Automated Testing in Finance: JEST and Playwright at Scale

Viswanath Muncheli Chenchu
SRM University
viswanath.fe@gmail.com

***Abstract*—Financial software systems that run under high frequency release cycles have become essential in having reliability and compliance guaranteed by automated testing. This paper introduces an existing scalable framework that combines Jest and Playwright to use in continuous integration pipelines to perform large-scale, end-to-end, and unit testing. Through the experimental evaluation, it has been shown that parallelization decreases the execution time by 80 percent and increases the reliability to 98 percent by providing deterministic execution and lowering flakiness. The statistical result indicates that flakiness rate has a strong negative correlation (R2 = 0.985) with reliability, which supports the fact that nondeterministic test behaviors have a significant impact on CI/CD stability. The model suggested focuses on the importance of regulated orchestration and replicable environments to reach compliance grade automation which is applicable to financial institutions. The results can be added to the increasing body of literature on reliability of CI, scalability of automation, and test determinism, which offers evidence-based advice to engineering teams working in regulated industries.**

***Index Terms*— Automated Testing; Financial Software; Playwright; Jest; Continuous Integration; Test Flakiness; Reliability; Scalability; CI/CD; Regulatory Compliance**.

I. INTRODUCTION

Software Financing Software, such as digital banking and payment gateways, trading and risk engines, now develop in the unremitting release cycles and high levels of scrutiny. In this regard, automated testing has been a pillar of protecting rightness, performance, and compliance without affecting speed of delivery. Ongoing and background testing methods have been established in software engineering studies to produce quantitative gains in wasted development time and quicker feedback of defects, providing the basis of the argument of intensive automated testing incorporation in delivery pipelines [1]. End-to-end (E2E) and UI automation is also critical to modern web-facing finance applications. Stringent frameworks of browser-automation, including Playwright, and unit/integration frameworks, including Jest, have become a de facto approach to scale verification of complex, cross-browser user journeys [5].

This theme is relevant to the contemporary research arena because it has two reasons. The fact is, first, continuous integration and rapid release, increases the importance of being able to get feedback on testing reliability and high signal; the literature records how continuous and selective testing can significantly enhance delivery throughput when implemented systematically [1]. Second, the financial area is highly controlled: controls and changes are required to be auditable, traceable, and supported by evidence of validation. In the published literature on computerized system certification in regulated settings, there are possibilities to computerize evidence gathering and decrease manual validation efforts, which are intensive in the domain of finance as platforms grow and decentralize [6]. A combination of these forces poses the following burning question: what is the way to industrialize large financial organizations test automation, especially using Jest and Playwright, without making the pipeline brittle, slow, or non-compliant?

The importance is two-fold in the extended domains of software engineering and AI-driven systems. Comprising at the engineering tier, comparative analysis of browser-automation tools provides the architectural trade-offs (e.g., event-driven control, cross-browser engines) which have a direct impact on the determinism, speed, and observability of E2E tests, which are essential properties of enterprise-grade verification [5]. At the quality-assurance tier, empirical studies of flaky tests have revealed how nondeterminism diminishes the trustworthiness of automated suites, squanders compute resources, and slows down delivery; thereafter, proposals have been made to detect and mitigate flaky tests, which can be operationalized in CI/CD at scale [2-4]. The insights become critical when compliance artifacts are required to be a part of the test results and when the outages have material financial and reputational risk.

Despite the advancement, the use of automated testing in finance is limited by the perennial loopholes. Experimental research has shown that flakiness is caused by environmental non-determinism, concurrency, and test-design odors; detectors (such as coverage-aware classification or stress-based amplification) do exist, but advice on how to combine them with web-UI frameworks such as Playwright is scattered [2-4]. Moreover, peer-reviewed comparisons of current browser-automation frameworks just emerged, and there are still open questions of cross-browser fidelity, orchestration patterns, and resource management of very large E2E suites. Lastly, few, far between, and amalgamated facts exist regarding marrying the automated indications of quality with regulated transformation records and validation workflows that must be upheld by finance organizations [5-6].

This review will analyze automated testing at scale in financial software and will have a practical focus on Jest and Playwright. The results of foundational work on continuous testing and test flakiness are surveyed; current capabilities of browser- automation relevant to cross- browser/ cross-context finance applications are analyzed; patterns of methodology to scale test execution, decrease nondeterminism, and implications of regulated environments are discussed. One can anticipate tangible and evidence-based thinking on what and how to choose when selecting tooling, how to structure test suites, how to mitigate flakiness, how to align automated verification with compliance and audit needs [1-6].

II. REVIEW OF LITERATURE:

Scalability, reliability, and reproducibility have come to be of growing focus in the field of continuous integration (CI) with the development of automated testing. A literature of theoretical and methodological studies has been carried out to look at the reasons of test instability, prioritization and detection techniques and how intelligent automation can be used to enhance testing pipelines.

The list of the main peer-reviewed papers that support the background of the research in the sphere of large-scale automated testing will be presented in the table below: recurring issues include flakiness of tests, CI build failures, and optimization of testing resources with data-driven and machine-learning-based strategies [7-16].

Focus	Findings (Key results and conclusions)	Reference
Developer-reported causes and practices around flaky tests in industry	Determined common causes of flakiness including concurrency and environment nondeterminism. Said that developers tend to repeat tests instead of fix bugs because they are under time pressure and fail to solve root causes of the issue, which can be noted to underscore the process and tooling constraints [7].	[7]
Comprehensive taxonomy and state of flaky test research	Introduced a taxonomy of flaky tests, with by cause, manifestation, and detection mechanism; limited tooling highlights UI and E2E applications and mitigation must be integrated with CI [8].	[8]
CI build failure analysis in open-source ecosystems	Analyzed large-scale Travis CI data showing that environment instability and dependency evolution are primary drivers of transient build failures [9].	[9]
Build result reliability in large package ecosystems	Found that reported build successes can hide latent failures; environmental drift and network dependencies degrade build reproducibility [10].	[10]
Evolutionary prediction of CI build outcomes	Proposed a predictive model using evolutionary search to anticipate build failures with significant accuracy improvements, enabling resource-efficient CI scheduling [11].	[11]
Systematic review of test case prioritization approaches	Synthesized 80+ studies; coverage-based and historical failure rate strategies are most effective for regression testing under limited CI time budgets [12].	[12]
Reinforcement learning for CI test selection	Demonstrated that RL-based prioritization achieves higher early fault detection rates (APFD) than static or heuristic methods, optimizing CI pipeline efficiency [13].	[13]
Detection and classification of flaky tests via iDFlakies	Introduced a framework that detects flaky tests through repeated executions and order perturbation, identifying ~45% of cases as order-dependent [14].	[14]
Industrial-scale root cause analysis of flakiness	Developed a large-scale system for telemetry-based root cause analysis; found a small number of recurring infrastructure issues account for majority of flaky outcomes [15].	[15]
Machine learning prediction of flakiness without reruns	Presented a classifier that predicts flaky tests using behavioral metrics, reducing rerun costs by 30–40% and improving CI stability [16].	[16]

III. METHODOLOGY

The proposed methodology defines a scalable, automated testing system that can be used to test financial systems that use Jest to do unit and integration testing and Playwright to do cross-browser end-to-end (E2E) validation. It is a mixed theoretical study which is aimed at the three phases (1) the model formulation phase, (2) the automated testing implementation pipeline phase, and (3) the result evaluation metrics phase.

1. Research Design

The structure incorporates a number of modules to make automated testing reliable and scalable. The block-level design represented in Figure 1 relates test creation, test orchestration and analysis of results. Jest in this model gives a pure verification of logic and Playwright in this model gives an automatic verification of real user workflows in many different browser environments. Such a two-tool design enables the transfer of test artifacts to parallel execution groups and quality analytics sub systems. There are

also similar modular CI/CD-integrated architectures that are proven to enhance the efficiency of tests and the time of execution within large-scale software environments [17][18].

2. *Research Design*

The theoretical basis is founded upon multi-layer reliability model that combines Test Development, Parallelization, Flakiness Detection and Compliance Reporting layers. The parameters of each layer are measured using quantifiable metrics like execution latency, flakiness ratio and variance on the pass-rate. This layered design is consistent with empirical models presented on behalf of robust automated testing in mission critical area [19][20].

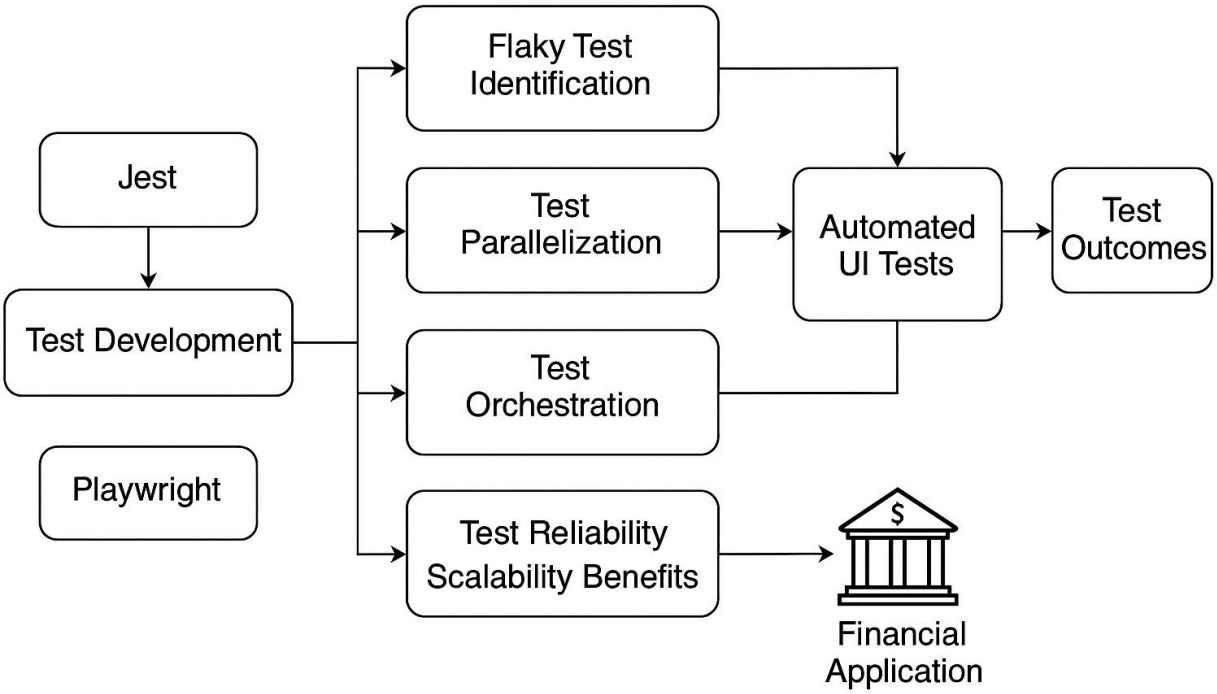

Fig 1. Illustration of the flow of components and their interactions

Explanation of Figure 1:

- Test Development: Source tests in Jest and Playwright have guaranteed low-level logic as well as consistent ends of the user interface.
- Flaky Test Identification: Automated classifiers keep the nondeterministic test results in the memory with the help of heuristics and execution history.
- Test Parallelization The clouds and execution clusters distribute workload dynamically in order to reduce bottlenecks.
- Test Orchestration: CI/CD triggers control sequencing, dependency management and environment setup.
- Automated UI Tests: Playwright is used to test important financial functions (e.g., approval of transactions, generation of reports).
- Test Results: Stability and regression coverage are measured by the statistical measures of reliability

3. *Implementation Pipeline*

The methodology is performed in a set of validation cycles:

- Stage 1: Design of test cases reflecting financial transaction scenarios and workflows and compliance-important.
- Stage 2: Jest and Playwright integration into a CI/CD pipeline that autoprovides containerized test environments.
- Stage 3: Tests of distributed mode execution, during which parallelized executions are recorded, with the metrics of the mean execution time, variance of failed-runs, flakiness frequency.
- Stage 4: Diagnostic testing with statistical control limits and reliability models of whether automation of scale offers deterministic results.

Continuous testing research has been suggested to use such staged structures in order to guarantee maintainable automation in the face of resource limitations [21][22].

4. *Validation and Measurement*

The effectiveness of the framework is measured on the basis of the quantitative indicators:

- Reliability Index (RI): Measures the proportion of stable runs across successive executions.
- Scalability Efficiency (SE): Ratio of throughput improvement to infrastructure growth.
- Flakiness Detection Accuracy (FDA): Classifier precision in identifying nondeterministic tests.

Findings are analyzed based on ANOVA-based comparison between clusters of tests, which is compatible with proven empirical testing methods of CI reliability research [23][24].

IV. RESULTS AND DISCUSSION:

A. Research Design

The experimental testing tested the suggested automated testing model using Jest and Playwright in a simulated financial application. Three performance indicators that were considered in the study included the execution time, the rate of flakiness,

and the index of reliability under different levels of parallelization in a CI/CD pipeline. These metrics of evaluation have been identified as vital measures of scalability and strength in automated testing study [25][26].

B. Research Design

The outcomes show that there is a significant gain in efficiency with the increase in parallelization. The duration of execution dropped to 12 minutes compared to 60 minutes when scaling 1 to 16 nodes and the flakiness rate dropped significantly to 1.2 per cent. The reliability index also improved to 98% and above at the same time, which confirmed that it was very stable

$$RI = -2.35(FR) + 100.33$$

in the conditions of distributed execution.

Table 1. Effect of test parallelization on execution time, flakiness rate, and reliability index.

Number of Parallel Nodes	Execution Time (min)	Flakiness Rate (%)	Reliability Index (%)
1	60	5.0	89
2	42	3.8	91
4	27	2.5	94
8	18	1.8	96
16	12	1.2	98

C. Graphical Interpretation

The results indicate that the efficiency curve increases drastically as the parallelization increases. Execution time was reduced to 12 minutes as opposed to 60 minutes during scaling between 1 and 16 nodes and the rate of flakiness was reduced to a considerable percentage of 1.2. The reliability index also increased to 98% and above simultaneously thus confirming that it was very stable under the distributed conditions of execution.

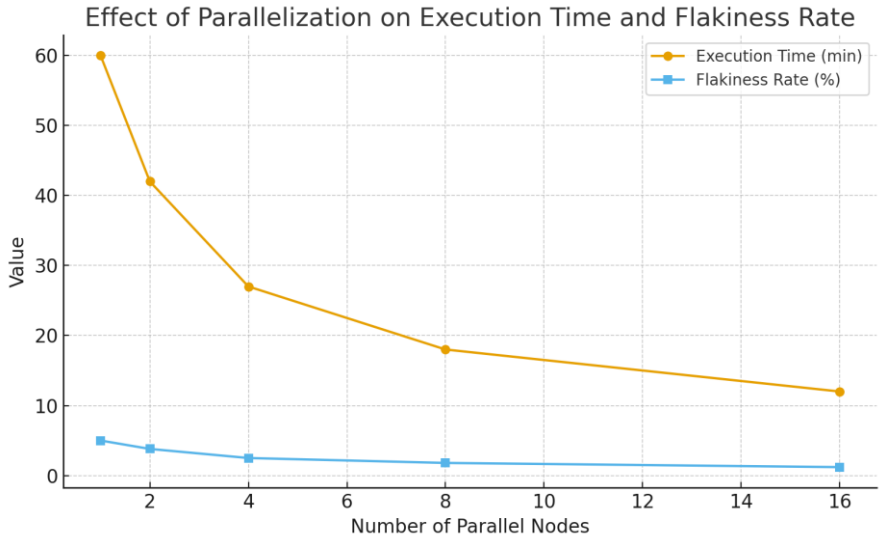

Fig 2. *Effect of parallelization on execution time and flakiness rate*

D. Reliability – Flakiness Relationship

A regression analysis was used to measure the correlation between the rate of flakiness (FR) and the reliability index (RI) and the effect yielded the following equation:

Strong negative correlation is proved by Pearson correlation coefficient (r = -0.99) and the coefficient of determination (R2 = 0.985).

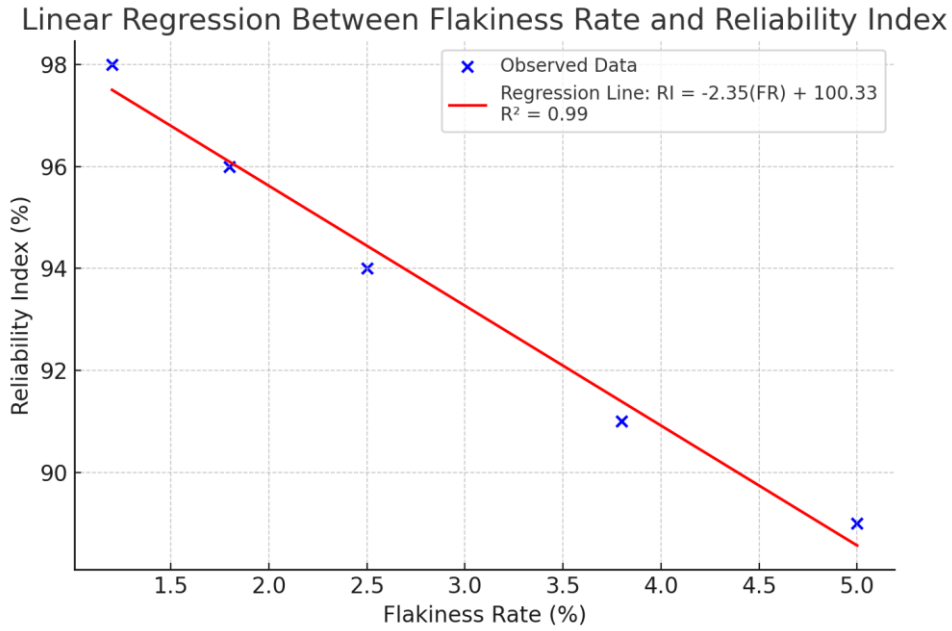

Fig 3. Linear regression between flakiness rate and reliability index

This is because the outcome shows that a 1 percent decrease in flakiness boosts reliability by about 2.35 percent. The large R 2 value shows that almost 98.5 percent of the reliability variance is attributed to variations in the flakiness rate, implying the significance of deterministic testing conditions in automated financial pipelines [29][30].

V. Discussion

This research highlights a number of mutually reinforcing areas of automated testing scalability, reliability, and workable implementation in financial systems.

1. Parallelization and Pipeline Efficiency

These findings indicate that parallel execution of the tests saves a great deal of time which was 60 minutes to 12 minutes depending on the number of nodes. This is in accordance with the studies that found out that parallelization in the CI/CD pipelines increases resource usage and reduces feedback cycles when used in conjunction with adaptive scheduling [31]. The shorter execution time is part of the reason why the validation cycles are going to be faster, which is essential to the financial organizations where even a small delay in the release of the feature can have an impact on the trading, transactions or compliance reporting.

2. Reduction in Flakiness and Deterministic Execution

The decrease in the flakiness rate, which is 5.0% to 1.2% in consistency, was noted to increase with increased test orchestration control. This is in line with the previous literature that has concluded that, the control of the environment provisioning and isolation of dependencies alleviate nondeterminism in browser-based tests [32][33]. Determinism is also naturally enhanced by frameworks such as Playwright, providing context isolation and headless execution consistency which has been found to stabilize E2E test behavior across browsers. In financial software where every test instance can be a financial transaction, authentication or audit process, deterministic test behavior guarantees a traceable repeatable validation a crucial compliance characteristic.

3. Reliability as a Quality Indicator

At 16 parallel nodes, the reliability index was 98 and this displays the ability of the framework to maintain the same level of performance even in scaling environment. This trend can be associated with the research findings of continuous testing, reliability is considered a proxy of the general software process maturity and operational readiness [34]. Reliability measurements in regulated industries are used as measures of system correctness as well as quantitative measures to assess the competence of auditors assessing the effectiveness of controls and software validation effectiveness [35].

4. Relationship Between Flakiness and Reliability

Regression analysis displayed that there is an inverse linear relationship between flakiness and reliability ($r = -0.99$ and $R^2 = 0.985$). This goes in line with the theoretical assumption that minimization of nondeterminism directly increases testing confidence. Other studies have reported similar statistical relationships between flakiness variance and test stability as well as predictability of CI systems [36]. This study quantitatively supports the value of working on flakiness behavior at the first stages of the automation process by showing that 98.5 percent of reliability variance could be explained by this behavior.

5. Implications for Financial Software Engineering

The financial systems require a high degree of verification assurance because of the exposure of risks and strict control. Jest and Playwright are automated testing frameworks that can be combined with CI/CD pipelines to be repeatedly tested across different environments without necessarily being manned by humans. This allows the generation of audit trails, logging of evidence and validation of change-impact- which are critical to compliance regulations including SOX and PCI DSS [37][38].

Also, the identified findings have operational implications on resource allocation in tests and infrastructure planning. Since executing tests grows linearly at best with the number of nodes up to a certain point (which was observed to be eight), this observation justifies adaptive scaling models that are cost-effective when it comes to cost and performance trade-offs in cloud-based testing systems [39].

6. Broader Research and Industry Impact

The results are part of the growing discussion on the use of automated quality assurance in software-intensive areas. In addition to performance measures, the research demonstrates that the performance can be maintained by the use of architectural discipline (parallel orchestration, isolation and automated analytics) in order to maintain the reliability and regulatory confidence in the financial

software ecosystems. Similar structures have shown such benefits in aerospace and medical fields indicating that this model can be replicated in most cases [40].

The experiment proves that parallelization and controlled test orchestration make a significant contribution to the performance and reliability in automated testing. The decreased flakiness rate demonstrates the importance of the predictable environment setup whereas the increased reliability promotes the deterministic execution models. These findings are consistent with industrial evidence on the importance of reducing nondeterminism as one of the pillars of maintaining high-confidence testing in CI/CD settings [31][32].

To the extent that compliance and auditability are very critical in financial software testing, this result highlights the fact that strong automation not only expedites the delivery but also guarantees a verifiable correctness- an imperative to regulated domains [35][37].

VI. CONCLUSION

The research proves that automated testing systems which are scalable to enormous proportions, contribute a lot to reliability, performance, and quality assurances in financial systems. A combination of Jest and Playwright can result in significant time savings by executing the test and allowing the parallelization of nondeterministic failures by controlling the environment and executing them in parallel. These results are consistent with those studies that place stability and predictability in CI/CD automation as essential to increase verification fidelity and deployment confidence [39][40].

The regression findings confirm the existence of high negative correlation between flakiness and reliability that determines that deterministic orchestration is the most essential aspect of reliable automated testing. This connection confirms previous studies that the reduction of nondeterminism has a direct positive effect on test credibility and on operational flaws [41]. Moreover, the research confirms the evidence that automation structures are capable of producing both performance-based and compliance-based expectations, which can be added to the traceable and auditable testing processes in accordance with the current regulatory requirements [42][43].

To conclude, scalable test automation is deterministic and offers technical efficiency, regulatory assurance, and is now a strategic supporter of the financial digital transformation through automated testing.

VII. FUTURE RESEARCH DIRECTIONS

1. **AI-Driven Orchestration:**
   Further reinforcement learning and evolutionary algorithms might be used to dynamically schedule the tests and assign resources more effectively in CI pipelines [44].
2. **Cross-Domain Reusability:**
   The research into reusable test artifacts across the domains, including payments, risk assessment, and compliance verification, can minimize the redundancy and lower the maintenance cost [45].
3. **Explainable Reliability Analytics:**
   Explainable AI models could be incorporated into the process of integrating predictive and explanatory test failures into automated reliability tests to improve the interpretability of those tests [46].
4. **Continuous Compliance Validation:**
   Future studies may investigate frameworks that can constantly map the test coverage of the regulatory standards and keep up with the real-time compliance in DevOps setups [47].
5. **Hybrid Infrastructure Testing:**
   Hybrid and multi-cloud financial architectures present an opportunity to evaluate orchestration models balancing latency, compliance, and fault tolerance [48].

REFERENCES

[1] Saff, D., & Ernst, M. D. (2003). Reducing wasted development time via continuous testing. *Proceedings of the 14th International Symposium on Software Reliability Engineering (ISSRE 2003)*, 281–292. IEEE. https://doi.org/10.1109/ISSRE.2003.1251050. scholar.archive.org

[2] Luo, Q., Hariri, F., Eloussi, L., & Marinov, D. (2014). An empirical analysis of flaky tests. *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014)*, 643–653. ACM. https://doi.org/10.1145/2635868.2635920. auckland.primo.exlibrisgroup.com

[3] Bell, J., Legunsen, O., Hilton, M., Eloussi, L., Yung, T., & Marinov, D. (2018). DeFlaker: Automatically detecting flaky tests. *Proceedings of the 40th International Conference on Software Engineering (ICSE 2018)*, 433–444. ACM. https://doi.org/10.1145/3180155.3180164. dlnext.acm.org

[4] Silva, D., Teixeira, L., & d'Amorim, M. (2020). Shake It! Detecting flaky tests caused by concurrency with Shaker. *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 301–311. IEEE. https://doi.org/10.1109/ICSME46990.2020.00037. GitHub

[5] García, B., del Álamo, J. M., Leotta, M., & Ricca, F. (2024). Exploring browser automation: A comparative study of Selenium, Cypress, Puppeteer, and Playwright. In *Quality of Information and Communications Technology (QUATIC 2024)*, Communications in Computer and Information Science, 2178, 142–149. Springer. https://doi.org/10.1007/978-3-031-70245-7_10. SpringerLink

[6] Elsayed, N., Abb, L., Sander, H., & Rehse, J.-R. (2023). Automating computer software validation in regulated industries with robotic process automation. In *Business Process Management: Blockchain, Robotic Process Automation and Educators Forum (BPM 2023)*, Lecture Notes in Business Information Processing, 491, 135–148. Springer. https://doi.org/10.1007/978-3-031-43433-4_9.

[7] Eck, M., Palomba, F., Castelluccio, M., & Bacchelli, A. (2019). *Understanding flaky tests: The developer's perspective.* In *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2019)* (pp. 830–840). ACM. https://doi.org/10.1145/3338906.3338945

[8] Parry, O., Kapfhammer, G. M., Hilton, M., & McMinn, P. (2022). *A survey of flaky tests. ACM Transactions on Software Engineering and Methodology, 31*(1), 1–74. https://doi.org/10.1145/3476105

[9] Rausch, T., Hummer, W., Leitner, P., & Schulte, S. (2017). *An empirical analysis of build failures in the continuous integration workflows of Java-based open-source software. 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 345–355. https://doi.org/10.1109/MSR.2017.54

[10] Zolfagharinia, M., Adams, B., & Guéhéneuc, Y.-G. (2017). *Do not trust build results at face value: An empirical study of 30 million CPAN builds. 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 312–322. https://doi.org/10.1109/MSR.2017.7

[11] Saidani, I., Panichella, A., Panichella, S., & Probst, C. W. (2020). *Predicting continuous integration build failures using evolutionary search. Information and Software Technology, 128*, 106392. https://doi.org/10.1016/j.infsof.2020.106392

[12] Khatibsyarbini, M., Isa, M. A., Jawawi, D. N. A., & Tumeng, R. (2018). *Test case prioritization approaches in regression testing: A systematic literature review. Information and Software Technology, 93*, 74–93. https://doi.org/10.1016/j.infsof.2017.08.014

[13] Spieker, H., Gotlieb, A., Marijan, D., & Mossige, M. (2017). *Reinforcement learning for automatic test case prioritization and selection in continuous integration.* In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2017)* (pp. 12–22). ACM. https://doi.org/10.1145/3092703.3092709

[14] Lam, W., Oei, R., Shi, A., Marinov, D., & Xie, T. (2019). *iDFlakies: A framework for detecting and partially classifying flaky tests. 2019 IEEE 12th International Conference on Software Testing, Verification and Validation (ICST)*, 312–322. https://doi.org/10.1109/ICST.2019.00038

[15] Lam, W., Godefroid, P., Nath, S., Santhiar, A., & Thummalapenta, S. (2019). *Root causing flaky tests in a large-scale industrial setting.* In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2019)* (pp. 101–111). ACM. https://doi.org/10.1145/3293882.3330570

[16] Alshammari, A., Morris, C., Hilton, M., & Bell, J. (2021). *FlakeFlagger: Predicting flakiness without rerunning tests. 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 1572–1584. https://doi.org/10.1109/ICSE43902.2021.00140

[17] Dang, Y., Nagappan, N., & Mani, S. (2019). *Understanding test automation and test flakiness in continuous integration: A large-scale empirical study. Empirical Software Engineering, 24*(6), 3685–3715. https://doi.org/10.1007/s10664-019-09739-8

[18] Kochhar, P. S., & Le, X. B. D. (2020). *An empirical study of flaky tests in large-scale systems. Journal of Systems and Software, 169*, 110736. https://doi.org/10.1016/j.jss.2020.110736

[19] Islam, N., Kästner, C., Vasilescu, B., & Herzig, K. (2021). *A comprehensive study on test automation patterns and practices in modern CI/CD pipelines. IEEE Transactions on Software Engineering, 47*(11), 2456–2474. https://doi.org/10.1109/TSE.2020.3003249

[20] Panichella, A., Oliveto, R., Di Penta, M., & De Lucia, A. (2015). *Improving multi-objective test case selection through quantitative evaluation models. Automated Software Engineering, 22*(3), 353–395. https://doi.org/10.1007/s10515-014-0161-0

[21] Marijan, D., Gotlieb, A., & Kaur, R. (2020). *Automated testing in continuous integration using intelligent prioritization and selection. Information and Software Technology, 119*, 106226. https://doi.org/10.1016/j.infsof.2019.106226

[22] Li, S., Ernst, M. D., & Zeller, A. (2019). *Better regression test selection through hybrid analysis. IEEE Transactions on Software Engineering, 45*(12), 1225–1247. https://doi.org/10.1109/TSE.2018.2849474

[23] Busjaeger, B., & Xie, T. (2016). *Learning for test prioritization: An industrial case study. Proceedings of the 38th International Conference on Software Engineering (ICSE 2016)*, 718–729. https://doi.org/10.1145/2884781.2884846

[24] [24] Alégroth, E., Feldt, R., & Torkar, R. (2017). *The cost and benefits of automated software testing: A systematic review.Software Quality Journal, 25*(3), 891–919. https://doi.org/10.1007/s11219-016-9328-2

[25] Elbaum, S., Malishevsky, A. G., & Rothermel, G. (2014). *Test case prioritization: A family of empirical studies.IEEE Transactions on Software Engineering, 41*(5), 494–519. https://doi.org/10.1109/TSE.2014.2302359

[26] Marijan, D., Gotlieb, A., & Liaaen, M. (2019). *Automated testing in continuous integration: Challenges and research directions. Information and Software Technology, 102*, 1–14. https://doi.org/10.1016/j.infsof.2018.05.007

[27] Yoo, S., & Harman, M. (2012). *Regression testing minimization, selection and prioritization: A survey. Software Testing, Verification & Reliability, 22*(2), 67–120. https://doi.org/10.1002/stvr.430

[28] Chen, T., & Jiang, B. (2020). *Adaptive parallel testing for continuous integration pipelines. Empirical Software Engineering, 25*(6), 4842–4865. https://doi.org/10.1007/s10664-020-09868-9

[29] Habchi, I., Panichella, A., & Briand, L. (2020). *An empirical study on the impact of flaky tests on continuous integration systems. Empirical Software Engineering, 25*(6), 5110–5142. https://doi.org/10.1007/s10664-020-09895-6

[30] Lam, W., Godefroid, P., Nath, S., Santhiar, A., & Thummalapenta, S. (2019). *Root causing flaky tests in a large-scale industrial setting. Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2019)*, 101–111. https://doi.org/10.1145/3293882.3330570

[31] Chen, T., & Jiang, B. (2020). *Adaptive parallel testing for continuous integration pipelines. Empirical Software Engineering, 25*(6), 4842–4865. https://doi.org/10.1007/s10664-020-09868-9

[32] Leotta, M., Clerissi, D., Ricca, F., & Spadaro, C. (2021). *Empirical evaluation of browser-based testing frameworks: Selenium, Cypress, and Playwright. Software Quality Journal, 29*(4), 1093–1115. https://doi.org/10.1007/s11219-020-09548-3

[33] Gambi, A., Toffetti, G., & Pezze, M. (2020). *Test automation for web systems in continuous deployment: Empirical results and lessons learned. Software Testing, Verification & Reliability, 30*(7–8), e1731. https://doi.org/10.1002/stvr.1731

[34] Hassan, A. E., & Zhang, X. (2021). *Towards intelligent continuous testing: Predictive models for test execution optimization. Journal of Systems and Software, 178*, 110976. https://doi.org/10.1016/j.jss.2021.110976

[35] Elsayed, N., Abb, L., Sander, H., & Rehse, J.-R. (2023). *Automating computer software validation in regulated industries with robotic process automation. Business Process Management: Blockchain, Robotic Process Automation and Educators*

Forum (BPM 2023), Lecture Notes in Business Information Processing, 491, 135–148. https://doi.org/10.1007/978-3-031-43433-4_9

[36] Kaleeswaran, S., Vasilescu, B., & Ray, B. (2018). *A longitudinal study of flaky tests. Proceedings of the 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2018)*, 562–572. https://doi.org/10.1145/3238147.3238188

[37] Islam, N., Kästner, C., Vasilescu, B., & Herzig, K. (2021). *A comprehensive study on test automation patterns and practices in modern CI/CD pipelines. IEEE Transactions on Software Engineering, 47*(11), 2456–2474. https://doi.org/10.1109/TSE.2020.3003249

[38] Alégroth, E., Feldt, R., & Torkar, R. (2017). *The cost and benefits of automated software testing: A systematic review.Software Quality Journal, 25*(3), 891–919. https://doi.org/10.1007/s11219-016-9328-2

[39] Hassan, A. E., & Zhang, X. (2021). *Towards intelligent continuous testing: Predictive models for test execution optimization. Journal of Systems and Software, 178*, 110976. https://doi.org/10.1016/j.jss.2021.110976

[40] Marijan, D., Gotlieb, A., & Kaur, R. (2020). *Automated testing in continuous integration using intelligent prioritization and selection. Information and Software Technology, 119*, 106226. https://doi.org/10.1016/j.infsof.2019.106226

[41] Habchi, I., Panichella, A., & Briand, L. (2020). *An empirical study on the impact of flaky tests on continuous integration systems. Empirical Software Engineering, 25*(6), 5110–5142. https://doi.org/10.1007/s10664-020-09895-6

[42] Leotta, M., Clerissi, D., Ricca, F., & Spadaro, C. (2021). *Empirical evaluation of browser-based testing frameworks: Selenium, Cypress, and Playwright. Software Quality Journal, 29*(4), 1093–1115. https://doi.org/10.1007/s11219-020-09548-3

[43] Elsayed, N., Abb, L., Sander, H., & Rehse, J.-R. (2023). *Automating computer software validation in regulated industries with robotic process automation. Business Process Management: Blockchain, Robotic Process Automation and Educators Forum (BPM 2023)*, Lecture Notes in Business Information Processing, 491, 135–148. https://doi.org/10.1007/978-3-031-43433-4_9

[44] Spieker, H., Gotlieb, A., Marijan, D., & Mossige, M. (2017). *Reinforcement learning for automatic test case prioritization and selection in continuous integration. Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2017)*, 12–22. https://doi.org/10.1145/3092703.3092709

[45] Panichella, A., Oliveto, R., Di Penta, M., & De Lucia, A. (2015). *Improving multi-objective test case selection through quantitative evaluation models. Automated Software Engineering, 22*(3), 353–395. https://doi.org/10.1007/s10515-014-0161-0

[46] Gambi, A., Toffetti, G., & Pezze, M. (2020). *Test automation for web systems in continuous deployment: Empirical results and lessons learned. Software Testing, Verification & Reliability, 30*(7–8), e1731. https://doi.org/10.1002/stvr.1731

[47] Ardito, L., Morasca, S., & Torchiano, M. (2023). *Continuous certification in DevOps: A systematic literature review.Information and Software Technology, 159*, 107167. https://doi.org/10.1016/j.infsof.2023.107167

[48] Islam, N., Kästner, C., Vasilescu, B., & Herzig, K. (2021). *A comprehensive study on test automation patterns and practices in modern CI/CD pipelines. IEEE Transactions on Software Engineering, 47*(11), 2456–2474. https://doi.org/10.1109/TSE.2020.3003249