НАУЧНЫЙ ОБЗОР

УДК 004.052.42
DOI

МЕТОДЫ ПОВЫШЕНИЯ НАДЕЖНОСТИ АВТОМАТИЗИРОВАННОГО ТЕСТИРОВАНИЯ ВЕБ-ПРИЛОЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ JAVA, SELENIUM И CI/CD-ИНФРАСТРУКТУРЫ

Кочетов Д.О.

Независимый исследователь и инженер по тестированию ПО, Москва, e-mail: k.dmi2016@yandex.ru

Автоматизация тестирования пользовательских интерфейсов значительно ускоряет процессы контроля качества, однако ее надежность снижается из-за нестабильности локаторов, задержек при загрузке контента и зависимости от внешних сервисов. Непредсказуемость выполнения тестов приводит к увеличению трудозатрат на поддержку инфраструктуры и снижению доверия к результатам. Цель исследования заключается в анализе и классификации современных методов повышения надежности автоматизированного тестирования веб-приложений с учетом инструментальной базы Java и Selenium, интеграции с контурами непрерывной интеграции и непрерывной доставки (Continuous Integration / Continuous Delivery), а также применения технологий машинного обучения и интеллектуальных систем для адаптивного сопровождения тестов. Работа выполнена в формате систематизированного обзора литературы. Для поиска публикаций использованы международные и отечественные библиографические базы, охватывающие период с 2016 по 2025 г.; проанализировано 106 источников, из которых отобрано 25. Рассмотрены исследования и практические рекомендации, включающие автоматизацию управления драйверами браузеров, переход к явным ожиданиям, применение паттерна Page Object Model, контейнеризацию тестовой инфраструктуры, параллельные запуски сценариев и использование аналитических инструментов для выявления нестабильных тестов. Отдельное внимание уделено технологиям машинного обучения, направленным на автоматическую адаптацию локаторов и повышение воспроизводимости сценариев. Анализ представленных подходов показывает, что сочетание инструментальных и организационных методов позволяет существенно повысить надежность тестовых процессов, снизить затраты на поддержку и ускорить обратную связь для разработчиков. Заключение отражает актуальность перехода к интеллектуальным системам адаптивного сопровождения тестов и интеграции облачных решений, что формирует перспективные направления дальнейших исследований.

Ключевые слова: автоматизированное тестирование, Selenium, надежность тестов, CI/CD, Java, flaky-тесты

METHODS FOR IMPROVING THE RELIABILITY OF AUTOMATED WEB APPLICATION TESTING USING JAVA, SELENIUM, AND CI/CD INFRASTRUCTURE

Kochetov D.O.

Independent researcher and QA-Engineer, Moscow, e-mail: k.dmi2016@yandex.ru

User interface testing automation significantly accelerates quality assurance processes; however, its reliability decreases due to unstable locators, delays in content loading, and dependence on external services. The unpredictability of test execution increases maintenance costs for infrastructure and reduces confidence in results. The aim of this study is to analyze and classify modern approaches to improving the reliability of automated web application testing, considering the use of Java and Selenium tools, integration with Continuous Integration and Continuous Delivery pipelines, as well as the application of machine learning technologies and intelligent systems for adaptive test maintenance. The research is presented as a systematic literature review. The search for publications was conducted in international and Russian bibliographic databases covering the period from 2016 to 2025; a total of 106 sources were analyzed, of which 25 were selected. The reviewed studies and practical recommendations address automated browser driver management, the transition to explicit waits, the implementation of the Page Object Model pattern, containerization of the testing infrastructure, parallel execution of scenarios, and the use of analytical tools to detect unstable tests. Special attention is given to machine learning technologies aimed at automatic locator adaptation and improving test reproducibility. The analysis demonstrates that combining technical solutions with organizational practices can significantly enhance the reliability of testing processes, reduce maintenance costs, and accelerate feedback for development teams. The conclusion emphasizes the relevance of transitioning to intelligent systems for adaptive test maintenance and integrating cloud-based solutions, outlining promising directions for future research.

Keywords: automated testing, Selenium, test reliability, CI/CD, Java, flaky tests

Введение

Автоматизированные инструменты тестирования, включая Selenium, демонстрируют высокую эффективность, но их надежность остается уязвимой. Даже минимальные изменения в структуре DOM способны нарушить работу локаторов, а асинхронное выполнение приводит к непредсказуемым результатам. Появление flaky-тестов снижает доверие разработчиков

и вынуждает тратить значительные ресурсы на отладку. Дополнительные сложности создают зависимые сервисы и инфраструктурные компоненты, от которых напрямую зависит успешность прогонов. Современные практики веб-тестирования тесно связаны с распределенными окружениями и системами непрерывной интеграции. Это усложняет процесс и увеличивает вероятность сбоев: сетевые задержки, несогласованность драйверов и ошибки синхронизации могут проявляться в виде ложных отказов. В условиях распределенных систем автоматизация сталкивается с рядом типичных уязвимостей – нестабильные сценарии, обусловленные асинхронностью, и хрупкие локаторы интерфейсов, требующие регулярной поддержки. Все это снижает воспроизводимость и подрывает устойчивость тестовых процессов.

Цель исследования – проанализировать и классифицировать современные методы повышения надежности автоматизированного тестирования веб-приложений, с учетом инструментальной базы Java и Selenium, интеграции с CI/CD-контурами, а также применения технологий машинного обучения и интеллектуальных систем для адаптивного сопровождения тестов.

Материалы и методы исследования

Работа выполнена в формате систематизированного обзора, что позволило охватить современные подходы к повышению надежности автоматизированного тестирования веб-приложений и сопоставить их по критериям применимости в проектах на базе Java, Selenium и CI/CD-инфраструктур. Для поиска источников использовались ключевые слова на русском и английском языках: «flaky-тесты», «устойчивость автотестов», «Selenium reliability», «CI/CD test automation», «Java testing frameworks», «test flakiness mitigation». Подбор литературы проводился в международных и отечественных библиографических базах (ACM Digital Library, IEEE Xplore, SpringerLink, Scopus, eLibrary, CyberLeninka). Временные рамки ограничены периодом с 2016 по 2025 г., что обеспечивает учет как фундаментальных работ последних лет, так и новейших решений, связанных с внедрением ML-подходов и облачных инструментов. В обзор были включены 25 публикаций из 106, содержащие эмпирические данные или предложенные методики, связанные с устойчивостью автоматизированного тестирования пользовательских интерфейсов. Рассматривались статьи, посвященные управлению драйверами, выбору стратегий локаторов, использованию распределенных сред, контейнеризации и интеграции с CI/CD-системами. Исключались материалы, описывающие только модульное тестирование без связки с UI-слоем, работы без экспериментальной базы, а также обзоры без конкретных практических предложений.

Результаты исследования и их обсуждение

Практика, согласно исследованию Б. Гарсия и соавт., показывает: во многих командах драйверами Selenium WebDriver попрежнему управляют вручную – скачивают, настраивают, поддерживают. Такая рутина повышает входной порог и добавляет точки отказа: несовпадение версий браузера и драйвера приводит к ложным падениям, а обновляющиеся «вечнозеленые» браузеры (Chrome, Firefox, Edge, Opera) регулярно ломают сборки сообщением вроде «данная версия chromedriver поддерживает только Chrome N». Устойчивость набора UI-тестов в этих условиях страдает, а стоимость сопровождения растет. Снять часть рисков помогают проверенные практики на уровне кода: корректная работа с ожиданиями и локаторами, а также применение шаблона Page Object Model, уменьшающего связанность и издержки на поддержку. В инфраструктуре Java-проекта дополнительные дивиденды дает автоматизированное управление драйверами (WebDriverManager) и интеграционные расширения для JUnit 5 (например, Selenium-Jupiter), которые убирают значительную долю ручной конфигурации [1].

Фиксированные паузы Thread.sleep выглядят простым решением, но на длинных прогонах они раздувают время выполнения и сами по себе вносят недетерминизм. Гораздо надежнее явные ожидания: условие выполнено – ожидание завершено; нет причины ждать – нет лишних секунд. Такой подход позволяет синхронизироваться с динамическим контентом, проверять сложные предикаты и заметно снижать долю «флейки»-падений. В зрелых кодовых базах логично проводить «санитизацию» тестов: целенаправленно убирать sleep и заменять их на explicit waits, добиваясь сокращения общего времени пайплайна без появления новых источников нестабильности [2].

Локаторы – один из самых хрупких элементов веб-автотестов: любая правка верстки или атрибутов легко выводит из строя сценарий. Наиболее живучими считаются идентификаторы id – они обычно уникальны и меняются редко, однако доступны не всегда. XPath универсален, но абсолютные пути ломаются чаще относительных. Согласно исследованию D. Clerissi, M. Leotta и F. Ricca повысить качество набора помогает системная профилактика: ревизии локаторов, переход на более устойчивые стратегии, а также «игровые» механики, стиму-

лирующие разработчиков тестов (например, регулярные задания по замене абсолютных XPath на относительные с накоплением «очков» и ачивок) [3]. Такая мотивация делает практики робастности регулярной привычкой, а не разовой кампанией.

Selenium остается базовым выбором для кросс-браузерной автоматизации благодаря открытой экосистеме и множеству интеграций [4]. Вместе с тем отдельные задачи эффективнее решают специализированные инструменты: Applitools применяет системы автоматизированного визуального распознавания для визуальных расхождений [5]; Sahi упрощает старт за счет встроенной среды, но уступает в гибкости крупным фреймворкам. В мире Java широкое распространение получил Selenide: он снимает слой шаблонного кода, предлагает встроенные «умные» ожидания и тем самым облегчает сопровождение и снижает вероятность ошибок синхронизации [6; 7].

Анализ методов уровня тест-дизайна демонстрирует, что устойчивость и предсказуемость UI-тестов зависят не только от применяемых инструментов, но и от системного пересмотра практик их построения. Автоматизация управления драйверами, переход от фиксированных пауз к явным ожиданиям и осознанная работа с локаторами позволяют снизить вероятность «флейки»-падений и сделать пайплайн менее ресурсоемким. Совмещение этих приемов с более зрелыми библиотеками и интеграционными расширениями формирует основу для снижения стоимости сопровождения и повышения доверия к результатам автоматизированного тестирования.

Распределенный запуск на Selenium Grid позволяет раскидать тесты по нескольким машинам и конфигурациям. Архитектура hub-and-node дает нужную масштабируемость, сокращает время обратной связи и повышает надежность за счет проверки в разных средах. Такой режим органично встраивается в Agile-процессы: дефекты всплывают раньше, а непрерывное тестирование перестает быть узким местом доставки [8].

Контейнеры привносят воспроизводимость и изоляцию, платформы уровня ElasTest используют Docker для сборки и развертывания SUT, подключают инструментирование и рекомендательные механизмы на ML, что помогает бороться со сквозной нестабильностью. Гибкость достигается как готовыми образами из Docker Hub, так и оркестрацией многоконтейнерных систем через Docker Compose [9]. На стороне CI/CD все больше внимания уделяется автоматической генерации пайплайнов: это снижает ручной труд, сокращает ошибки конфигурации и ускоряет выпуск – в эксперименте Н.И. Дьяченко и А.В. Забродина полный цикл от коммита до новой версии занял 5 мин 37 с без перерыва в работе сервиса [10].

Согласно исследованию B. García и соавт. E2E-сценарии падают «черным ящиком», и без клиентских артефактов первопричину не видно. Систематический сбор консольных логов и трассировок из автоматизированных браузеров закрывает этот пробел. Поскольку ручной сбор дорогостоящий и уязвимый к ошибкам, полезны расширения класса BrowserWatcher, которые добавляют наблюдаемость «на грани браузера». Предупреждения в консоли – хороший предиктор будущих проблем, их профилактическая очистка снижает риск повторяющихся падений [11].

Эффективность инфраструктурных решений для автоматизированного тестирования проявляется в их способности совмещать масштабируемость и воспроизводимость с глубокой наблюдаемостью. Контейнеризация и оркестрация не только сокращают время обратной связи, но и формируют устойчивый контур, где непредсказуемость среды минимизируется. Встраивание инструментов логирования и мониторинга в связке с CI/CD позволяет превратить тестовый процесс из набора разрозненных процедур в целостный механизм, поддерживающий надежность веб-приложений на всех стадиях жизненного цикла.

GitLab CI/CD, Jenkins и сходные системы позволяют формировать цепочки этапов, где порядок тестов отражает логику контроля качества. Запуски происходят автоматически при каждом изменении, неуспешные результаты блокируют слияние и возвращают обратную связь разработчику. Для UI-сценариев критичны скорость и детерминизм [12]. Масштабирование достигается параллельными прогонами на распределенном парке агентов вместе с Selenium Grid. При этом необходимо добиваться независимости тестов и управлять нестабильными кейсами – ограничивать время шага, применять высокоуровневые библиотеки поверх WebDriver, настраивать ожидания, использовать карантин для проблемных тестов до их исправления [13].

Шаблонизация и синтез пайплайн из метаданных проекта стандартизируют процесс, снимают рутину и уменьшают вероятность человеческих ошибок. Этапы сборки, тестирования и деплоя подбираются под специфику репозитория, что упрощает поддержку и делает поведение конвейера предсказуемым. Испытания показывают, что такой подход реально снижает трудозатраты и стабилизирует выпуск [10].

По мнению J. Bell и соавт. повторные прогоны для выявления «флейки» дороги

и тормозят цикл. Альтернатива – анализ покрытия последних изменений: если упавший тест не исполнял измененные строки, падение с высокой вероятностью flaky [14]. Инструменты класса DeFlaker реализуют эту идею и позволяют отмечать нестабильные тесты без каскада перезапусков. R. Khankhoje отмечает, что параллельно применяются стратегические меры: изоляция нестабильных кейсов, приоритизация исправления первопричин, а не «перезапуск как лекарство» [15].

Эффективность CI/CD-практик во многом определяется тем, насколько инженерные команды умеют сочетать автоматизацию рутинных действий с активным управлением нестабильными сценариями. Совмещение шаблонного синтеза пайплайнов, аналитических инструментов для выявления «флейки» и стратегической изоляции проблемных тестов превращает тестовую инфраструктуру в адаптивную систему, способную поддерживать баланс между скоростью поставки и качеством продукта. Такая организация процессов снижает издержки на сопровождение и укрепляет доверие к результатам автоматизированного контроля качества.

Чтобы уменьшить ложные срабатывания, из сравнения исключают «шум» – всплывающие уведомления, анимации и прочие волатильные элементы. Маскирование и фильтры позволяют концентрироваться на значимых отклонениях макета и контента. Технологический стек часто включает Selenium для взаимодействия с UI, библиотеки обработки изображений (например, PIL) и параллелизацию на уровне тестового раннера (Pytest-xdist). Параллельные прогоны по наборам устройств/браузеров дают приемлемое время обратной связи. Линия развития – приближение к «человеческому» восприятию интерфейса: использование компьютерного зрения для семантического сравнения экранов [16]. Коммерческие решения вроде Applitools и Functionize применяют ИИ для устойчивого визуального сравнения и даже «самовосстановления» сценариев при несущественных изменениях UI, что снижает издержки поддержки [5].

Визуальное тестирование постепенно выходит за рамки «механического» сравнения скриншотов и превращается в интеллектуальный инструмент оценки интерфейсов. Совмещение методов фильтрации шумовых артефактов с компьютерным зрением и ИИ-платформами формирует новый уровень надежности: система способна различать критические расхождения и игнорировать незначительные визуальные изменения без участия человека. В результате снижается не только число ложных срабатываний, но и издержки сопровождения, а тестовые сценарии начинают отражать восприятие интерфейса конечным пользователем, а не только формальные расхождения пикселей.

Крупные языковые модели (GPT-4, Codex, CodeT5) уже умеют анализировать код и синтезировать осмысленные тестовые случаи, демонстрируя высокие показатели покрытия и точности. При этом сохраняется необходимость экспертной валидации: «галлюцинации» и нерелевантные проверки пока не редкость, а стоимость ложных тревог в CI/CD велика [17]. Как отмечают D.P. Nguyen и S. Maag, частая причина падений – эволюция DOM и атрибутов. Self-healing-подходы пытаются адаптироваться автоматически: модели распознают целевой элемент по набору признаков и перенастраивают шаги взаимодействия. «Бескодовое» тестирование с ML-надстройкой снижает объем ручного рефакторинга при изменениях интерфейса [18]. Интеграция машинного обучения включает сбор датасетов, извлечение признаков, обучение, онлайн-инференс, генерацию тестов и цикл обратной связи. В ряде экспериментов такие пайплайны показывают прирост надежности и эффективности по сравнению с чисто скриптовыми решениями [19].

Применение ML/AI в автоматизации тестирования демонстрирует переход от реактивной поддержки сценариев к проактивному самовосстановлению, где система не только фиксирует поломку, но и предлагает корректировку на основе накопленных закономерностей. Такой сдвиг открывает возможность рассматривать тестовую инфраструктуру как адаптивную экосистему, в которой алгоритмы снижают стоимость ручного рефакторинга и повышают устойчивость пайплайна к изменениям интерфейса, сохраняя при этом контроль за качеством за счет экспертной верификации.

Переиспользование библиотек ускоряет разработку, но одновременно приносит уязвимости. Команды нередко затягивают обновления из-за боязни регрессий и накладных расходов на обслуживание зависимостей. Исследования отмечают: более половины security-апдейтов, созданных ботами, доходят до слияния. Наличие тестов и CI повышает готовность принимать такие PR – снижается риск нарушить обратную совместимость незамеченным дефектом. По мнению H. Mohayeji и соавт., Dependabot централизует обнаружение и обновление уязвимых пакетов через автоматические pull-requests. Для больших портфелей репозиториев это становится действенным механизмом поддержания базовой гигиены цепочки поставок [20].

Классификация методов повышения надежности автотестов
Источник: составлено автором

Чек-лист по слоям автоматизированного тестирования

Слой	Проблемы	Метод решения
Код (тест-дизайн)	– несовместимость версий браузеров и драйверов при ручном управлении; – задержки и нестабильность из-за Thread.sleep; – хрупкость локаторов (особенно абсолютных XPath)	– автоматизация управления драйверами; – замена фиксированных пауз на явные ожидания; – применение Page Object Model для снижения связности; – переход на устойчивые стратегии локаторов (id, относительные XPath), регулярные ревизии; – использование высокоуровневых библиотек
Инфраструктура	– длительные прогоны без параллелизации; – нестабильность среды и трудности воспроизведения; – недостаточная наблюдаемость (отсутствие логов и артефактов)	– запуск на распределенных узлах (Selenium Grid); – контейнеризация и оркестрация (Docker, Docker Compose); – интеграция логирования и мониторинга; – автоматическая очистка и поддержка тестовых данных
CI/CD	– ошибки ручной настройки пайплайнов; – зависимость скорости поставки от нестабильных кейсов; – высокая стоимость повторных прогонов для поиска flaky-тестов	– автоматическая генерация и шаблонизация пайплайнов; – масштабирование за счет параллельных запусков; – управление flaky-тестами (карантин, ограничение времени шага); – анализ покрытия для выявления нестабильных тестов
ML/AI	– высокая стоимость ручного рефакторинга при изменениях DOM; – ложные тревоги и «галлюцинации» моделей; – медленная адаптация сценариев к меняющемуся UI	– использование self-healing локаторов; – генерация тестов на основе ML-моделей; – сбор датасетов и обучение моделей на истории запусков; – экспертная валидация результатов

Источник: составлено автором на основе полученных данных в ходе исследования.

Современные практики работы с зависимостями показывают, что автоматизация обновлений превращается из вспомогательного инструмента в критический элемент безопасности: централизованные механизмы наподобие Dependabot минимизируют человеческий фактор и систематизируют устранение уязвимостей. Эффективность этих решений напрямую зависит от зрелости CI/CD-контура и качества тестового покрытия, позволяя командам одновременно снижать риск регрессий и поддерживать устойчивость цепочки поставок на уровне, недостижимом при ручной поддержке. На рисунке систематизированы методы повышения надежности автоматизированного тестирования.

Масштаб проблемы подтверждается индустриальными цифрами: доля «флейки» в крупных компаниях измеряется процентами от миллионов прогонов. Снижение ложноположительных/ложноотрицательных срабатываний и изоляция тестов от внешних зависимостей заметно укрепляют доверие к конвейеру [21; 22]. Для прогнозирования нестабильности применяются байесовские модели, вплоть до сетей, дающих существенный выигрыш в стабильности CI [21]. В более общем виде используются байесовские схемы с учетом априорной информации (в том числе с регрессионным ее взвешиванием), а также методы статистического контроля процессов: карты Шухарта для независимых показателей и статистика Хотеллинга для многомерных зависимых метрик [23; 24].

Как отмечают A. Ahmad, O. Leifler и K. Sandahl, оперативные дашборды и хранилища артефактов (логи, скриншоты, видео) ускоряют поиск причины падений и упрощают выявление «флейки» [21]. Визуализация тенденций по ключевым показателям помогает целенаправленно снижать технический долг набора тестов. Комплексные стратегии тестирования – от модульных до нагрузочных – при встроенности в конвейер повышают качество и уменьшают вероятность попадания дефектов в продакшн, что подтверждается прикладными исследованиями [22].

Анализ метрик показывает, что надежность автотестов перестает быть субъективной оценкой и превращается в количественно измеряемый показатель, где статистические модели и процессный контроль задают объективные границы стабильности CI. Интеграция байесовских прогнозов с оперативной визуализацией и артефактами позволяет не только быстрее локализовать «флейки», но и формировать стратегии снижения операционных издержек разработки.

Для масштабируемых наборов на Java + Cucumber + Selenium важны принципы модульности: независимые сценарии, Page Object, параметризация, регулярная актуализация, контроль версий и обязательная отчетность. В Java-экосистеме Selenide упрощает чтение и поддержку сценариев за счет декларативных API и «умных» ожиданий. Cucumber настраивается на автозапуск по каждому изменению кода; Selenide без труда «срастается» с JUnit/TestNG и типовыми конвейерами (Jenkins, GitLab CI/CD, Travis). Это делает включение UI-контролей в поток поставки рутинной процедурой, а не спецоперацией [25; 7].

Стабильность достигается не только кодом, но и гигиеной окружения: регулярная очистка зомби-процессов на узлах Grid, автоматическое управление драйверами, подготовка тестовых данных. Для ускорения – параллельные прогоны и распределенные агенты, но при условии независимости тестов [13]. Cucumber Reports, Allure, ReportPortal, Grafana и другие инструменты обеспечивают прозрачность: от результатов конкретного запуска до динамики ключевых метрик [25; 13]. Это поддерживает дисциплину качества и ускоряет обратную связь. Переход на Selenide имеет смысл оформлять как последовательный план: подготовка среды, перенос и рефакторинг сценариев, настройка ожиданий, интеграция с CI/CD [7, с. 65]. Итог – более стабильный, читаемый и удобный для сопровождения набор UI-тестов. В таблице отображены ключевые проблемы в автоматизированном тестировании и методы их решения.

Систематизация методик разработки и опора на проверенные библиотеки позволяют превратить автоматическое тестирование в управляемый и воспроизводимый процесс, а не в набор разрозненных практик. Четкая организация сценариев в связке с Cucumber и Selenide, дополненная инструментами мониторинга и отчетности, формирует прозрачный цикл обратной связи и закрепляет дисциплину качества. В результате тестовый контур становится не только технически устойчивым, но и управленчески предсказуемым, что напрямую снижает риск сбоев в поставке продукта.

Заключение

В условиях CI/CD разработчики вынуждены балансировать между скоростью получения обратной связи и стабильностью прогонов. Повторное выполнение тестов помогает фильтровать нестабильные результаты, но замедляет выпуск продукта, тогда как их пропуск чреват пропуском критических ошибок. Этот компромисс остается одной из ключевых проблем. Практика показывает, что параллельное выполнение сценариев ускоряет процесс, но одновременно повышает риск возникновения состояний гонки и flaky-поведения. Даже успешно прошедшие тесты в одном окружении не гарантируют корректности в другом: различия в версиях библиотек, браузеров и конфигурациях приводят к ложным отказам и ставят под сомнение достоверность полученных данных. Анализ показывает, что дальнейшее развитие методов должно быть сосредоточено на повышении устойчивости тестов и сокращении числа flaky-сбоев. Требуется больше исследований в области автоматической адаптации локаторов и интеграции тестовых решений с облачными CI/CD-средами. Эти направления формируют перспективные задачи для будущих исследований.

Список литературы

1. García B., Muñoz-Organero M., Alario-Hoyos C., Kloos C.D. Automated driver management for Selenium WebDriver // Empirical Software Engineering. 2021. Vol. 26, Is. 5. P. 1–50. URL: https://link.springer.com/article/10.1007/s10664-021-09975-3 (дата обращения: 22.07.2025). DOI: 10.1007/s10664-021-09975-3.

2. Olianas D., Leotta M., Ricca F. SleepReplacer: a novel tool-based approach for replacing thread sleeps in selenium WebDriver test code // Software Quality Journal. 2022. Vol. 30. P. 1089–1121. URL: https://link.springer.com/article/10.1007/s11219-022-09596-z (дата обращения: 22.07.2025). DOI: 10.1007/s11219-022-09598-5.

3. Clerissi D., Leotta M., Ricca F.A Gaming Quest to Improve Web Locators Robustness // Gamify 2024: Proceedings of the 3rd ACM International Workshop on Gamification in Software Development, Verification, and Validation. 2024. P. 10–17. URL: https://dl.acm.org/doi/abs/10.1145/3678869.3685684 (дата обращения: 22.08.2025). DOI: 10.1145/3678869.3685684.

4. García B., Gallego M., Gortázar F., Muñoz-Organero M. A Survey of the Selenium Ecosystem // Electronics. 2020. Vol. 9, Is. 7. Art. 1067. P. 1–29. URL: https://www.mdpi.com/2079-9292/9/7/1067 (дата обращения: 28.07.2025). DOI: 10.3390/electronics9071067.

5. Mohammed A.A.F., Ibrahim K.A. GUI Automation Testing Tools: a Comparative Study // Excellence Journal for Engineering Sciences. 2024. Vol. 1, Is. 1. P. 44–62. URL: https://www.researchgate.net/profile/Ali-Abdalftah-Fadul/publication/385893512_GUI_Automation_Testing_Tools_a_Comparative_Study/links/673a2f1937496239b2c359ba/GUI-Automation-Testing-Tools-a-Comparative-Study.pdf (дата обращения: 22.07.2025).

6. Satheesh A., Singh M. Comparative Study of Open Source Automated Web Testing Tools: Selenium and Sahi // Indian Journal of Science and Technology. 2017. Vol. 10, Is. 13. P. 1–9. URL: https://sciresol.s3.us-east-2.amazonaws.com/IJST/Articles/2017/Issue-13/Article4.pdf (дата обращения: 22.08.2025). DOI: 10.17485/ijst/2017/v10i13/109048.

7. Маркевич Д.В., Хомоненко А.Д. Обновление стека инструментов для тестирования: причины и шаги перехода с Selenium на Selenide // Интеллектуальные технологии на транспорте. 2024. № 2 (38). С. 57–68. URL: https://cyberleninka.ru/article/n/obnovlenie-steka-instrumentov-dlya-testirovaniya-prichiny-i-shagi-perehoda-s-selenium-na-selenide (дата обращения: 22.07.2025). DOI: 10.20295/2413-2527-2024-238-57-68.

8. Manukonda K.R.R. Enhancing test automation coverage and efficiency with Selenium Grid: a study on distributed test-

ing in Agile environments // International Journal of Advanced Research in Engineering and Technology (IJARET). 2024. Vol. 15, Is. 3. P. 119–127. URL: https://iaeme.com/Home/article_id/IJARET_15_03_011 (дата обращения: 22.07.2025).

9. García B., Gallego M., Gortázar F., López L. ElasTest, an open-source platform to ease end-to-end testing // Challenges and Opportunities in ICT Research Projects (EPS Madrid 2017). 2017. P. 3–21. URL: https://www.scitepress.org/Link.aspx?doi=10.5220/0007904700030021 (дата обращения: 22.07.2025). DOI: 10.5220/0007904700030021.

10. Дьяченко Н.И., Забродин А.В. Автоматическая генерация пайплайнов CI/CD на основе метаданных проекта: новый подход к ускорению разработки программного обеспечения // Интеллектуальные технологии на транспорте. 2025. № 1 (41). С. 74–82. URL: https://cyberleninka.ru/article/n/avtomaticheskaya-generatsiya-payplaynov-ci-cd-na-osnove-metadannyh-proekta-novyy-podhod-k-uskoreniyu-razrabotki-programmnogo (дата обращения: 22.07.2025). DOI: 10.20295/2413-2527-2025-141-74-82.

11. García B., Ricca F., del Alamo J.M., Leotta M. Enhancing web applications observability through instrumented automated browsers // Journal of Systems and Software. 2023. Vol. 201. P. 1–18. URL: https://www.sciencedirect.com/science/article/pii/S0164121223001188 (дата обращения: 22.07.2025). DOI: 10.1016/j.jss.2023.111723.

12. Никифоров А.В. Как внедрение непрерывной интеграции и тестирование помогает обеспечить соответствие требованиям при разработке программного продукта // International Journal of Professional Science. 2023. № 4. С. 88–96. URL: https://cyberleninka.ru/article/n/kak-vnedrenie-nepreryvnoy-integratsii-i-testirovanie-pomogaet-obespechit-sootvetstvie-trebovaniyam-pri-razrabotke-programmnogo (дата обращения: 22.07.2025).

13. Кириллов С.С. Внедрение автоматизированных тестов в систему непрерывной интеграции // Научные вести. 2022. № 7 (48). С. 37–44. URL: https://elibrary.ru/item.asp?edn=qmkxra (дата обращения: 22.07.2025).

14. Bell J., Legunsen O., Hilton M., Eloussi L., Yung T., Marinov D. DeFlaker: automatically detecting flaky tests // Proceedings of the 40th International Conference on Software Engineering (ICSE'18) (Gothenburg, Sweden, May 27 – June 3, 2018). ACM, 2018. P. 1–12. DOI: 10.1145/3180155.3180164.

15. Khankhoje R. Strategies for mitigating flaky tests in automated environments // International Journal of Science and Research (IJSR). 2019. Vol. 8, Is. 3. P. 1950–1954. URL: https://www.ijsr.net/getabstract.php?paperid=SR231228182000 (дата обращения: 22.07.2025). DOI: 10.21275/SR231228182000.

16. Макаров К.С., Фаткин Р.И. Скриншотное тестирование как многоаспектный вид автоматизированной динамической верификации веб-приложений // Программные системы и вычислительные методы. 2025. № 1. С. 32–54. URL: https://www.nbpublish.com/library_read_article.php?id=73535 (дата обращения: 22.07.2025). DOI: 10.7256/2454-0714.2025.1.73535.

17. Mishra A. Generative AI in Automated Software Testing: A Comparative Study // International Journal of Science and Technology (IJST). 2024. Vol. 2, Is. 1 (Jan–Mar). P. 1–13. URL: https://www.ijstjournal.com/wp-content/uploads/journal/published_paper/volume-2/issue-1/IJST241006.pdf (дата обращения: 22.07.2025).

18. Nguyen D.P., Maag S. Codeless web testing using Selenium and machine learning // Proceedings of the 15th International Conference on Software Technologies (ICSOFT). SciTePress, 2020. P. 51–60. URL: https://hal.science/hal-02909787/ (дата обращения: 22.07.2025). DOI: 10.5220/0009885400510060.

19. Alsuwailem G., Alharbi O. Utilizing machine learning for predicting software faults through Selenium testing tool // International Journal of Computations, Information and Manufacturing (IJCIM). 2023. Vol. 3, Is. 2. P. 13–27. URL: https://www.journals.gaftim.com/index.php/ijcim/article/view/309 (дата обращения: 22.07.2025). DOI: 10.54489/ijcim.v3i2.309.

20. Mohayeji H., Agaronian A., Constantinou E., Zannone N., Serebrenik A. Securing dependencies: A comprehensive study of Dependabot's impact on vulnerability mitigation // Empirical Software Engineering. 2025. Vol. 30. Article № 89. P. 1–37. URL: https://link.springer.com/article/10.1007/s10664-025-10638-w(дата обращения: 22.07.2025). DOI: 10.1007/s10664-025-10622-7.

21. Ahmad A., Leifler O., Sandahl K. Empirical analysis of factors and their effect on test flakiness: practitioners' perceptions // Software Testing, Verification & Reliability. 2021. P. 1–24. URL: https://onlinelibrary.wiley.com/doi/full/10.1002/stvr.1791 (дата обращения: 28.07.2025). DOI: 10.1002/stvr.1791.

22. Esther D. Automated testing strategies for quality assurance in CI/CD pipelines // ResearchGate. 2024. P. 1–5. URL: https://www.researchgate.net/publication/385286073 (дата обращения: 22.07.2025).

23. Андреев А.Г., Казаков Г.В., Корянов В.В. Метод оценки надежности программного обеспечения по результатам испытаний на этапе его разработки // Инженерный журнал: наука и инновации. 2016. № 6 (54). С. 1–15. URL: https://cyberleninka.ru/article/n/metod-otsenki-nadezhnosti-programmnogo-obespecheniya-po-rezultatam-ispytaniy-na-etape-ego-razrabotki (дата обращения: 22.07.2025). DOI: 10.18698/2308-6033-2016-6-1504.

24. Клячкин В.Н., Карпунина И.Н. Статистические методы оценки стабильности функционирования технических систем // Надежность и качество сложных систем. 2018. № 2 (22). С. 36–42. URL: https://cyberleninka.ru/article/n/statisticheskie-metody-otsenki-stabilnosti-funktsionirovaniya-tehnicheskih-sistem (дата обращения: 22.07.2025). DOI: 10.21685/2307-4205-2018-2-5.

25. Абдурайимов Л.Н. Создание масштабируемых автоматических тестов с использованием Java Cucumber и Selenium // Информационно-компьютерные технологии в экономике, образовании и социальной сфере. 2023. № 1 (39). С. 5–14. URL: https://www.elibrary.ru/item.asp?id=50524423 (дата обращения: 22.07.2025).