A Systematic Literature Review on Frontend Testing

Vlad-Andrei Ichim[a,*,1], Dragoș-Mihai Ignat[a] and Mircea Măierean[a]

[a]Faculty of Mathematics and Computer Science, Babeș-Bolyai University, Mihail Kogălniceanu 1 Street, Cluj-Napoca, 400347, Cluj, Romania

ARTICLE INFO

Keywords:
Artificial-intelligence-based testing
Web application testing
Automated test generation
Test robustness and self-healing
E2E Testing

ABSTRACT

This systematic review surveys web-application testing research from 2019–2024 that employs AI techniques. Fifteen studies cluster into four themes: (1) language- and vision-model generation of end-to-end tests, (2) reinforcement-learning explorers for GUI coverage, (3) robustness tactics such as self-healing locators and dependency detection, and (4) methods that convert UI executions into API suites and specifications. Evaluations track failure detection, coverage, effort and time, revealing efficiency gains but persistent gaps in benchmarks and industrial validation.

1. Systematic literature review (SLR)

Front-end testing plays an essential role in ensuring the quality, usability, and reliability of modern web applications, particularly as user interfaces grow more complex and dynamic. With the increasing diversity of devices, browsers, and interaction patterns, testing the frontend has become both more essential and more challenging. This systematic literature review provides a comprehensive analysis of the most relevant research papers on frontend testing techniques published in the past five years, with a focus on test case generation, execution, automation tools, and strategies for improving robustness and coverage in user interface validation.

2. Study design

2.1. Review need identification

The increasing complexity of modern web applications, driven by dynamic content, rich user interactions, and various device compatibility, has made front-end testing a critical and challenging aspect of software quality assurance. As user interfaces evolve rapidly, ensuring correctness, usability, and responsiveness across browsers and platforms demands robust and scalable testing approaches. This systematic literature review explores recent advancements in front-end testing, analysing key techniques, tools, and challenges identified in research over the past years.

2.2. Research questions definition

This literature review seeks to answer the following research questions regarding the current state of test case prioritization techniques:

1. RQ1: What are the main types and strategies used in modern frontend and web testing?
2. RQ2: How are tests generated, prioritized, or adapted to reflect real-world use?
3. RQ3: How are AI and machine learning influencing web testing?
4. RQ4: What are the current trends in tool usage and evaluation for frontend testing?
5. RQ5: What methods support test robustness, maintainability, and optimization?

2.3. Protocol definition

This systematic literature review (SLR) was conducted using a curated set of 30 research papers related to web application testing, with a focus on front-end validation techniques. From this initial pool, 15 papers were selected based on their direct relevance to frontend testing, the diversity of proposed approaches, the clarity of their evaluation methods, recent publication dates, and their overall contribution to the field. Key data was extracted from each paper, and a qualitative synthesis was performed to identify recurring themes, patterns, and significant differences across studies, in order to answer the defined research questions regarding modern frontend testing practices.

3. Conducting the SLR

3.1. Search and selection process

3.1.1. Database search

The initial set of 30 papers was identified through prior research efforts, encompassing a broad range of publications related to test case prioritization techniques.

3.1.2. Merging, and duplicates and impurity removal

Not applicable as the initial dataset was a specifically selected list of articles.

3.1.3. Application of the selection criteria

The initial pool of 30 papers was narrowed down to 15 based on the following criteria: relevance to frontend testing of web applications, diversity of techniques and methodologies, clarity and rigor of evaluation approaches, recent publication date, future prospects of automating testing generation and the overall informativeness and contribution of each paper to the field.

3.2. Data extraction

15 relevant articles were selected for this systematic literature review based on the defined criteria. These articles are detailed in Table 1.

3.3. Data synthesis

The following section provides summaries of each of the 15 selected articles, highlighting the main technique

Id	Citation	Title, Authors	Year
P1	Kertusha, Assress, Duman and Arcuri (2025)	A Survey on Web Testing: On the Rise of AI and Applications in Industry	2025
P2	Alian, Nashid, Shahbandeh, Shabani and Mesbah (2025)	Feature-Driven End-To-End Test Generation	2025
P3	Zheng, Liu, Xie, Liu, Ma, Hao and Liu (2021)	Automatic Web Testing using Curiosity-Driven Reinforcement Learning	2021
P4	Li, Huang, Cui, Towey, Ma, Li and Xia (2025)	A Survey on Web Application Testing: A Decade of Evolution	2025
P5	Wang, Wang, Fan, Li and Liu (2024)	Leveraging Large Vision Language Model For Better Automatic Web GUI Testing	2024
P6	Yandrapally, Sinha, Tzoref-Brill and Mesbah (2023)	Carving UI Tests to Generate API Tests and API Specification	2023
P7	Ali and Xiaoling (2019)	A Reliable and an Efficient Web Testing System	2019
P8	Sherifi, Slhoub and Nembhard (2024)	The Potential of LLMs in Automating Software Testing: From Generation to Reporting	2024
P9	Hasan, Rahman, Chowdhury, Rahman, Abdulle, Sadia and Hasan (2022)	Testing React Single Page Web Application using Automated Tools	2022
P10	Devarapalli (2019)	Frontend Testing Improvement: A New Approach to Ensuring Web Application Quality	2019
P11	Dobbala (2022)	Validate Faster, Develop Smarter: A Review of Frontend Testing Best Practices and Frameworks	2022
P12	García, Leotta, Ricca, Whitehead, Prasetya, Vos and Storm (2024)	AI-Generated Test Scripts for Web E2E Testing with ChatGPT and Copilot: A Preliminary Study	2024
P13	Mollah and van den Bos (2023)	From User Stories to End-to-End Web Testing	2023
P14	Leotta, Stocco, Ricca and Tonella (2015)	Using Multi-Locators to Increase the Robustness of Web Test Cases	2015
P15	Biagiola, Stocco, Mesbah, Ricca and Tonella (2019)	E2E Web Test Dependency Detection using NLP	2019

Table 1
The list of selected papers

proposed, the methods used for evaluation, the key findings, and the contributions to the field.

Article 1: A Survey on Web Testing: On the Rise of AI and Applications in Industry. This systematic survey reviews 214 peer-reviewed papers on web application testing published between 2014 and April 2024. It maps publication trends, venues, research contributions, topics, tools and industrial uptake. Activity has remained steady, with ICST the leading conference and Selenium the dominant experimental framework. Most studies propose novel automated or AI-assisted techniques – especially black-box test generation – but comparatively few involve human participants, large open-source suites or industrial case studies. Only one-third of the 160-plus tools released are open source, and just 17% of papers report collaborations with industry. AI and search-based methods appear in 62 studies, spanning reinforcement learning, computer-vision-driven UI analysis and natural-language guidance. Emerging challenges include flaky tests, locator fragility, and test maintenance. The authors call for richer datasets, stronger empirical evaluations and closer academia–industry partnerships to translate promising research into robust, scalable practice. Overall, the survey charts a decade's progress and outlines pivotal future directions ahead.

Article 2: Feature-Driven End-To-End Test Generation. This paper presents AUTOE2E, a new approach that uses large language models to generate meaningful E2E tests based on real app features instead of random UI actions. The authors define what they mean by a "feature" and introduce a new metric called feature coverage to better evaluate how well tests reflect actual user flows. They also built a dataset called E2EBENCH, which includes 8 real open-source web apps. For each app, they listed out the main features and tracked what happens when users interact with them. Their tool, AUTOE2E, explores the app, looks at what users can do on each page, and guesses what feature those actions belong to. It uses LLMs to make smart guesses and ranks the most likely features. The interesting part is that it can understand more complicated features that take several steps, and the tests it creates feel closer to what a real person would do on the site.

Article 3: Automatic Web Testing using Curiosity-Driven Reinforcement Learning. In order to test web applications more intelligently, this paper presents WebExplor, an automated testing tool that leverages reinforcement learning. WebExplor uses a curiosity-based reward system to learn how to explore apps on its own, rather than relying on pre-written scripts or models. During testing, it creates a deterministic finite automaton to direct investigation and

steer clear of repetitive tasks. WebExplor was tested by the researchers on fifty real-world websites, six open-source projects, and a commercial SaaS application. According to the results, it performed better than Selenium and other current tools in terms of failure detection and code coverage. WebExplor even discovered 12 new bugs in the commercial app, which were later verified and fixed by developers, demonstrating the app's value in practical testing.

Article 4: A Survey on Web Application Testing: A Decade of Evolution. Covering 314 peer-reviewed studies published between 2014 and 2023, this survey presents the most comprehensive map of web-application testing since 2014. Output has plateaued at 25–40 papers per year, with conferences producing 71% of work. Model-based and combinatorial optimisation techniques dominate test-case generation, yet AI-driven approaches are accelerating. Over 90% of studies run fully automated executions in local environments, revealing a scalability gap for cloud-native testing. Effectiveness is commonly judged by failure detection rate or DOM/code coverage, while efficiency focuses on generation and execution time. The authors catalogue more than 140 tools—41% security scanners, 39% functional/regression frameworks—and expose the limited release of open-source artefacts. Persistent challenges include flaky GUI locators, balancing coverage versus cost, unreliable failure diagnosis, and the absence of standardised benchmarks. They urge stronger empirical designs, public datasets, and academia–industry collaboration, forecasting that LLMs, virtual DOM metrics, and cloud pipelines will shape the next decade of rigorous experimentation.

Article 5: Leveraging Large Vision Language Model For Better Automatic Web GUI Testing. This paper presents VETL, a novel method for automated web GUI testing that handles two major testing tasks: selecting the appropriate elements to interact with and producing meaningful text inputs using Large Vision-Language Models. VETL works by analyzing screenshots and website structure to understand what inputs are needed, then uses LVLMs to fill in forms and click buttons, like how a human would. It also uses a smart exploration strategy to decide whether to follow what it already knows or try something new. Compared to older tools like WebExplor, VETL found 25% more web actions and detected real bugs in both open-source and commercial websites, showing its strong potential in GUI testing.

Article 6: Carving UI Tests to Generate API Tests and API Specification. APICARV is the first technique that automatically converts existing end-to-end UI test suites into efficient API-level tests and a matching OpenAPI specification. While the UI tests execute, APICARV records HTTP traffic, filters static resource calls, and constructs an API graph whose nodes represent URL segments. Dynamic graph analysis and three rounds of directed probing identify path parameters, unseen endpoints and missing HTTP operations; the resulting templates are assembled into a coherent specification and the confirmed call sequences are emitted as executable REST tests that bypass the fragile UI layer. Experiments on seven open-source web applications (150 kLoC, 117 endpoints) show that carved API suites retain the same code coverage as the original UI tests yet run over ten times faster. The inferred specifications reach 98% precision and expose specification inconsistencies, and augmenting EvoMaster and Schemathesis with the carved tests boosts branch coverage by up to 99% and statement coverage by 52%.

Article 7: A Reliable and an Efficient Web Testing System. This paper explores how web applications can be tested more effectively to make them reliable and efficient. The authors talk about both static and dynamic testing methods, like black box, white box, and grey box testing, and stress how important it is to start testing early—especially during the requirements phase. They highlight that having good documentation, like Software Requirements Specification, helps catch bugs early on and saves time later. Instead of doing experiments, they mainly review existing research and tools, comparing different strategies and discussing which tools work best depending on the project. One interesting point is their focus on non-functional testing (like performance or usability), which they say is often ignored but really important for quality web software.

Article 8: The Potential of LLMs in Automating Software Testing: From Generation to Reporting. This paper proposes a framework that uses Large Language Models (LLMs) like Gemini and ChatGPT to automate software testing—from generating unit tests to reporting results.It goes beyond just test generation—it also explains the purpose of each test , measures how much of the code is tested (coverage). Everything is summarized in a clear PDF report, making it easy for developers to understand what was tested and why. Four projects done in Python or Java were used for testing, and the results indicated high test coverage (up to 100%) and average execution times of 80–87 seconds. Python apps ran slightly faster, while Java had better coverage. Failures occurred mainly in Java due to vague prompts or compilation issues. Overall, the approach cuts down manual effort, but still has limits, like no support for integration tests. Thus, a lot of potential was seen towards quality assurance in actual projects with the help of large language models.

Article 9: Testing React Single Page Web Application using Automated Tools. This study examines and contrasts three automated testing tools for React-based single page applications (SPAs): Cypress, Enzyme, and Jest. The objective was to determine which tool performs best in terms of developer friendliness, ease of integration, and execution speed. The team tested all three on a real React app called TakeNote, using 20 test cases. Results showed that Enzyme had the fastest test execution, Jest was the easiest to integrate (as it comes preinstalled in React projects), and Cypress had the most user-friendly interface, but was slowest. Tests were run on different hardware setups to ensure fairness. The authors conclude that tool choice depends on team needs—Enzyme is fast, Jest is versatile, and Cypress is easy to write with but slower.

Article 10: Frontend Testing Improvement: A New Approach to Ensuring Web Application Quality. In order to address the persistent issues in frontend testing, such as trouble testing user interface elements, back-end integration, and debugging, this paper suggests a novel approach that draws inspiration from Netflix's proactive failure testing of the Molly algorithm. Instead of waiting for errors to happen, this method injects simulated failures during development to catch weak spots early. It helps developers test real user flows more effectively, even in complex apps. The approach is meant to reduce manual effort, improve reliability, and work well in CI/CD pipelines. The paper also covers how to integrate this into existing dev environments and highlights the need for training, feedback loops, and automation to make frontend testing more scalable and less frustrating.

Article 11: Validate Faster, Develop Smarter: A Review of Frontend Testing Best Practices and Frameworks. This study examines how best practices and contemporary frontend testing tools are changing to meet the increasing complexity of web applications. It focuses on popular frameworks like Jest, Cypress, and React Testing Library, explaining how they support unit, integration, and end-to-end testing. The author used a mix of methods—literature review, interviews with developers, surveys, and tool evaluations—to understand how these tools are used in real projects. Results show these frameworks improve testing speed, code quality, and team collaboration, but challenges like tool selection and maintaining complex tests still exist. The paper also explores how AI might help automate testing even more in the future, especially with visual testing and test case generation.

Article 12: AI-Generated Test Scripts for Web E2E Testing with ChatGPT and Copilot: A Preliminary Study. This preliminary study assesses whether AI code-generation tools can accelerate development of web end-to-end (E2E) test scripts. One junior tester wrote 96 Selenium-WebDriver/JUnit tests for eight diverse web applications using four strategies: fully manual coding, GitHub Copilot, a single-pass "ChatGPT Lite", and an iterative "ChatGPT Max". Effort was measured as weighted development time (seconds per Gherkin step). Manual creation was slowest (median 53.7 s/LOC). ChatGPT Max was fastest (38.4 s/LOC) and significantly outperformed manual and ChatGPT Lite; Copilot and ChatGPT Lite gave modest, non-significant gains. Statistical analysis used Wilcoxon tests with Bonferroni correction and Cliff's delta effect sizes. Results indicate that AI assistance reduces scripting time, but benefits depend on an experienced tester who can refine generated code and prompts. Threats include single-participant scope and reliance on time as the sole effort metric. The authors plan larger replications to evaluate generalisability and explore other quality dimensions. Overall, AI agents show promise for practical E2E test automation work.

Article 13: From User Stories to End-to-End Web Testing. This article outlines a methodical approach to converting user stories—plain language feature descriptions—into comprehensive E2E test cases for web applications. The method was built by reviewing existing literature and interviewing real-world testers. It includes how to choose which user stories to test, how to break them into basic and alternative flows, and how to design and implement automated test cases using tools like Selenium. The authors tested their method on three real web apps, including one for booking appointments, another for tracking data from remote devices, and a third for searching university staff info. Their approach is simple and practical, designed to fit fast-paced agile environments without relying on complex models or full automation. Their approach focuses on keeping things simple, practical, and well-suited for fast-paced agile environments, avoiding the overhead of complex models or fully automated systems.

Article 14: Using Multi-Locators to Increase the Robustness of Web Test Cases. The paper proposes "multi-locators", a novel way to harden Selenium-style web test cases against DOM changes. Instead of relying on a single XPath, each GUI element is given a set of alternative locators produced by five generation algorithms (FirePath absolute, FirePath relative-ID, Selenium IDE, Montoto and ROBULA+). During test execution a weighted voting scheme selects the element that receives the highest aggregate confidence; when the chosen element is found, any broken locators are automatically rebuilt, making the suite self-healing across releases. An empirical study on six open-source PHP web applications (675 elements, two successive releases each) shows that multi-locators cut the proportion of broken locators from 12% (best single algorithm) to 8%, approaching the theoretical optimum, while adding only 2–4% execution overhead. 93% of automated repairs were correct, and the approach requires no cross-validation if heuristic weights mirroring algorithm robustness are used in everyday regression suites.

Article 15: E2E Web Test Dependency Detection using NLP. This paper introduces TEDD, a tool designed to detect test dependencies in end-to-end (E2E) web testing. The problem it tackles is that many E2E tests aren't truly independent—they often rely on shared app states, which breaks things like test reordering and parallelization. TEDD also fixes missing links between tests to build a clear map showing which tests rely on which others. It validates and recovers missing dependencies to build a reliable test dependency graph. sTEDD was tested on six real-world web apps and achieved up to 70% faster validation than other methods, while enabling test suite parallelization with speeds of up to 7×. It is particularly helpful for cleaning up test suites that are slow or fragile.

The main aspects of these articles are exemplified in Figure 1.

4. Synthesis and Discussion

The fifteen primary studies converge around four high-level themes:

- **LLM / LVLM test generation (4 papers).** Works such as *AUTOE2E* and ChatGPT-assisted scripting

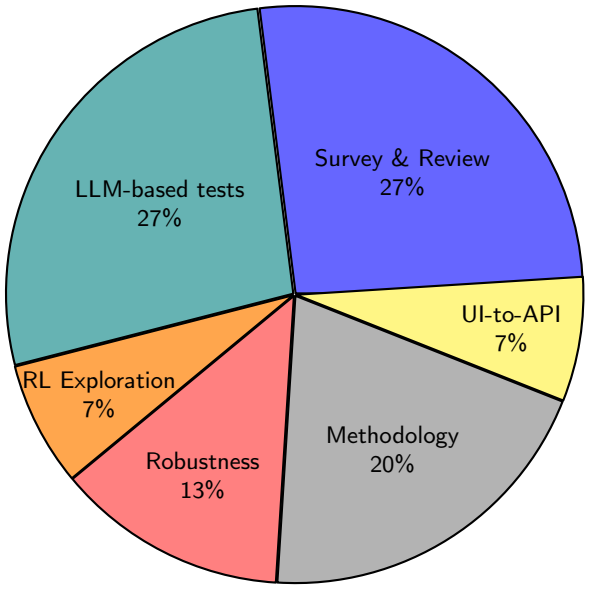

Figure 1: Distribution of research themes among the 15 reviewed articles

show that large language and vision–language models can craft longer, semantically richer end-to-end (E2E) flows than random crawlers, reducing authoring time by $\approx 30\,\%$.

- **Intelligent exploration (1 paper).** Curiosity-driven reinforcement learning (*WebExplor*) raises GUI coverage and fault discovery, though reproducibility and training costs remain open concerns.
- **Robustness engineering (2 papers).** Self-healing multi-locators and NLP-based dependency detection cut flaky-test failures by 30–70 % with negligible runtime overhead.
- **UI-to-API repurposing (1 paper).** *APICARV* shows that HTTP traces harvested from UI tests can be re-emitted as fast API suites while simultaneously inferring OpenAPI specifications (98 % precision).

The remaining seven papers deliver surveys, empirical tool comparisons, or methodological guidance that frame these technology-centric studies.

4.1. Methodological Patterns

Most evaluations use *failure-detection rate* and *code/DOM coverage* as effectiveness proxies, while *authoring time* and *execution time* capture efficiency. Only two studies include human-factor aspects (e.g. developer effort perception); none reports cost–benefit analyses across release cycles. Benchmark reuse is limited—*E2EBENCH*, a 50-site crawl corpus, and a handful of open-source apps dominate—raising threats to external validity.

4.2. Practical Implications

- **Adoption potential.** LLM assistants integrate with mainstream IDEs (e.g. Copilot), whereas RL explorers need bespoke infrastructure and GPU budgets.
- **Pipeline fit.** Robustness techniques are drop-in for existing Selenium workflows, offering immediate ROI in CI/CD environments plagued by flaky locators.
- **Shift-left testing.** UI-to-API carving enables suites that execute at least 10× faster, supporting earlier feedback cycles.

4.3. Research Gaps

1. Lack of **standardised, publicly hosted benchmarks** covering SPAs, micro-frontend architectures, and mobile-web hybrids.
2. Absence of **longitudinal robustness studies** tracking test health across many real releases.
3. Scarcity of **industrial replications**: fewer than 20 % of papers involve industry partners.
4. Limited **explainability**: current LLM pipelines act as black boxes that are hard to debug.

4.4. Future Directions

- *Hybrid agents* that combine LLM planning with RL exploration to unite semantic reasoning and efficient search.
- *Metric unification*, e.g. mapping feature coverage to business-value proxies, to enable cross-study comparison.
- *Cloud-native orchestration* with cost-aware resource scaling for enterprise adoption.
- *Human-in-the-loop frameworks* allowing testers to steer prompts or accept/reject self-healing patches, improving trust and accountability.

Overall, AI-assisted techniques promise significant gains in the efficiency and effectiveness of web-application testing, but their full impact depends on richer benchmarks, stronger empirical designs, and closer academia–industry collaboration.

5. Answers to Research Questions

Based on the analysis of the articles, we can provide the following answers to the research questions from the beginning of the paper:

1. **RQ1:** Modern web testing employs model-based or scripted Selenium suites, AI-generated end-to-end tests via LLMs/LVLMs, reinforcement-learning exploration, self-healing robustness techniques, and UI-to-API carving approaches.
2. **RQ2:** Tests are created through LLM prompting, LVLM form filling, or RL planners, then prioritised by feature coverage and adapted with self-healing locators and dependency-aware reordering.

3. **RQ3:** AI enables natural-language planning, vision-guided element selection, curiosity exploration, locator repair and UI-traffic specification mining, increasing coverage and efficiency yet lacking shared benchmarks and interpretability.
4. **RQ4:** Selenium, Cypress and Jest remain dominant, while AUTOE2E, WebExplor, VETL and APICARV emerge; evaluations still rely on failure or coverage metrics and mostly small open-source datasets.
5. **RQ5:** Robust, maintainable suites use multi-locator ensembles, NLP dependency graphs, UI-to-API carving, and LLM assistants to cut flaky tests, enable parallel runs, and reduce maintenance overhead.

6. Conclusions

This systematic review of fifteen web-application-testing papers published offers a clear picture of a rapidly evolving research area. Traditional script-based approaches remain the backbone of industry practice, yet the field is clearly pivoting towards *data-driven automation*: LLMs models now generate targeted feature end-to-end suites, reniforcement learning agents boost GUI exploration, and self-healing or UI-to-API techniques tackle long-standing maintenance aspects. Effectiveness is still judged mainly by failure-detection rate or code/DOM coverage, while efficiency is reported as authoring or execution time; only isolated work introduces newer metrics such as *feature coverage*.

Despite measurable gains—up to $\sim 30\%$ effort reduction and $10\times$ faster execution—the empirical landscape is fragmented: evaluations rely on small open-source apps, ad-hoc benchmarks and limited industrial involvement ($< 20\%$ of studies).

Open gaps and future work. *(i) Benchmarking.* Community-curated, publicly hosted datasets covering SPAs, micro-frontends and mobile-web hybrids are urgently needed to enable comparisons between same targeted-applications. *(ii) Longitudinal robustness.* Few studies track test health across many real releases; such evidence is critical for assessing self-healing and dependency-analysis claims. *(iii) Industrial validation.* Cost–benefit analyses and human-in-the-loop deployments would clarify adoption barriers for LLM and RL pipelines. *(iv) Explainability.* Black-box AI agents hinder debugging; interpretable prompt templates, plan visualisation and counter-example generation are promising remedies. *(v) Hybrid orchestration.* Combining semantic LLM planning with RL exploration under cloud-native schedulers may unlock both depth and breadth of coverage at scale.

By closing these gaps, future research can transform today's promising prototypes into robust, scalable and trustworthy tools that raise the quality bar for modern web applications.

References

Ali, K., Xiaoling, X., 2019. A reliabel and an efficient web testing system URL: https://arxiv.org/abs/1903.01221, arXiv:1903.01221.

Alian, P., Nashid, N., Shahbandeh, M., Shabani, T., Mesbah, A., 2025. Feature-driven end-to-end test generation URL: https://arxiv.org/abs/2408.01894, arXiv:2408.01894.

Biagiola, M., Stocco, A., Mesbah, A., Ricca, F., Tonella, P., 2019. Web test dependency detection, in: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Association for Computing Machinery, New York, NY, USA. p. 154–164. URL: https://doi.org/10.1145/3338906.3338948, doi:10.1145/3338906.3338948.

Devarapalli, C.A., 2019. Frontend testing improvement: A new approach to ensuring web application quality. International Journal of Science and Research (IJSR) 8, 2032–2037. URL: https://www.ijsr.net/getabstract.php?paperid=SR24401235433, doi:10.21275/SR24401235433.

Dobbala, M.K., 2022. Validate faster, develop smarter: A review of frontend testing best practices and frameworks. Journal of Mathematical Computer Applications , 1–4doi:10.47363/JMCA/2022(1)151.

García, B., Leotta, M., Ricca, F., Whitehead, J., Prasetya, W., Vos, T., Storm, T., 2024. Use of chatgpt as an assistant in the end-to-end test script generation for android apps, in: Proceedings of the 15th ACM International Workshop on Automating Test Case Design, Selection and Evaluation, ACM. pp. 5–11. doi:10.1145/3678719.3685691. online publication date: 13-Sep-2024.

Hasan, M.H., Rahman, M.A., Chowdhury, M.H., Rahman, M.H., Abdulle, K.H., Sadia, F., Hasan, M., 2022. Testing react single page web application using automated testing tools, in: Proceedings of the 17th International Conference on Evaluation of Novel Approaches to Software Engineering - ENASE, INSTICC. SciTePress. pp. 469–476. doi:10.5220/0011077900003176.

Kertusha, I., Assress, G., Duman, O., Arcuri, A., 2025. A survey on web testing: On the rise of ai and applications in industry URL: https://arxiv.org/abs/2503.05378, arXiv:2503.05378.

Leotta, M., Stocco, A., Ricca, F., Tonella, P., 2015. Using multi-locators to increase the robustness of web test cases, in: 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), pp. 1–10. doi:10.1109/ICST.2015.7102611.

Li, T., Huang, R., Cui, C., Towey, D., Ma, L., Li, Y.F., Xia, W., 2025. A survey on web application testing: A decade of evolution URL: https://arxiv.org/abs/2412.10476, arXiv:2412.10476.

Mollah, H., van den Bos, P., 2023. From user stories to end-to-end web testing, in: 2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 140–148. doi:10.1109/ICSTW58534.2023.00036.

Sherifi, B., Slhoub, K., Nembhard, F., 2024. The potential of llms in automating software testing: From generation to reporting URL: https://arxiv.org/abs/2501.00217, arXiv:2501.00217.

Wang, S., Wang, S., Fan, Y., Li, X., Liu, Y., 2024. Leveraging large vision language model for better automatic web gui testing URL: https://arxiv.org/abs/2410.12157, arXiv:2410.12157.

Yandrapally, R., Sinha, S., Tzoref-Brill, R., Mesbah, A., 2023. Carving ui tests to generate api tests and api specification URL: https://arxiv.org/abs/2305.14692, arXiv:2305.14692.

Zheng, Y., Liu, Y., Xie, X., Liu, Y., Ma, L., Hao, J., Liu, Y., 2021. Automatic web testing using curiosity-driven reinforcement learning URL: https://arxiv.org/abs/2103.06018, arXiv:2103.06018.