Скокова Ангелина Викторовна, магистрант,
ФГАОУ ВО МГТУ «СТАНКИН»

УСТОЙЧИВОСТЬ UI-АВТОТЕСТОВ ВЕБ-ПРИЛОЖЕНИЙ

Аннотация. Рассматриваются теоретические факторы устойчивости UI-автотестов веб-приложений. Показано, что хрупкость тестов обусловлена динамической структурой интерфейса, асинхронной загрузкой данных, нестабильными локаторами, слабой синхронизацией и нарушением разделения ответственности в тестовом коде. Предложена систематизация факторов устойчивости, включающая стратегический выбор уровня проверки, архитектурную организацию тестов, синхронизацию с состоянием интерфейса и диагностическую отчетность.

Ключевые слова: Автоматизированное тестирование, UI-автотесты, веб-приложение, устойчивость тестов, Page Object Model, Selenium WebDriver.

Введение

Автоматизированное тестирование является элементом инженерного управления качеством программного обеспечения. В современных процессах разработки его задача не сводится к механическому обнаружению дефектов: тестирование снижает неопределенность при принятии решений о готовности продукта к поставке и обеспечивает обратную связь о состоянии системы. Стандарт ISO/IEC/IEEE 29119-1:2022 описывает общие понятия программного тестирования и задает терминологическую основу для построения тестовой деятельности [1].

Для веб-приложений особое значение имеет автоматизация проверок пользовательского интерфейса. UI-автотесты моделируют действия конечного пользователя на уровне графического интерфейса и позволяют подтвердить прохождение критических пользовательских сценариев. Вместе с тем этот уровень тестирования наиболее чувствителен к изменению интерфейсного слоя, поэтому его эффективность определяется не количеством сценариев, а устойчивостью, сопровождаемостью и диагностической ценностью.

Цель статьи заключается в систематизации теоретических факторов, влияющих на устойчивость UI-автотестов веб-приложений, и в определении методических условий, позволяющих снизить их хрупкость при сопровождении тестового набора. Объектом рассмотрения выступают UI-автотесты веб-приложений, предметом – архитектурные, синхронизационные и процессные факторы, определяющие их устойчивость.

Нормативную основу рассмотрения составляют международные и национальные документы по программному тестированию, включая ISO/IEC/IEEE 29119-1:2022 и ГОСТ Р 56920-2024, разработанный с учетом основных положений международного стандарта ISO/IEC/IEEE 29119-1:2022 [1, 7].

Место UI-тестов в стратегии автоматизации

Тестовый набор должен строиться с учетом различий между уровнями проверок. Unit-тесты целесообразны для локальной логики и вычислений, API-тесты – для проверки бизнес-операций и взаимодействия сервисов, а UI-тесты – для сценариев, в которых существенны доступность элементов, корректность навигации, отображение состояния и целостность пользовательского пути. В материалах ISTQB тестовая пирамида рассматривается как модель планирования набора проверок, где быстрые низкоуровневые тесты образуют основание, а широкие интерфейсные проверки применяются ограниченно [2].

Следовательно, UI-автоматизация не должна подменять остальные уровни тестирования. Если значительная часть регресса выполняется только через браузер, возрастает

стоимость запуска, анализа и сопровождения, а сам тестовый набор становится более чувствительным к изменениям интерфейса. Рациональная стратегия предполагает, что через интерфейс проверяются только те риски, которые невозможно или нецелесообразно достоверно оценить на более низком уровне.

Таким образом, устойчивость UI-автотестов зависит не только от качества их программной реализации, но и от корректного выбора уровня проверки. Перенос низкоуровневых проверок на UI-уровень увеличивает долю хрупких сценариев и повышает трудоемкость регрессионного анализа.

Такой подход согласуется с контекстно ориентированным пониманием тестирования, при котором набор проверок определяется не формальным стремлением к максимальному покрытию, а рисками продукта, назначением системы и стоимостью получения достоверной информации о качестве [8].

Причины хрупкости UI-автотестов

Под хрупкостью UI-автотестов понимается способность тестов терять работоспособность при изменениях интерфейса, не нарушающих проверяемую бизнес-логику. Причины такой хрупкости можно условно разделить на четыре группы: интерфейсные, синхронизационные, идентификационные и архитектурные. Для современных веб-приложений интерфейсная страница может формироваться динамически, изменяя структуру DOM и доступность элементов в процессе выполнения сценария; элементы появляются после асинхронных запросов, меняют видимость, переинициализируются JavaScript-компонентами и зависят от роли пользователя или состояния данных.

Синхронизационная группа причин связана с несоответствием момента выполнения тестовой команды фактическому состоянию интерфейса. Документация Selenium указывает, что одной из типичных проблем браузерной автоматизации является выполнение команды до наступления нужного состояния веб-приложения; элемент должен быть не только создан в DOM, но и отображен пользователю [3]. Поэтому ожидание факта существования элемента не является достаточным условием для клика или ввода.

Идентификационная группа причин относится к локаторам. Тесты становятся хрупкими, если идентифицируют элементы по порядку вложенности, случайным CSS-классам, изменяемому тексту или длинным XPath-цепочкам. Более устойчивым является применение стабильных признаков, согласованных для целей автоматизации. Такие признаки уменьшают зависимость теста от визуальной переработки страницы и позволяют локализовать изменения в объекте страницы.

Архитектурная группа причин проявляется в дублировании действий, прямом обращении тестов к DOM и смешении бизнес-сценария с техническими деталями интерфейса. При подобной структуре изменение одного экрана приводит к множественным правкам, усложняет сопровождение и снижает доверие к регрессионному прогону.

Методические условия повышения устойчивости

Базовым архитектурным средством снижения связанности тестов с интерфейсом является Page Object Model. В документации Selenium этот подход представлен как способ моделирования страниц или их частей в виде объектов тестового кода, что позволяет уменьшить дублирование и локализовать изменения UI [4]. М. Фаулер также определяет Page Object как объектную оболочку над HTML-страницей, предоставляющую тестам прикладной интерфейс взаимодействия [5].

Однако Page Object Model эффективна только при соблюдении разделения ответственности. Само применение данного шаблона не устраняет хрупкость автоматически: при неверной организации объект страницы может превратиться в формальный слой, внутри которого сохраняются дублирование, неявные ожидания и зависимость от нестабильных

элементов интерфейса. В структуре автоматизированного тестирования тестовый метод должен отражать пользовательскую цель и фиксировать ожидаемый результат выполнения сценария. Объект страницы целесообразно рассматривать как уровень инкапсуляции локаторов, признаков загрузки интерфейса и элементарных действий с его компонентами. В свою очередь, слой бизнес-шагов должен объединять последовательности операций, обладающих предметным смыслом и соответствующих логике пользовательского взаимодействия с системой. При сохранении локаторов, механизмов ожидания и технических обходных решений непосредственно в тестовых сценариях не обеспечивается требуемый уровень архитектурной устойчивости, поскольку тесты остаются зависимыми от деталей реализации интерфейса.

Одним из ключевых условий повышения надежности автотестов является отказ от фиксированных временных задержек в пользу явных ожиданий. Ожидание должно быть связано не с абстрактным интервалом времени, а с конкретным проверяемым состоянием системы: видимостью элемента, его доступностью для взаимодействия, исчезновением индикатора загрузки, наличием ожидаемого маркера страницы, изменением статуса или переходом к следующему состоянию после выполнения действия. Такой подход обеспечивает более корректную синхронизацию теста с поведением приложения, снижает вероятность ложных срабатываний и делает причину отказа более диагностируемой.

Дополнительным фактором устойчивости автоматизированного тестирования является качество отчетности. При отсутствии достаточных диагностических данных падение теста затруднительно интерпретировать, что снижает практическую ценность автоматизации. Следовательно, отчет о выполнении автотестов должен включать такие вещи как: последовательность выполненных шагов, параметры тестового окружения, скриншоты ключевых состояний интерфейса, а также сообщения об ошибках и иную информацию, необходимую для анализа причины сбоя. На практике, указанные требования могут быть реализованы с использованием специализированных инструментов отчетности, например Allure Report, который ориентирован на представление результатов автоматизированного тестирования через шаги, статусы, вложения и историю выполнения [6]. Такой подход позволяет отделять дефект продукта от проблемы локатора, данных, окружения или тестовой логики.

Процессный аспект устойчивости UI-автотестов связан с обеспечением тестируемости интерфейса и регулярным анализом причин отказов тестовых сценариев. На этапе проектирования интерфейса должны быть предусмотрены стабильные идентификаторы ключевых элементов, предсказуемые состояния страниц и понятные сообщения об ошибках. На этапе сопровождения необходимо фиксировать, чем вызвано падение: дефектом продукта, ошибкой ожидания, изменением локатора, нестабильностью данных, инфраструктурным сбоем или изменением требований. Такая классификация обеспечивает обратную связь между разработкой, тестированием и сопровождением тестового фреймворка.

Заключение

Устойчивость UI-автотестов является интегральной характеристикой, зависящей от технических, архитектурных и процессных факторов. Хрупкость тестов формируется не только из-за особенностей браузерной автоматизации, но и вследствие неправильного выбора уровня проверки, нестабильных локаторов, недостаточной синхронизации, слабого разделения ответственности в тестовом коде и отсутствия системной диагностики результатов выполнения.

Снижение хрупкости возможно при комплексном подходе: UI-тесты должны использоваться для критических пользовательских сценариев; элементы интерфейса должны иметь стабильные признаки; действия должны выполняться после подтверждения готовности

страницы; тестовый код должен опираться на Page Object Model и слой бизнес-шагов; результаты прогонов должны сопровождаться диагностической отчетностью и классификацией причин отказов. Следовательно, устойчивость UI-автотестов следует рассматривать не как свойство отдельного тестового сценария, а как результат согласованности стратегии выбора уровня проверок, архитектуры тестового кода, механизмов синхронизации и процессов анализа результатов. При соблюдении этих условий UI-автотесты становятся сопровождаемым инструментом контроля качества веб-приложений.

Список литературы:

1. ISO/IEC/IEEE 29119-1:2022. Software and systems engineering – Software testing – Part 1: General concepts. Geneva: International Organization for Standardization, 2022. URL: https://www.iso.org/standard/81291.html (дата обращения: 10.05.2026).
2. ISTQB Certified Tester Foundation Level Syllabus v4.0.1. International Software Testing Qualifications Board, 2024. URL: https://istqb.org/wp-content/uploads/2024/11/ISTQB_CTFL_Syllabus_v4.0.1.pdf (дата обращения: 15.05.2026).
3. Selenium Documentation. Waiting Strategies. URL: https://www.selenium.dev/documentation/webdriver/waits/ (дата обращения: 15.05.2026).
4. Selenium Documentation. Page Object Models. URL: https://www.selenium.dev/documentation/test_practices/encouraged/page_object_models/ (дата обращения: 15.05.2026).
5. Fowler M. Page Object. URL: https://martinfowler.com/bliki/PageObject.html (дата обращения: 15.05.2026).
6. Allure Report Documentation. URL: https://allurereport.org/docs/ (дата обращения: 15.05.2026).
7. ГОСТ Р 56920-2024. Системная и программная инженерия. Тестирование программного обеспечения. Часть 1. Общие понятия. М.: Российский институт стандартизации, 2024.
8. Кейнер К., Бах Д., Петтикорд Б. Тестирование программного обеспечения: контекстно ориентированный подход / пер. с англ. С. Черникова. СПб.: Питер, 2025. 352 с.