НАУЧНЫЙ ОБЗОР

УДК 004.415.5:004.05

АНАЛИЗ НАДЕЖНОСТИ SELENIUM-СКРИПТОВ В РЕАЛЬНЫХ ПРОЕКТАХ

Кочетов Д.О.

Независимый исследователь и инженер по тестированию программного обеспечения, Москва, Российская Федерация, e-mail: k.dmi2016@yandex.ru

Статья посвящена критической проблеме нестабильных автоматизированных тестов в Selenium WebDriver, которые периодически завершаются сбоем без изменения исходного кода и существенно снижают надежность процессов непрерывной интеграции и доставки. Цель исследования заключается в анализе данных факторов и разработке практических рекомендаций по повышению устойчивости Selenium-скриптов в промышленных проектах. Работа выполнена в формате прикладного исследования с элементами систематизированного обзора литературы. Для поиска публикаций использованы международные и отечественные библиографические базы, охватывающие период с 2014 по 2025 г.; проанализировано 52 источника, из которых отобрано 15, наиболее полно отражающих современные подходы к стабилизации автоматизированного тестирования. Анализируются коренные причины флейковости, включая асинхронную природу структуры веб-страницы, нестабильные локаторы элементов и внешние инфраструктурные факторы. Предложен комплексный методологический подход, включающий: систему явных ожиданий специфических инвариантов пользовательского интерфейса, стратегии построения стабильных локаторов с использованием специальных атрибутов данных элементов интерфейса, ответственное применение ретраев и мониторинг ключевых метрик надежности. Особое внимание уделяется системе метрик для количественной оценки стабильности тестов, включая долю нестабильных тестов, частоту повторных запусков и среднее время восстановления. Работа содержит практические рекомендации по интеграции решений в тестовые пайплайны. Результаты исследования представляют ценность для инженеров по тестированию и разработчиков, стремящихся повысить надежность процессов автоматизированного тестирования.

Ключевые слова: нестабильные тесты, надежность автоматизированного тестирования, Selenium WebDriver, стабильные локаторы, метрики стабильности, CI/CD

ANALYSIS OF SELENIUM SCRIPT RELIABILITY IN REAL-WORLD PROJECTS

Kochetov D.O.

Independent researcher and quality assurance-engineer, Moscow, Russian Federation, e-mail: k.dmi2016@yandex.ru.

The article addresses the critical issue of unstable automated tests in Selenium WebDriver, which intermittently fail without any changes to the source code and significantly reduce the reliability of continuous integration and delivery processes. The aim of the study is to analyze these factors and develop practical recommendations for improving the stability of Selenium scripts in industrial projects. The work is conducted as an applied study with elements of a systematic literature review. International and domestic bibliographic databases covering the period from 2014 to 2025 were used to search for publications; 52 sources were analyzed, of which 15 were selected as most representative of current approaches to stabilizing automated testing. The study analyzes the root causes of instability, including the asynchronous nature of the web page structure (Document Object Model), unstable element locators, and external infrastructural factors. A comprehensive methodological approach is proposed, which includes a system of explicit waits for user interface invariants, strategies for constructing stable locators using special data attributes, responsible use of retries, and monitoring of key reliability metrics. Particular attention is paid to a system for the quantitative assessment of test stability, including indicators such as the proportion of unstable tests, retry frequency, and mean time to recovery. The paper provides practical recommendations for integrating these solutions into test pipelines. The results of the study are of value to test engineers and developers seeking to improve the reliability of automated testing processes.

Keywords: flaky tests, reliability of automated testing, Selenium WebDriver, stable locators, stability metrics, CI/CD

Введение

Selenium WebDriver является фактически стандартом для автоматизации end-to-end (E2E) тестирования веб-интерфейсов благодаря кроссплатформенной поддержке браузеров, развитой экосистеме инструментов и глубокой интеграции в конвейеры непрерывной интеграции и доставки (CI/CD). Однако ценность E2E-покрытия, обеспечивающего проверку системы в целом, сопряжена с высокими рисками нестабильности тестов. Основные источники этой нестабильности включают асинхронную природу Document Object Model (DOM), высокую динамику одностраничных приложений (SPA) и инфраструктурные факторы, такие как задержки в сети или нагрузка на серверы. В современных исследованиях отмечается, что стабильность тестов критически зависит от качества локаторов и стратегий

ожидания элементов [1, 2]. Официальная документация Selenium подчеркивает, что ключом к созданию устойчивых тестов является применение корректных стратегий ожидания загрузки элементов и использования надежных методов их поиска.

Проблема надежности автоматизированных тестов является центральной в обеспечении качества программного обеспечения, и ее ключевым аспектом выступает феномен нестабильных, или так называемых flaky-тестов. Flaky-тест определяется как тест, который способен демонстрировать как успешный, так и неуспешный исход при абсолютно неизменном коде тестируемого приложения и самом коде теста. Эта недетерминированность напрямую подрывает возможность однозначной интерпретации результатов тестирования и разрушает доверие разработчиков к процессу непрерывной интеграции (CI/CD пайплайну) [3].

В последние годы наблюдается рост исследований, направленных на изучение причин возникновения flaky-тестов и методов повышения стабильности E2E-тестов в реальных промышленных проектах [4, 5]. Эти работы подчеркивают важность применения корректных стратегий ожидания, устойчивых локаторов и мониторинга метрик стабильности для обеспечения доверия к результатам автоматизированного тестирования.

Цель исследования – анализ коренных причин нестабильности Selenium-скриптов и разработка практических рекомендаций по повышению их устойчивости в промышленных проектах, включая стратегическое применение явных ожиданий, построение стабильных локаторов и мониторинг метрик стабильности тестов [1, 6, 7].

Материал и методы исследования

Исследование выполнено в формате систематизированного обзора литературы, направленного на обобщение современных подходов к повышению стабильности Selenium-скриптов и снижению числа нестабильных тестов в проектах с CI/CD-интеграцией. Поиск источников проводился с использованием ключевых слов на русском и английском языках: «нестабильные тесты», «flaky tests», «Selenium WebDriver reliability», «explicit waits», «stable locators», «CI/CD test automation». Отбор публикаций осуществлялся в международных и отечественных библиографических базах данных – ACM Digital Library, IEEE Xplore, SpringerLink, Scopus, eLibrary и CyberLeninka. В анализ включались работы, опубликованные в период с 2016 по 2025 г., что позволило учесть как фундаментальные исследования, так и практические решения последних лет, применяемые в промышленной автоматизации тестирования. В общей сложности рассмотрено 52 источника, из которых 15 были отобраны для детального анализа. В обзор включались статьи, содержащие результаты эмпирических исследований, статистику по нестабильности тестов, а также описания практических методик по повышению надежности тестовых сценариев. В поле внимания вошли публикации, посвященные выбору и настройке локаторов, управлению ожиданиями элементов, влиянию инфраструктурных факторов (задержек, загрузки браузера), а также интеграции стратегий стабилизации тестов в конвейеры непрерывной интеграции. Исключались источники, ограниченные модульным тестированием, не содержащие экспериментальной части или конкретных практических рекомендаций. Для подготовки настоящего систематического обзора использованы современные принципы, изложенные в протоколе PRISMA 2020 [8].

Результаты исследования и их обсуждение

Многочисленные эмпирические исследования систематически анализируют коренные причины нестабильности. Установлено, что основные источники нестабильности заключаются в проблемах асинхронности и состояний гонки (race conditions), связанных с некорректными стратегиями ожидания загрузки элементов в пользовательском интерфейсе. Эти выводы подтверждаются обзором T. Amjed, где UI- и инфраструктурные причины флейков классифицированы в многоголосом обзоре [4]. Дополнительно, анализ последних промышленных кейсов показывает, что использование data-атрибутов и CSS-селекторов позволяет добиться наибольшей устойчивости тестов без ущерба для покрытия [1]. Значительный вклад вносят нестабильные и хрупкие локаторы элементов, которые теряют свою актуальность при изменениях в структуре DOM-дерева [9]. Дополнительными факторами риска являются неконтролируемые зависимости от внешней инфраструктуры, состояния сети, сторонних сервисов, а также взаимное влияние тестов друг на друга при определенном порядке их выполнения. Следует особо подчеркнуть, что для UI-тестов, в особенности веб-приложений, совокупное негативное влияние этих факторов является наиболее выраженным по сравнению с другими уровнями тестирования [10].

Для эффективного управления проблемой нестабильных тестов необходима комплекс-

ная система метрик, позволяющая количественно оценивать уровень стабильности и эффективность применяемых мер. В таблице приведен комплексный подход к устранению нестабильности тестов, включая стратегические направления, конкретные меры и соответствующие метрики эффективности. Ряд современных исследований и практических руководств подчеркивает необходимость использования метрик Flaky Test Rate (FTR), Retry Rate и MTTR для объективной оценки стабильности тестов в CI/CD [4, 11]. В работе Olianas et al. (2025) демонстрируется, что визуализация FTR и Retry Rate в CI-дэшбордах повышает скорость реагирования команд на флейковые тесты [12].

Ключевые метрики включают:

– Flaky Test Rate (FTR): процент тестов, проявляющих нестабильное поведение за определенный период времени.

– Retry Rate: среднее количество повторных запусков, необходимых для получения стабильного результата.

– Mean Time To Detect (MTTD): среднее время обнаружения нестабильного теста.

– Mean Time To Repair (MTTR): среднее время устранения коренной причины нестабильности.

– False Positive Rate: процент ложных срабатываний в общем количестве падений.

– Регулярный мониторинг этих метрик позволяет объективно оценивать эффективность стратегий борьбы с нестабильностью и принимать обоснованные решения по оптимизации тестовых процессов.

Практические последствия распространения flaky-тестов носят крайне негативный характер и имеют серьезные операционные издержки. К ним относится значительное увеличение времени обратной связи (feedback time) для разработчиков из-за необходимости многократных перезапусков тестовой сборки для верификации результатов, а также рост числа ложных срабатываний (false positives), которые требуют бесполезной траты времени на анализ несуществующих дефектов. Это подтверждается отчетами индустриальных компаний по CI/CD, где массовые повторные прогоны приводят к значительной потере доверия к автотестам [4, 5].

Широко распространенная практика маскировки проблемы путем массового использования повторных прогонов приводит к накоплению большого количества избыточной статистики, которая затрудняет анализ реальных проблем. Как следствие, кульминацией этих проблем становится полная или частичная потеря доверия разработчиков к автотестам как к надежному инструменту.

Для обеспечения стабильности тестовых скриптов настоятельно рекомендуется системное использование механизмов явных ожиданий, таких как WebDriverWait в сочетании с ExpectedConditions (или их аналогами на выбранном языке программирования). Ключевой принцип заключается в том, чтобы дожидаться достижения конкретного целевого инварианта состояния элемента, а не произвольной задержки. К наиболее критически важным ожидаемым условиям относятся: visibilityOf (видимость элемента), elementToBeClickable (готовность элемента к взаимодействию), presenceOfElementLocated (присутствие элемента в DOM-дереве), invisibilityOf (исчезновение элемента) и stalenessOf («устаревание» элемента, например, при обновлении DOM). Категорически не рекомендуется смешивать данный подход с неявными ожиданиями (implicit waits) на уровне сессии, так как это приводит к непредсказуемому наложению таймаутов и существенно снижает детерминированность поведения теста [7].

Комплексный подход к устранению нестабильности тестов

Стратегическое направление	Конкретные меры	Метрики эффективности
Стабильные локаторы	Использование data-атрибутов (data-testid), приоритет CSS над XPath, избегание позиционных селекторов	Снижение количества StaleElement ReferenceException [1, 7]
Явные ожидания	Применение WebDriverWait с Expected-Conditions, отказ от Thread.sleep()	Уменьшение времени выполнения тестов [1, 13]
Контролируемые ретраи	Ограниченное количество попыток (2–3), экспоненциальная задержка между попытками	Снижение Retry Rate [4, 11]
Мониторинг и анализ	Регулярный расчет FTR, ведение реестра flaky-тестов, анализ root-причин	Снижение MTTR, уменьшение количества тестов в карантине [4, 12]

Примечание: составлена автором на основе анализа литературных источников, официальной документации Selenium и собственных наблюдений в рамках исследования

Рекомендации по правильной настройке ожиданий подробно описаны в официальной документации Selenium и индустриальных best-practice руководствах [1, 2].

Пример корректной реализации на Java:

```
WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
WebElement btn = wait.until(ExpectedConditions.elementToBeClickable(
    By.cssSelector("[data-testid='checkout-submit']")));
btn.click();
```

К числу распространенных антипаттернов, которых следует избегать, относятся: использование «жестких» пауз (Thread. sleep), попытки взаимодействия с невидимыми элементами, ожидание лишь факта присутствия элемента в DOM без проверки его готовности к клику, а также смешение стратегий неявных и явных ожиданий, что ведет к непредсказуемым суммарным задержкам.

Стратегия повторных запусков реализуется на двух основных уровнях: локально на уровне тестового фреймворка и глобально на уровне конвейера CI/CD. На уровне фреймворка для Python-проектов, использующих pytest, применяется плагин pytest-rerunfailures с указанием количества повторов (--reruns) и, опционально, задержки между ними через конфигурацию в pytest. ini. В экосистеме Java для TestNG типичным решением является реализация интерфейса IRetryAnalyzer или использование специализированных listener'ов для автоматизации прогонов.

Крайне важен принцип ответственного применения данной стратегии. Прогоны позволяют смягчить «шум» от случайных флейков, но они не должны маскировать реальные дефекты в коде. Это подтверждается анализом причин флейковости и практических рекомендаций по их смягчению, а также в обзорах по best-practice масштабируемой автоматизации, где подчеркивают ответственность за настройку ретраев и ограничение их использования [2, 4]. Для оценки влияния внешних факторов на проявление флейков и корректной интерпретации повторов полезно учитывать результаты экспериментов по влиянию вычислительных ресурсов [11]. Кроме того, исследования по анализу UI-flaky тестов рекомендуют обязательно фиксировать нестабильные тесты (с помощью меток или помещения в карантин) и проводить их триаж и root-cause анализ до того, как ретраи станут повсеместной практикой [14]. Необходимо обязательно фиксировать факт нестабильности теста (через систему меток или карантин), собирать статистику по срабатываниям, ограничивать максимальное количество попыток (N) и использовать экспоненциальную задержку (back-off) между ними, а также проводить постоянный триаж и анализ коренных причин флейковости.

Выбор и конструирование устойчивых локаторов базируется на нескольких ключевых принципах. Наивысший приоритет имеют стабильные и уникальные атрибуты элемента: в первую очередь ID (при условии его гарантированной стабильности), а затем CSS-селекторы, основанные на неизменяемых атрибутах. Наиболее эффективной практикой является включение в продуктовый код специальных атрибутов-хуков для тестирования (таких как data-testid или data-qa), что полностью развязывает тест от изменений в верстке или стилях. Следует избегать сложных XPath-выражений, жестко завязанных на структуру DOM-дерева и позиционные индексы; допускается использование лишь относительно коротких XPath, построенных от стабильных «якорных» элементов [7].

Данные рекомендации напрямую поддержаны официальной документацией Selenium и best-practice руководствами ведущих индустриальных провайдеров.

Корректная настройка временных параметров является необходимым компонентом стабильности. Необходимо явно и адекватно настраивать таймауты загрузки страницы (pageLoadTimeout) и выполнения асинхронных скриптов (scriptTimeout). В сценариях, предполагающих динамическое обновление интерфейса (например, после AJAX-запроса или навигации), следует избирательно использовать ожидание условия stalenessOf(element) по отношению к старому элементу непосредственно перед попыткой поиска его обновленной версии в DOM. Это гарантирует, что последующие операции будут выполняться с актуальным состоянием элемента, а не с его устаревшей ссылкой, что предотвращает классические ошибки типа StaleElementReferenceException.

Для устойчивого управления качеством тестирования и достижения предсказуемости процессов непрерывной интеграции необходимы операциональные и объективно измеряемые метрики. Ключевой количественной мерой является Flake Rate (FR) – процент запусков, в которых тест демонстрировал нестабильное поведение, вычисляемый как отношение числа флей-

ковых исходов к общему числу запусков, умноженное на 100%. Альтернативной оценкой может служить Flakiness Rate, определяемый как доля неконсистентных результатов на заданном интервале наблюдения, методология которого варьируется в зависимости от поставщика аналитики. Для глубинного анализа необходимо отслеживать Failure Rate применительно именно к флейковым тестам, а также строить тренды проявления нестабильности в разрезе отдельных тестов, модулей или компонентов системы, что поддерживается современными системами CI-телеметрии.

Метрика MTTR-test-flake (Mean Time To Repair) отражает среднее время, затрачиваемое на устранение причины флейка или перевод теста в карантин с момента его обнаружения, что характеризует скорость реакции команды. Retry ratio, или среднее количество повторных запусков, необходимое для достижения успешного прохождения тестов, служит индикатором общей стабильности пайплайна; его рост является четким сигналом деградации надежности. Quarantine count, или количество тестов, временно исключенных из основного прогона, также требует мониторинга. Длительное нахождение теста в карантине без анализа и устранения первопричин указывает на системные риски, которые маскируются, а не решаются. Современные исследования подчеркивают необходимость регулярного анализа этих метрик в контексте CI/CD, поскольку именно визуализация MTTR и Retry Ratio позволяет оперативно выявлять деградацию стабильности пайплайна [2, 13].

Для стратегического планирования тестового покрытия критически важна метрика Coverage-vs-Stability, анализирующая соотношение доли end-to-end тестов к модульным и интеграционным, а также их соответствующий вклад в общий Flake Rate; это позволяет следовать рекомендациям по сокращению доли «тяжелых» E2E-тестов, вносящих наибольший вклад в нестабильность. Подобные подходы активно применяются в промышленной практике крупных компаний, где осуществляется приоритизация тестов на основе показателя Coverage-vs-Stability [12, 15].

Все перечисленные метрики должны собираться автоматически непосредственно в процессе CI (с использованием build scans или дашбордов) и быть доступными для детального анализа с декомпозицией по типам тестов, используемым фреймворкам, компонентам системы и тестовым окружениям. Для автоматизированного сбора таких показателей используются встроенные средства CI-платформ (например, Jenkins Build Scans, GitLab Analytics, Azure DevOps Insights) а ключевые метрики (FR, MTTR-flake, Retry Ratio, Quarantine Count) должны регулярно отображаться на дашбордах и анализироваться командами на совещаниях по качеству для своевременного принятия корректирующих действий [14, 15].

Для системного повышения стабильности автотестов командам рекомендуется придерживаться следующих принципов [16]. При проектировании тестов необходимо использовать исключительно явные ожидания (explicit waits), настроенные на конкретные и проверяемые инварианты состояния пользовательского интерфейса (например, кликабельность, видимость), и полностью избегать смешения стратегий неявных (implicit) и явных ожиданий в рамках одной сессии. Краеугольным камнем устойчивости является применение стабильных локаторов, поэтому для этого целесообразно внедрять в продуктовый код специализированные атрибуты (такие как data-testid или data-qa) для ключевых элементов, что обеспечивает полную независимость тестов от изменений в верстке, и избегать использования хрупких XPath-выражений, зависящих от абсолютной позиционной структуры DOM. Инженерия тестовых окружений должна включать эмуляцию реалистичных сетевых условий (через DevTools Protocol) и обеспечение контроля над ресурсами (CPU, память) раннеров в CI-системе для минимизации внешнего шума.

В результате тестовый контур становится не только технически устойчивым, но и управленчески предсказуемым, что напрямую снижает риск сбоев в поставке продукта.

Заключение

Проведенное исследование подтвердило, что проблема нестабильных тестов (flaky tests) остается одной из ключевых в индустрии автоматизации тестирования. Несмотря на развитие инструментальных решений, коренные причины флейковости по-прежнему связаны с асинхронностью DOM, отсутствием детерминированных ожиданий и зависимостью тестов от нестабильной инфраструктуры.

На основании анализа источников показано, что комплексный подход, включающий оптимизацию локаторов, внедрение метрик стабильности (FR, MTTR, Retry Ratio) и системную работу с техническим долгом тестов, позволяет существенно повысить устойчивость CI/CD-пайплайнов и доверие к результатам автотестов.

Практические рекомендации, изложенные в работе, направлены на формирование

культуры ответственности за стабильность тестовой инфраструктуры и развитие аналитического подхода к мониторингу метрик надежности. В совокупности это создает предпосылки для перехода к интеллектуальным системам анализа качества и дальнейшего применения методов машинного обучения для адаптивного сопровождения тестов.

Список литературы

1. Nass M., Alegroth E., Feldt R., Leotta M., Ricca F. Similarity-based web element localization for robust test automation // Proceedings / article (ACM / arXiv). 2022. URL (arXiv): https://arxiv.org/pdf/2208.00677 (дата обращения: 13.10.2025).

2. Joe D. Best Practices for Scalable Test Automation with Selenium // International Journal of Advanced and Innovative Research. 2022. Vol. 11. Is. 1. P. 918–928. URL: https://www.researchgate.net/publication/383914679_Best_Practices_for_Scalable_Test_Automation_with_Selenium (дата обращения: 13.10.2025).

3. Shruti A., Nitin Ch. Exploring the Benefits and Challenges of Automated Testing // Journal of Software Engineering and Systems Testing. 2023. P. 1–15. URL: https://admin.mantechpublications.com/index.php/JoSEST/issue/viewFile/6553/7156 (дата обращения: 13.10.2025).

4. Amjed T., Shawn R., Jens D., Negar H., Lu Zh. Test flakiness' causes, detection, impact and responses // Journal of Systems and Software. 2023. P. 112186. URL: https://www.sciencedirect.com/science/article/pii/S0164121223002327 (дата обращения: 13.10.2025). DOI: 10.1016/j.jss.2023.111837.

5. Balsam S., Mishra D. Web application testing – Challenges and opportunities // Journal of Systems and Software. 2024. P. 112186. URL: https://www.sciencedirect.com/science/article/pii/S0164121224002309 (дата обращения: 13.10.2025). DOI: 10.1016/j.jss.2024.112186.

6. Артюхова А.С. Проблемы автоматизации тестирования и подходы к их решению // CETERIS PARIBUS. 2016. № 10. С. 5–11. URL: https://elibrary.ru/wxqkzd?ysclid=mi0egewyvm564870742 (дата обращения: 13.10.2025). EDN: WXQKZD.

7. Ткачев А.В., Иртегов Д.В. Методика автоматического тестирования развивающегося веб-приложения // Журнал информационных технологий. 2019. Т. 17. Вып. 3. С. 78–91. URL: https://journals.nsu.ru/jit/archive/2019/vypusk-3-tom-17-2019/razdel/metodika-avtomaticheskogo-testirovaniya-razvivayushchegosya-veb-prilozheniya/ (дата обращения: 13.10.2025). DOI: 10.25205/1818-7900-2019-17-3-93-110.

8. Page M.J., McKenzie J.E., Bossuyt P.M., Boutron I., Hoffmann T.C., Mulrow C.D., Shamseer L., Tetzlaff J.M., Akl E.A., Brennan S.E., Chou R., Glanville J., Grimshaw J.M., Hróbjartsson A., Lalu M.M., Li T., Loder E.W., Mayo-Wilson E., McDonald S., McGuinness L.A., Stewart L.A., Thomas J., Tricco A.C., Welch V.A., Whiting P., Moher D. The PRISMA 2020 statement: an updated guideline for reporting systematic reviews // BMJ. 2021. Vol. 372. n71. URL: https://www.bmj.com/content/372/bmj.n71 (дата обращения: 13.10.2025). DOI: 10.1136/bmj.n71.

9. Xuan J., Monperrus M. Test Case Purification for Improving Fault Localization // Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE). 2014. P. 52–63. URL: https://arxiv.org/abs/1409.3176 (дата обращения: 13.10.2025). DOI: 10.1145/2635868.2635906.

10. Luo Q., Hariri F., Eloussi L., Marinov D. An Empirical Analysis of Flaky Tests // Proceedings of the 2014 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2014). ACM, 2014. P. 1–11. URL: https://mir.cs.illinois.edu/marinov/publications/LuoETAL-14FlakyTestsAnalysis.pdf (дата обращения: 13.10.2025). DOI: 10.1145/2635868.2635920.

11. Denini Silva D., Gruber M., Gokhale S., Arteca E., Turcotte A., d'Amorim M., Lam W., Winter S., Bell J. The Effects of Computational Resources on Flaky Tests // arXiv preprint. 2023. URL: https://arxiv.org/abs/2310.12132 (дата обращения: 13.10.2025).

12. Olianas D., Leotta M., Ricca F., Biagiola M., Tonella P. STILE: a tool for optimizing E2E web test scripts and parallel execution // Software Quality Journal // ScienceDirect (conference/journal source). 2025. URL: https://www.sciencedirect.com/science/article/pii/S0164121224003480 (дата обращения: 13.10.2025).

13. Koppanati P. Handling Dynamic Web Elements in Selenium for Robust Automation // European Journal of Advances in Engineering and Technology. 2021. Vol. 8. Is. 2. P. 138–143. URL: https://www.researchgate.net/publication/387743058_Handling_Dynamic_Web_Elements_in_Selenium_for_Robust_Automation (дата обращения: 13.10.2025).

14. Romano A., Song Z., Grandhi S., Yang W., Wang W. An Empirical Analysis of UI-based Flaky Tests // ICSE. 2021. P. 1. URL: https://weihang-wang.github.io/papers/UIFlaky-icse21.pdf (дата обращения: 13.10.2025).

15. Manukonda K.R.R. A Comprehensive Evaluation of Selenium WebDriver for Cross-Browser Test Automation: Performance, Reliability, and Usability // Journal of Artificial Intelligence, Machine Learning & Data. 2023. URL: https://www.researchgate.net/publication/382027459_A_Comprehensive_Evaluation_of_Selenium_Webdriver_for_Cross-Browser_Test_Automation_Performance_Reliability_and_Usability (дата обращения: 13.10.2025).

16. Gruber M., Heine M., Oster N., Philippsen M., Fraser G. Practical Flaky Test Prediction using Common Code Evolution and Test History Data // arXiv preprint. 2023. URL: https://arxiv.org/abs/2302.09330 (дата обращения: 19.10.2025). DOI: 10.48550/arXiv.2302.09330.

Конфликт интересов: Авторы заявляют об отсутствии конфликта интересов.

Conflict of interest: The authors declare that there is no conflict of interest.