это может проявляться в виде сложных интерфейсов с множеством динамически изменяющихся элементов, что требует ручной настройки системы оповещений.

В рамках предложенного метода мы обошли ограничение создав систему сравнения скриншотов и их последующего сохранения для отчетности. Схема такого решения

представлена выше на рис. 1. Каждый скриншот в хранилище именуется согласно определенной логике: в названии указывается дата выполнения теста, тестируемый функционал и аспект (например, это может быть тестирование одной из форм на странице и тогда скриншот будет содержать только ее, либо полная страница со всеми аспектами). В контексте регрессионного тестирования такой подход позволяет проводить регулярную проверку стабильности ПО, не затрачивая большого количества времени на прохождения однотипных тест-кейсов.

В случае работы с динамическими элементами, такими как автозамены номеров и т.д., описано решение, при котором данные элементы маскируются с помощью Javascript, что позволяет настроить автоматическую систему фильтрации для исключения их из проверки.

Также стоит подчеркнуть, что минусом многоаспектного скриншотного тестирования может являться его зависимость от вычислительных ресурсов. Поскольку метод предполагает одновременную проверку множества аспектов, это требует больших вычислительных мощностей, особенно при работе с крупными и сложными программами. Это может проявляться в необходимости хранения большого объема данных (скриншотов), а также в обработке изображений для их анализа. Это создает дополнительную нагрузку на систему и может увеличивать время выполнения тестов.

Данное ограничение можно обойти разными способами, в контексте данной работы используется компрессия изображений без потерь, которая позволяет значительно уменьшить объем сохраняемых данных. Таким образом, исходное изображение сохраняется без удаления какой-либо важной информации в формате PNG.

Также стоит учитывать, что для составления квартальной и годовой отчетности на многих проектах необходимо долговременное хранение данных. В тех случаях, когда отчеты после прохождения тестов уже обработаны и исправлены и быстрый доступ к ним не столь необходим, можно использовать методы архивации данных. Архивирование подразумевает объединение нескольких файлов в один файл с последующим сжатием для уменьшения общего объема данных. Например, при необходимости можно разбивать данные, полученные в ходе тестирования, по кварталам, в которых производились проверки, или по типу поверяемого функционала для упрощения управления файлами. Таким образом, в любой момент можно будет поднять именно тот пласт информации, который будет необходим. Архивация также является эффективным способом переноса данных для резервного копирования или передачи на удаленные серверы.

Так, реализация многоаспектного скриншотного тестирования может стать серьезной проблемой, если технические требования, процессы и методы, необходимые для конкретного ПО, не были проанализированы и описаны[9].

В контексте предложенного метода можно сформулировать следующие рекомендации для дальнейших исследований:

1. Провести дополнительные исследования по применению многоаспектного скриншотного тестирования для мобильных приложений, где влияние динамических элементов интерфейса и разнообразие устройств могут существенно усложнить процесс верификации;

2. Проверить метод многоаспектного скриншотного тестирования в рамках разных задач верификации с использование более сложного веб-приложения, что потребует разработки алгоритмов оптимизации ресурсоемкости метода при работе с

крупными и высоконагруженными приложениями;

3. Исследовать возможности интеграции многоаспектного подхода с другими видами тестирования, а также усовершенствовать предложенный метод включением в него системы, которая имитирует поведение человека при тестировании программного обеспечения, используя последние достижения в области компьютерного зрения[10].

Заключение

В рамках проведенного исследования была достигнута поставленная цель — доказана эффективность многоаспектного скриншотного тестирования как метода автоматизированной динамической верификации веб-приложений. На основе экспериментальных данных подтверждена гипотеза, что многоаспектное скриншотное тестирование превосходит традиционные методы, такие как функциональное и UI-тестирование, за счет гибридного анализа функциональных и визуальных характеристик приложения.

В ходе эксперимента, проведенного в рамках подготовки данной статьи, было показано, что возможности многоаспектного скриншотного тестирования превышают возможные ограничения и недостатки за счет сокращения времени на тестирование (особенно регрессионного), детализированного анализа разных аспектов, гибкости в изменении стратегии тестирования. Данные эксперимента могут быть экстраполированы на более сложные проектные решения, что делает применение многоаспектного скриншотного тестирования более эффективным способом динамической верификации ПО.

Предложенная методика базируется на теоретически обоснованных алгоритмах сравнения изображений, которые обеспечивают высокую точность анализа отклонений. Разработанная модель верификации позволяет эффективно структурировать тестовые сценарии и упрощает их анализ.

Таким образом, многоаспектное скриншотное тестирование позволяет создать гибкую и точную систему верификации, которая учитывает разные аспекты ПО за счет использования сразу нескольких видов тестирования в рамках одного запуска тестов, что позволяет автоматизировать сложные процессы с минимальным количеством ложных срабатываний. Более того, возможность сравнения между разными версиями приложения позволяет отслеживать изменения, что помогает убедиться в осознанном внесении правок, которые не нарушают функциональность или внешний вид ПО. Использование Python, Pytest и PIL дает возможность легко анализировать изменения между версиями и быстро выявлять неожиданные отклонения.

Библиография

1. The Economic Impacts of Inadequate Infrastructure for Software Testing. NIST Report, May 2002. Режим доступа: https://www.nist.gov/system/files/documents/director/planning/report02-3.pdf (Дата обращения: 05.10.2024).
2. Гурин Р. Е., Рудаков И. В., Ребриков А. В. Методы верификации программного обеспечения // Машиностроение и компьютерные технологии. 2015. № 10. С. 235-251.
3. Quadri S. M. K., Farooq S. U. Software testing-goals, principles, and limitations // International Journal of Computer Applications. 2010. Т. 6. № 9. С. 7-9.
4. Kumar S. Reviewing software testing models and optimization techniques: an analysis of efficiency and advancement needs // Journal of Computers, Mechanical and Management.

2023. Т. 2. № 1. С. 43-55.
5. Xie Q., Memon A. M. Designing and comparing automated test oracles for GUI-based software applications // ACM Transactions on Software Engineering and Methodology (TOSEM). 2007. Т. 16. № 1. С. 4.
6. Кудрявцева Е. Ю. Автоматизированное тестирование веб-интерфейсов // Горный информационно-аналитический бюллетень (научно-технический журнал). 2014. № S. С. 354-356.
7. Кулямин В. В. Методы верификации программного обеспечения / В. В. Кулямин. М.: ИСП РАН, 2008. 111 с.
8. Персиваль Г. Python. Разработка на основе тестирования / Г. Персиваль. М.: ДМК Пресс, 2018. 622 с.
9. Берегейко О. П., Дубовский А. С. Автоматизация тестирования веб-приложений // Вестник магистратуры. 2016. № 12-4 (63). С. 39-41.
10. Dwarakanath A., Neville D., Sanjay P. Machines that test Software like Humans. arXiv preprint arXiv:1809.09455 (2018).

Результаты процедуры рецензирования статьи

В связи с политикой двойного слепого рецензирования личность рецензента не раскрывается.
Со списком рецензентов издательства можно ознакомиться здесь.

Представленная статья на тему «Скриншотное тестирование как многоаспектный вид автоматизированной динамической верификации веб-приложений» соответствует тематике журнала «Программные системы и вычислительные методы» и посвящена вопросу обеспечения высокого уровня качества и поддержания стабильного функционирования программного обеспечения. Так как даже малозначительные ошибки могут привести к потере стабильности, ухудшению пользовательского опыта и т. д., а как следствие к экономическим издержкам, важно регулярно проводить анализ состояния программного обеспечения.
В статье представлен широкий анализ литературных российских и зарубежных источников по теме исследования.
В качестве цели исследования авторы указывают разработку метода многоаспектного скриншотного тестирования, сочетающего функциональные и UI-подходы, для улучшения точности выявления дефектов и оптимизации ресурсов в процессе верификации веб-приложений.
Стиль и язык изложения материала является научным и доступным для широкого круга читателей. Статья по объему соответствует рекомендуемому объему от 12 000 знаков.
Статья достаточно структурирована - в наличии введение, заключение, внутреннее членение основной части (методы исследования, описание предлагаемого метода, экспериментальная апробация многоаспектного скриншотного тестирования, обсуждение результатов). В работе содержится графический материал, представленный 6 рисунками, а также 4 таблицы.
В качестве методологии исследования авторы указывают разработку и применение многоаспектного скриншотного тестирования, которое представляет собой комбинацию функционального и UI-тестирования, позволяющую эффективно решать задачи верификации веб-приложений. Основная идея метода заключается в проведении параллельного анализа функциональных и визуальных характеристик веб-приложения путем автоматического сравнения эталонных и тестовых скриншотов.
Также авторами проведен эксперимент, целью которого была проверка следующий гипотезы: многоаспектное скриншотное тестирование является более эффективным

видом верификации веб-приложений по сравнению с функциональным и UI тестированием, благодаря гибридному анализу всех аспектов ПО в рамках выбранного тестового сценария за один запуск тестов. Оценка эффективности проводилась методом сравнительного анализа на примерах функционального, UI и многоаспектного скриншотного тестирования.

Практическая значимость статьи четко обоснована – доказана эффективность многоаспектного скриншотного тестирования как метода автоматизированной динамической верификации веб-приложений. На основе экспериментальных данных авторами подтверждена гипотеза, что многоаспектное скриншотное тестирование превосходит традиционные методы, такие как функциональное и UI-тестирование, за счет гибридного анализа функциональных и визуальных характеристик приложения.

Предложенный авторами метод проходил апробацию на искусственно созданном веб-приложении с ограничением проверки только критически важной составляющей в рамках четко сформулированной ограниченной задачи верификации. Предложенный метод верификации показал свою состоятельность, однако требует дополнительной проверки при интеграции в более сложные системы и постановки развернутой задачи верификации, которая будет требовать не только поверки критически важного функционала одной страницы, но и менее критичных, однако важных критериев верификации веб-приложения.

Статья «Скриншотное тестирование как многоаспектный вид автоматизированной динамической верификации веб-приложений» может быть рекомендована к публикации в журнале «Программные системы и вычислительные методы».