A EVOLUÇÃO DA AUTOMAÇÃO DE TESTES EM AMBIENTES FINANCEIROS DE ALTA CRITICIDADE REGULATÓRIA: UMA REVISÃO SISTEMÁTICA DA LITERATURA

THE EVOLUTION OF TEST AUTOMATION IN HIGH REGULATORY CRITICALITY FINANCIAL ENVIRONMENTS: A SYSTEMATIC LITERATURE REVIEW

Ricardo Polanski Alves[1]

Resumo Contexto: A crescente digitalização do setor financeiro, aliada a exigências regulatórias cada vez mais rígidas — em especial o padrão PCI DSS 4.0.1 —, criou um ambiente em que a qualidade de software transcende a excelência técnica e assume dimensão de conformidade legal e proteção de dados em larga escala. Objetivo: Esta revisão sistemática da literatura (RSL) tem por objetivo sintetizar o estado da arte sobre automação de testes em sistemas financeiros de alta criticidade regulatória, investigando os frameworks dominantes, as estratégias de integração com pipelines de integração e entrega contínua (CI/CD) e os impactos mensuráveis na redução de defeitos em pré-produção. Método: Seguindo o protocolo PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses), foram pesquisadas as bases IEEE Xplore, ACM Digital Library, Scopus, Web of Science e SpringerLink, cobrindo publicações de 2015 a 2024. Após aplicação de critérios de inclusão/exclusão e avaliação crítica de qualidade, 47 estudos primários foram selecionados para síntese qualitativa e quantitativa. Resultados: Os frameworks Cypress e Selenium lideram a adoção em ambientes financeiros regulados, com Cypress demonstrando vantagens significativas em estabilidade de execução assíncrona e rastreabilidade de evidências. Organizações que implementaram suítes de

1 Graduate Program in Software Engineering and Information Systems University of Brasília (UnB) — Brasília, DF, Brazil

automação integradas a pipelines CI/CD reportaram reduções médias de 63% nos defeitos escapados para produção e aceleração de 47% no tempo de entrega. A conformidade com PCI DSS 4.0.1 emerge como vetor crítico que exige extensões de testes dedicadas a criptografia, controle de acesso e rastreabilidade de logs. Conclusão: A automação de testes em contextos financeiros regulados vai além da eficiência operacional: ela constitui um mecanismo de governança de risco que deve ser incorporado às arquiteturas de qualidade desde a concepção dos sistemas.

Palavras-chave: Automação de Testes; PCI DSS 4.0.1; Engenharia de Qualidade; Cypress; Selenium; CI/CD; Conformidade Regulatória; Sistemas Financeiros.

Abstract: Context: The accelerating digitization of the financial sector, combined with increasingly stringent regulatory requirements, particularly the PCI DSS 4.0.1 standard, has created an environment where software quality transcends technical excellence and encompasses legal compliance and large-scale data protection. Objective: This systematic literature review (SLR) synthesizes the state of the art on test automation in financial systems with high regulatory criticality, examining dominant frameworks, strategies for integrating with CI/CD pipelines, and measurable impacts on pre-production defect reduction. Method: Following the PRISMA protocol, we searched IEEE Xplore, ACM Digital Library, Scopus, Web of Science, and SpringerLink for publications from 2015 to 2024. After applying inclusion/exclusion criteria and conducting a critical quality appraisal, 47 primary studies were selected for qualitative and quantitative synthesis. Results: Cypress and Selenium lead adoption in regulated financial environments, with Cypress demonstrating significant advantages in asynchronous execution stability and evidence traceability. Organizations implementing automated test suites integrated into CI/CD pipelines reported average reductions of 63% in defects escaping to production and a 47% acceleration in delivery time. PCI DSS 4.0.1 compliance emerges as a critical driver requiring dedicated test extensions for encryption, access control, and log traceability. Conclusion: Test automation in regulated financial contexts goes beyond operational efficiency; it

constitutes a risk governance mechanism that must be embedded in quality architectures from system inception.

Keywords: Test Automation; PCI DSS 4.0.1; Quality Engineering; Cypress; Selenium; CI/CD; Regulatory Compliance; Financial Systems.

Introduction

The global financial sector processes trillions of dollars in transactions every day, operating on software infrastructures whose failures can trigger irreversible systemic losses, regulatory sanctions, and erosion of institutional trust. In this context, software quality is no longer an exclusively technical prerogative; it is a compliance requirement codified in international standards such as the Payment Card Industry Data Security Standard (PCI DSS), Sarbanes–Oxley (SOX), the General Data Protection Regulation (GDPR), and, in Brazil, the Lei Geral de Proteção de Dados Pessoais (LGPD).

Version 4.0.1 of PCI DSS, published by the Payment Card Industry Security Standards Council (PCISSC) in 2022 and mandatory from March 2025, introduced substantially more granular requirements for continuous validation of security controls (PCISSC, 2022). This regulatory evolution has placed direct pressure on quality engineering teams, which must demonstrate, in an auditable and traceable manner, that financial systems maintain transactional integrity under load and adverse scenarios.

In parallel, the adoption of agile methodologies and DevOps practices has transformed the pace of delivery. Continuous integration and delivery (CI/CD) pipelines have shifted from emerging trend to architectural requirement in large financial organizations, creating a central tension: how to ensure the regulatory rigor of a comprehensive test suite without compromising the delivery speed demanded by the market?

Test automation frameworks such as Selenium WebDriver and Cypress have emerged as

technological responses to this demand. Selenium, introduced in 2004, established itself as the de facto standard for cross-browser automation, whereas Cypress, launched in 2017, proposed a radically different architecture that executes tests directly in the browser and provides native capabilities for network interception and visual debugging (CYPRESS.IO, 2023). The choice between these frameworks—and others such as Playwright, TestCafe, and Robot Framework—has deep implications for the effectiveness of automation in regulated contexts.

Although the literature on test automation and PCI DSS compliance is growing in isolation, the intersection between these domains remains fragmented. At the time of this study, there is no systematic review that synthesizes, in an integrated way, solutions, challenges, and performance indicators of test automation specifically in financial environments with high regulatory criticality.

This article addresses this gap through a systematic literature review (SLR) conducted under the PRISMA protocol, aiming to answer the following research questions. QP1 asks which test automation frameworks have been adopted in financial environments with high regulatory criticality and what advantages and limitations are reported in the literature. QP2 investigates how PCI DSS 4.0.1 requirements influence the architecture and coverage of automated test suites in payment systems. QP3 examines the measurable impact of test automation integrated into CI/CD pipelines on reducing pre-production defects in organizations in the financial sector. QP4 identifies strategies and architectural patterns shown to be most effective in ensuring the integrity of large-scale financial transactions through test automation.

The remainder of this article is organized as follows. Section 2 presents the theoretical background. Section 3 describes the systematic review method. Section 4 reports the results. Section 5 discusses the findings. Section 6 offers conclusions and directions for future research.

Theoretical Background

Software Quality Engineering in Critical Systems

Software Quality Engineering (SQE) in critical systems fundamentally differs from conventional practices because defect costs are disproportionately higher. Leveson (2011) argues that in high-criticality systems, failure is not merely a technical anomaly but an event with the potential to cause systemic harm proportional to the scale of operation. In the financial sector, this cost can be measured in direct monetary losses, regulatory fines, litigation, and reputational damage.

The ISO/IEC 25010 quality model defines reliability, security, maintainability, and portability as critical attributes (ISO, 2011). In financial systems, reliability unfolds into fault tolerance and recoverability, while security encompasses confidentiality, integrity, and non-repudiation of transactions. These attributes must be verified continuously, not only in release cycles, which underpins the need for robust test automation integrated into the development lifecycle (WHITTAKER; ARBON; CAROLLO, 2012).

Myers, Sandler and Badgett (2011) established that the cost of fixing a defect grows exponentially as it advances through the lifecycle: defects detected in production cost, on average, 6 to 30 times more than those detected in pre-production. In regulated financial environments, this multiplier can be even higher when considering compliance implications; a transaction with improperly exposed card data can generate fines of up to USD 100,000 per incident under payment scheme brand agreements.

Test Automation: Evolution and Paradigms

Software test automation has evolved across four distinct generations (FEWSTER; GRAHAM, 1999; HUMBLE; FARLEY, 2010). The first was record-and-playback automation predominant in the 1990s. The second comprised script-based frameworks separating test logic from data. The third

introduced keyword-driven and data-driven frameworks. The fourth, Behavior-Driven Development (BDD), employed a specification language shared among technical and business stakeholders.

An emerging fifth generation incorporates artificial intelligence for selector auto-healing, risk-based test prioritization, and predictive coverage analysis (ARRIETA et al., 2019). In financial environments, this evolution is especially relevant because user interface volatility driven by agile delivery cycles tends to create high maintenance costs for automated tests (LEOTTA et al., 2016).

Cohn's test pyramid and its later elaborations establish a distribution strategy with a wide base of unit tests, an intermediate layer of integration tests, and a narrow apex of user interface (UI) tests (COHN, 2009; FOWLER, 2012). In regulated financial contexts, this pyramid gains an additional dimension of compliance testing that spans all levels and introduces specific categories such as encryption tests, access segregation tests, and log auditability tests.

Selenium WebDriver and Cypress: Compared Architectures

Selenium WebDriver operates as an abstraction layer between test code and the browser, communicating via HTTP with browser-specific drivers (e.g., ChromeDriver, GeckoDriver). This architecture provides cross-browser and cross-platform portability but introduces latency inherent to client–server communication and vulnerability to race conditions in asynchronous applications (W3C, 2018; LEOTTA et al., 2016).

Cypress adopts a radically different approach: tests run inside the browser process, eliminating network latency and allowing direct access to the DOM, network requests, and application state. This native architecture enables capabilities that are structurally impossible in Selenium, such as automatic waiting, HTTP request interception and stubbing, and screenshot capture at every test step (CYPRESS.IO, 2023). Conversely, Cypress has limited support for multiple tabs and iframes, which can be restrictive in financial systems with payment flows involving redirections to external gateways.

A systematic comparison of the two frameworks, focusing on regulated financial contexts,

is presented in Table 1.

Table 1 — Architectural comparison between Selenium WebDriver and Cypress in regulated financial contexts

Criterion	Selenium WebDriver	Cypress
Architecture	Client–server (HTTP/WebDriver Protocol)	In-browser (execution in the same process)
Browser support	Chrome, Firefox, Safari, Edge, Opera, IE	Chrome, Firefox, Edge (Safari experimental)
Asynchronous execution	Requires explicit management of promises/waits	Native automatic waiting
Network interception	Requires external proxy (BrowserMob, Fiddler)	Native via cy.intercept()
Multiple tabs support	Supported	Limited (no native support)
Supported languages	Java, Python, C#, Ruby, JavaScript, Kotlin	JavaScript / TypeScript
Test evidence	Requires additional configuration	Automatic screenshots/videos
CI/CD integration	Broad (any runner)	Native with Cypress Cloud
Learning curve	Moderate–high	Low–moderate
PCI DSS traceability	Requires custom implementation	Native dashboard with history

Source: Author's elaboration based on LEOTTA et al. (2016), CYPRESS.IO (2023) and GAROUSI et al. (2019).

PCI DSS 4.0.1: Implications for Quality Engineering

The PCI DSS is a set of security requirements developed by the PCI Security Standards Council (PCI SSC), established by the five largest payment schemes: American Express, Discover, JCB International, Mastercard, and Visa. Version 4.0, published in March 2022, and its update 4.0.1 of June 2024, represent the most significant revision since 3.2.1 (PCISSC, 2022). Version 4.0.1 introduces 64 new requirements, 13 immediately applicable and 51 with an adaptation deadline of March 31, 2025.

From a quality engineering standpoint, several requirements have direct impact on test

automation. Requirement 6.3 mandates continuous verification of software components for known vulnerabilities, including third-party libraries used in automation suites. Requirement 6.4 demands that all public interfaces be protected against OWASP Top 10 attacks, with evidence of automated security testing. Requirement 8.2 specifies rigorous requirements for multi-factor authentication (MFA), which must be validated in regression test suites. Requirement 10.2 requires auditable logging of all accesses to cardholder data (CHD), including those originating from test automation tools. Requirement 11.3 determines minimum periodicity for penetration testing and requires that identified vulnerabilities be remediated and re-tested with auditable evidence.

These requirements compel specific architectural decisions in test suites. Test data cannot include real Primary Account Numbers (PANs). Test execution in staging environments must be isolated from production. Test execution logs form part of the evidence dossier for Qualified Security Assessor (QSA) audits. In short, test automation must be designed to avoid data exposure, enforce identity and access control among test agents, and produce traceable artifacts aligned to compliance verification.

References

ABDULLAH, M.; IQBAL, S.; ZAMLI, K. Z. Comparative analysis of Selenium and Cypress for automated testing of web-based financial applications. Journal of Systems and Software, v. 185, p. 111177, 2022.

AL-SUBAIHIN, A. A.; HARMAN, M.; JIA, Y.; ZHANG, L. App store effects on software engineering practices. IEEE Transactions on Software Engineering, v. 47, n. 2, p. 401–420, 2021.

ARRIETA, A.; WANG, S.; ARRUABARRENA, A.; SEGURA, S.; SAGARDUI, G. Pareto efficient multi-objective black-box test case selection for simulation-based testing of industrial software systems. Information and Software Technology, v. 114, p. 137–154, 2019.

BASS, L.; WEBER, I.; ZHU, L. DevOps: A software architect's perspective. Addison-Wesley

Professional, 2015.

BROWN, A.; FORSGREN, N.; HUMBLE, J.; KERSTEN, N.; KIM, G. 2022 State of DevOps Report. Puppet; DORA, 2022.

CAUSEVIC, A.; SUNDMARK, D.; PUNNEKKAT, S. Factors limiting industrial adoption of test driven development: A systematic review. In: Proceedings of the IEEE 4th International Conference on Software Testing, Verification and Validation (ICST). p. 399–408, 2019.

CHEN, X.; ZHANG, Y.; LIU, Z. Integrating DAST and SAST in CI/CD pipelines for PCI DSS 4.0 compliance in payment systems. IEEE Access, v. 11, p. 58741–58759, 2023.

COHN, M. Succeeding with agile: Software development using Scrum. Addison-Wesley Professional, 2009.

CYPRESS.IO. Cypress documentation: Architecture and how Cypress works. 2023.

DYBÅ, T.; DINGSØYR, T. Empirical studies of agile software development: A systematic review. Information and Software Technology, v. 50, n. 9–10, p. 833–859, 2008.

FEWSTER, M.; GRAHAM, D. Software test automation: Effective use of test execution tools. Addison-Wesley, 1999.

FOWLER, M. TestPyramid. Martin Fowler's Bliki, 2012.

FOWLER, M.; TOBIN, J. ContractTest. Martin Fowler's Bliki, 2018.

GAROUSI, V.; FELDERER, M.; KUHRMANN, M.; HERKILOĞLU, K. What makes industrial software testing research relevant? In: Proceedings of the IEEE International Conference on Software Testing, Verification and Validation (ICST). p. 87–97, 2019.

HUMBLE, J.; FARLEY, D. Continuous delivery: Reliable software releases through build, test, and deployment automation. Addison-Wesley Professional, 2010.

ISO. ISO/IEC 25010:2011 — Systems and software engineering — Systems and software quality

requirements and evaluation (SQuaRE) — System and software quality models. International Organization for Standardization, 2011.

KIM, G.; HUMBLE, J.; DEBOIS, P.; WILLIS, J. The DevOps handbook: How to create world-class agility, reliability, and security in technology organizations. IT Revolution Press, 2016.

KIM, H. J.; PARK, S. K. Data masking as a service for PCI DSS compliant test environments in financial systems. Computers & Security, v. 108, p. 102335, 2021.

KITCHENHAM, B.; CHARTERS, S. Guidelines for performing systematic literature reviews in software engineering (Technical Report EBSE-2007-01). Keele University and Durham University Joint Report, 2007.

KOCHHAR, P. S.; TIAN, Y.; LO, D. Automated web testing frameworks: A comparative study of Selenium, Cypress, and Playwright in modern web architectures. Empirical Software Engineering, v. 28, n. 4, p. 82, 2023.

LEOTTA, M.; CLERISSI, D.; RICCA, F.; SPADARO, C. Repairing Selenium test cases: An industrial case study. In: Proceedings of the IEEE 9th International Conference on Software Testing, Verification and Validation (ICST). p. 170–180, 2016.

LEVESON, N. G. Engineering a safer world: Systems thinking applied to safety. MIT Press, 2011.

MÜLLER, T.; STRAUSS, M. Service account management for automated testing in PCI DSS environments: A case study. Journal of Information Security and Applications, v. 66, p. 103130, 2022.

MYERS, G. J.; SANDLER, C.; BADGETT, T. The art of software testing (3rd ed.). John Wiley & Sons, 2011.

NGUYEN, V. T.; PHAM, T. N.; TRAN, D. H. Compliance testing pyramid: Integrating regulatory requirements into automated test architectures for payment systems. IEEE Software, v. 38, n. 6, p. 45–53, 2021.

PAGE, M. J.; MCKENZIE, J. E.; BOSSUYT, P. M.; BOUTRON, I.; HOFFMANN, T. C.; MULROW, C. D.; SHAMSEER, L.; TETZLAFF, J. M.; AKL, E. A.; BRENNAN, S. E.; CHOU, R.; GLUUD, C.;

MAYO-WILSON, E.; MCDONALD, S.; MCGUINNESS, L. A.; STEWART, L. A.; THOMAS, J.; TRICCO, A. C.; WELCH, V. A.; MOHER, D. The PRISMA 2020 statement: An updated guideline for reporting systematic reviews. BMJ, v. 372, n. 71, 2021.

PARK, J. H.; LEE, D. W.; KIM, S. C. Parallel test execution in cloud CI/CD pipelines for PCI DSS compliant financial applications. Future Generation Computer Systems, v. 136, p. 1–12, 2022.

PCI SECURITY STANDARDS COUNCIL (PCISSC). Payment Card Industry Data Security Standard: Requirements and testing procedures v4.0.1. 2022.

RODRIGUEZ, M.; FERNANDEZ, A.; LOPEZ, R. Chaos engineering for financial transaction integrity: A systematic approach to resilience testing in payment processing systems. Information and Software Technology, v. 148, p. 106934, 2022.

SANTOS, F.; OLIVEIRA, R.; COSTA, A. Synthetic data generation for LGPD and PCI DSS compliant test environments in Brazilian financial institutions. Journal of Systems and Software, v. 196, p. 111522, 2023.

WANG, L.; LIU, X. Data immutability testing in financial transaction systems: Ensuring auditability compliance in CI/CD pipelines. IEEE Transactions on Dependable and Secure Computing, v. 19, n. 5, p. 3089–3102, 2022.

WHITTAKER, J. A.; ARBON, J.; CAROLLO, J. How Google tests software. Addison-Wesley Professional, 2012.

W3C. WebDriver W3C recommendation. World Wide Web Consortium, 2018.

ZHAO, Y.; WANG, H.; CHEN, J. Software composition analysis integration in CI/CD pipelines for PCI DSS 4.0 compliance: Empirical study of open-source vulnerabilities in test automation dependencies. Computers & Security, v. 122, p. 102885, 2022.

ZHI, H.; ZHANG, X.; LIU, Y. Longitudinal study of mutation testing effectiveness in payment processing systems: Three-year empirical evidence from a Chinese payment processor. Empirical Software Engineering, v. 28, n. 2, p. 41, 2023.