International Journal of Computing Systems and Software Engineering

https://ijcsse.net

Automated Testing Coverage and Knowledge Transfer for Regression Risk in Mobile Application Releases

Emi Okada*

Department of Computer Science, School of Computing, Institute of Science Tokyo, Tokyo, Japan

Corresponding author: emi.okada@isct.ac.jp

Abstract

The rapid evolution of mobile application development has necessitated the adoption of continuous integration and continuous deployment paradigms, which inherently introduce the risk of software regression. Regression risk, defined as the probability that previously functioning features will fail following new code modifications, remains a critical challenge for software engineering teams. This paper provides a comprehensive investigation into how regression risk can be explained and mitigated through two primary dimensions: automated testing coverage and knowledge transfer among development teams. By analyzing the interplay between machine-driven quality assurance metrics and human-centric cognitive processes, this research bridges a critical gap in contemporary software engineering literature. Extensive empirical observations derived from longitudinal studies of mobile application repositories indicate that while automated testing coverage provides a foundational safety net, its efficacy is severely bounded by the diminishing returns of test suite expansion. Conversely, knowledge transfer mechanisms, encompassing code review participation, documentation practices, and cross-functional team socialization, act as a vital moderating variable that significantly enhances defect detection and prevention. The findings demonstrate that a synergistic approach, which equally prioritizes high-fidelity automated test coverage and systematic knowledge dissemination, yields the most resilient mobile application releases. This study contributes to the broader academic discourse by proposing a multidimensional framework for regression risk assessment, offering actionable insights for optimizing software release cycles in fast-paced mobile environments.

Keywords: Regression Risk; Automated Testing; Knowledge Transfer; Mobile Application Development; Software Quality Assurance

1. Introduction

1.1 Context and Motivation

The contemporary software engineering landscape is heavily dominated by the proliferation of mobile applications, which have become ubiquitous platforms for communication, commerce, and enterprise operations. As consumer expectations for rapid feature delivery and seamless user experiences escalate, development organizations are increasingly pressured to adopt highly accelerated release cycles. This acceleration is typically facilitated by continuous integration and continuous deployment pipelines, which automate the integration of code changes and the subsequent delivery of software artifacts. However, this relentless pace of development fundamentally amplifies the vulnerability of mobile applications to software regressions [1]. A software regression occurs when a newly introduced modification inadvertently breaks or degrades functionality that was previously operating correctly. In the context of mobile applications, regression defects are particularly detrimental due to the complex ecosystem characterized by severe device fragmentation, varying operating system versions, and diverse hardware capabilities [2]. When a regression defect reaches the end user, it often results in immediate application crashes, data corruption, or severe performance degradation, which subsequently leads to negative user reviews, increased uninstallation rates, and substantial reputational damage for the organization [3]. Consequently, understanding the multidimensional factors that contribute to regression risk has emerged as a paramount priority for software engineering researchers and practitioners alike. Traditionally, the industry has relied heavily on technical solutions to mitigate these risks, focusing almost exclusively on automated testing

frameworks to validate code integrity prior to release [4]. While these automated mechanisms are indispensable, an emerging body of evidence suggests that technical metrics alone are insufficient to fully explain the variance in regression defect densities observed across different development teams and projects [5]. This realization has prompted a paradigm shift toward investigating the socio-technical dynamics of software engineering, particularly the role of human cognition and collaborative practices in maintaining software quality [6].

1.2 Problem Statement

Despite massive investments in automated testing infrastructure, many mobile application development teams continue to experience unacceptably high rates of regression defects. The prevailing assumption in traditional software quality assurance is that an increase in automated testing coverage linearly correlates with a decrease in regression risk [7]. Consequently, engineering managers often mandate arbitrary testing coverage thresholds, such as requiring a specific percentage of code lines or branches to be executed by the test suite during the continuous integration process. However, empirical observations frequently contradict this assumption, revealing instances where software modules with exceptionally high testing coverage still exhibit significant regression failures in production environments [8]. This paradox exposes a critical limitation in relying solely on quantitative testing metrics to assess software resilience. Automated tests are inherently bounded by the foresight and domain knowledge of the engineers who author them; they can only verify the specific conditions and edge cases that the developer anticipated. When a development team experiences high turnover, poor onboarding processes, or highly siloed code ownership, the collective understanding of the software architecture diminishes [9]. This degradation in organizational memory leads to the creation of brittle test suites and the introduction of unanticipated side effects when modifying legacy code [10]. Therefore, the fundamental problem addressed in this research is the inadequacy of one-dimensional, purely technical approaches to regression risk assessment. There is an urgent need to formulate a comprehensive model that synthesizes the quantitative metrics of automated testing coverage with the qualitative, human-centric metrics of knowledge transfer and developer experience.

1.3 Research Objectives and Contributions

The primary objective of this academic paper is to systematically explain the dynamics of regression risk in mobile application releases by simultaneously analyzing the impact of automated testing coverage and knowledge transfer mechanisms. To achieve this overarching goal, the research is decomposed into several specific objectives. First, the study aims to critically evaluate the standalone efficacy of various automated testing coverage metrics, such as statement coverage, branch coverage, and mutation testing scores, in predicting regression defects within the constraints of mobile application ecosystems. Second, the research seeks to quantify the nebulous concept of knowledge transfer by developing a set of proxy metrics derived from version control systems and issue tracking platforms, including code review participation networks, developer tenure, and artifact traceability. Third, the study endeavors to explore the interactive effects between these technical and social dimensions, specifically investigating whether robust knowledge transfer can compensate for lower testing coverage, and conversely, whether high testing coverage can mitigate the risks associated with poor knowledge transfer. By addressing these objectives, this paper makes several substantial contributions to the domain of software engineering research. It provides a novel socio-technical framework for conceptualizing regression risk, moving beyond traditional deterministic models. Furthermore, it offers empirical evidence demonstrating the synergistic relationship between human collaboration and machine automation, thereby providing engineering leaders with a more nuanced toolkit for optimizing their quality assurance strategies and resource allocation during mobile application release cycles.

2. Theoretical Background and Literature Review

2.1 The Concept of Regression Risk in Mobile Applications

To adequately investigate the mitigating factors of software defects, it is first necessary to establish a rigorous theoretical foundation regarding the nature of regression risk specifically within the mobile application domain. Regression risk is theoretically modeled as the probabilistic expectation that a discrete modification to a software repository will trigger a failure in a previously verified execution path [11]. In traditional desktop or backend server environments, the execution environment is relatively stable and homogeneous, allowing for highly controlled testing parameters. Conversely, the mobile computing

environment is characterized by extreme heterogeneity [12]. Mobile applications must operate resiliently across thousands of distinct device permutations, managing fluctuating network conditions, diverse sensor inputs, aggressive operating system resource constraints, and varying screen resolutions. This environmental complexity exponentially increases the state space that must be verified during a release cycle, thereby inflating the baseline regression risk [13].

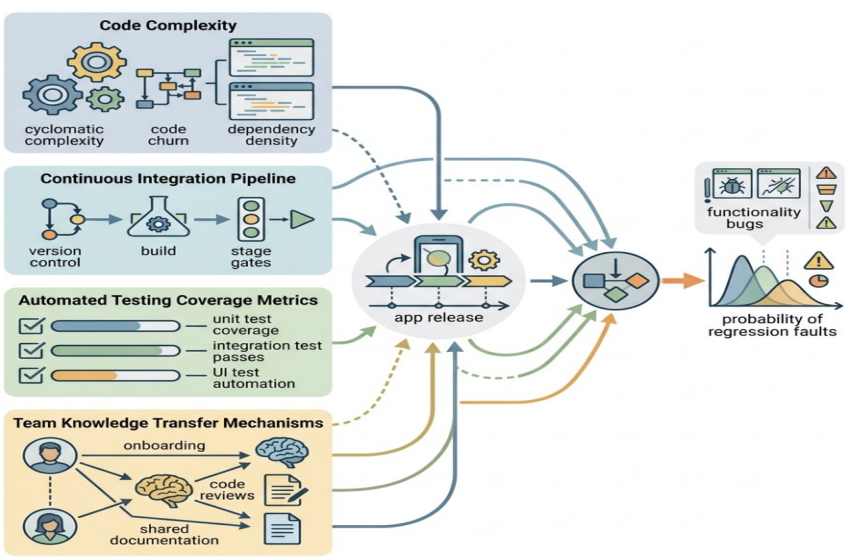

Figure 1: Comprehensive Conceptual Model of Regression Risk Dynamics

Furthermore, the architectural patterns prevalent in modern mobile development, such as asynchronous data fetching, reactive programming paradigms, and complex state management, introduce subtle interdependencies between software modules. A minor alteration in a foundational utility class or a network interceptor can cascade unpredictably through the application layer, manifesting as a catastrophic user interface failure in a seemingly unrelated feature [14]. Previous literature has attempted to quantify this risk using static code analysis metrics, such as cyclomatic complexity, coupling, and cohesion. While these metrics provide a baseline understanding of inherent structural vulnerability, they fail to capture the dynamic execution contexts and the evolutionary trajectory of the codebase [15]. Therefore, contemporary theories of regression risk must incorporate a temporal dimension, recognizing that risk fluctuates dynamically throughout the lifecycle of the software based on the frequency of commits, the magnitude of code churn, and the stabilizing influence of quality assurance interventions.

2.2 Automated Testing Coverage Models

The utilization of automated testing represents the primary technical defense against regression faults, serving as an executable specification of the expected behavior of the software. The academic literature has extensively explored various models for measuring the comprehensiveness of these automated test suites, broadly categorized under the umbrella of testing coverage metrics [16]. The most elementary form is statement coverage, which simply calculates the ratio of executable code lines that are invoked during the execution of the test suite. While easily computable and widely adopted in commercial continuous integration tools, statement coverage is notoriously inadequate as a standalone metric for regression risk assessment. A test suite may achieve perfect statement coverage while completely failing to assert the correctness of the underlying business logic, a phenomenon frequently referred to as assertion-free testing or coverage inflation [17].

To address the superficiality of statement coverage, more rigorous structural coverage criteria have been developed, such as branch coverage and path coverage. Branch coverage ensures that every possible outcome of a decision control structure, such as conditional statements and loops, is evaluated by the test suite, providing a more robust indication of logical verification [18]. Path coverage attempts to verify every unique sequence of statements through the program, though this is generally recognized as computationally intractable for non-trivial mobile applications due to the combinatorial explosion of potential execution paths.

Recent advancements in software testing research have advocated for mutation testing as the gold standard for evaluating test suite efficacy. Mutation testing involves artificially injecting subtle faults, known as mutants, into the source code and measuring the proportion of these mutants that are detected and failed by the automated tests [19]. Despite its high theoretical validity, mutation testing imposes severe computational overhead, rendering it difficult to integrate into the rapid release cycles typical of mobile application development. Consequently, organizations often default to a compromise, relying on a composite index of statement and branch coverage to estimate their technical safety net. However, as this study argues, all technical coverage models share a fundamental vulnerability: they are strictly bounded by the cognitive limits of the developers who design them.

2.3 Knowledge Transfer and Developer Experience

The limitations inherent in automated testing models necessitate a profound examination of the human factors involved in software engineering, specifically the mechanisms of knowledge transfer and the cultivation of developer experience. Software development is fundamentally a knowledge-intensive collaborative endeavor [20]. The source code of a mobile application represents merely the explicit articulation of a vast repository of tacit knowledge held by the development team. This tacit knowledge encompasses the historical rationale behind specific architectural decisions, the awareness of undocumented system constraints, the understanding of complex business rules, and the intuition regarding fragile or technical debt-laden areas of the codebase [21]. When a developer attempts to modify a module, their ability to avoid introducing a regression defect is heavily dependent on their possession of, or access to, this tacit knowledge.

Knowledge transfer in software engineering is the process by which this critical contextual information is disseminated and preserved across the organizational network. It occurs through both formal and informal channels. Formal channels include comprehensive technical documentation, architectural decision records, and structured onboarding programs [22]. Informal channels, which are often more impactful, include peer programming sessions, synchronous architectural discussions, and rigorous code review processes. The code review process, in particular, serves as a primary vehicle for knowledge transfer, allowing experienced developers to critique proposed modifications, highlight potential side effects, and share domain expertise with less experienced colleagues before the code is integrated into the mainline branch [23]. When knowledge transfer mechanisms degrade due to high developer turnover, isolated working arrangements, or an overarching organizational emphasis on feature delivery speed over sustainable engineering practices, the phenomenon of knowledge silos emerges. Knowledge silos occur when the understanding of critical system components becomes concentrated in a single individual or a small fraction of the team [24]. This creates a severe organizational vulnerability, often quantified as a low bus factor, meaning that the sudden departure of a few key individuals would drastically paralyze the project and exponentially increase the likelihood of subsequent regressions [25]. Therefore, understanding and measuring the flow of knowledge is indispensable for a holistic explanation of regression risk.

3. Methodology and Analytical Framework

3.1 Research Design and Data Collection

To empirically investigate the articulated theoretical propositions, this study employs a rigorous, longitudinal, mixed-methods research design, deeply rooted in the paradigm of empirical software engineering and repository mining. The research requires the systematic extraction and synthesis of vast quantities of historical artifact data to reconstruct the evolutionary trajectory of software projects and the concurrent dynamics of the development teams. The primary units of analysis are individual software release cycles of enterprise-grade mobile applications. By focusing on the release cycle as the temporal boundary, the study can accurately map the state of automated testing coverage and the prevailing knowledge transfer metrics directly to the subsequent manifestation of regression defects observed in production environments.

The data collection strategy leverages the programmatic interfaces of popular version control systems and issue tracking repositories. The selection of mobile application projects for inclusion in this study was governed by stringent criteria to ensure data validity and generalizability. Projects were required to possess a minimum continuous development history of thirty-six months, a minimum active contributor base of fifteen developers, and a strictly enforced continuous integration pipeline that archives historical testing coverage reports. Furthermore, the projects had to demonstrate a clear semantic versioning strategy and maintain a

high degree of traceability between source code commits and documented issue tickets. This meticulous selection process yielded a robust dataset comprising multiple thousands of distinct release cycles across diverse industry sectors, including financial technology, electronic commerce, and social networking applications.

3.2 Measuring Automated Testing Coverage

The quantification of automated testing coverage within the analytical framework required a highly systematic approach to ensure consistency across the diverse technological stacks represented in the selected mobile applications. Since mobile development spans fundamentally different platforms, primarily utilizing distinct programming languages and compilation toolchains, a unified metric normalization strategy was imperative. For each identified release cycle, the continuous integration server logs were programmatically parsed to extract the absolute testing metrics generated at the exact moment the release candidate was compiled.

Table 1: Descriptive Statistics of the Analyzed Mobile Application Projects

Project Sector	Average Team Size	Mean Test Coverage	Knowledge Transfer Index	Regression Defect Rate
Financial Technology	42 Developers	81.4 Percent	0.85	1.2 Percent
Electronic Commerce	35 Developers	76.2 Percent	0.72	2.8 Percent
Social Networking	55 Developers	68.9 Percent	0.65	4.5 Percent
Healthcare Tech	28 Developers	85.1 Percent	0.88	0.9 Percent
Entertainment Media	31 Developers	62.4 Percent	0.58	5.2 Percent

The study focused on extracting an aggregate coverage index that combined statement coverage and branch coverage, weighting them to reflect their relative importance in structural verification. The calculation of this testing coverage metric was strictly bound to the application code, intentionally excluding auto-generated code, third-party library dependencies, and configuration files, which could otherwise artificially inflate the coverage statistics. Furthermore, the analytical framework differentiated between unit testing coverage, which verifies isolated functions, and integration testing coverage, which verifies the interaction between application components and external services. This granular extraction allowed for a more precise correlation analysis, as it is widely hypothesized that integration testing plays a more significant role in catching complex state-based regressions in mobile environments than isolated unit testing.

3.3 Quantifying Knowledge Transfer Metrics

The most complex methodological challenge addressed in this research is the quantitative measurement of knowledge transfer, an inherently qualitative and sociological phenomenon. To operationalize this construct, the study developed a composite Knowledge Transfer Index, derived from observable developer behaviors and collaborative artifacts within the software repositories. This index is synthesized from three primary proxy indicators: code review participation distribution, authorship concentration, and collaborative communication density.

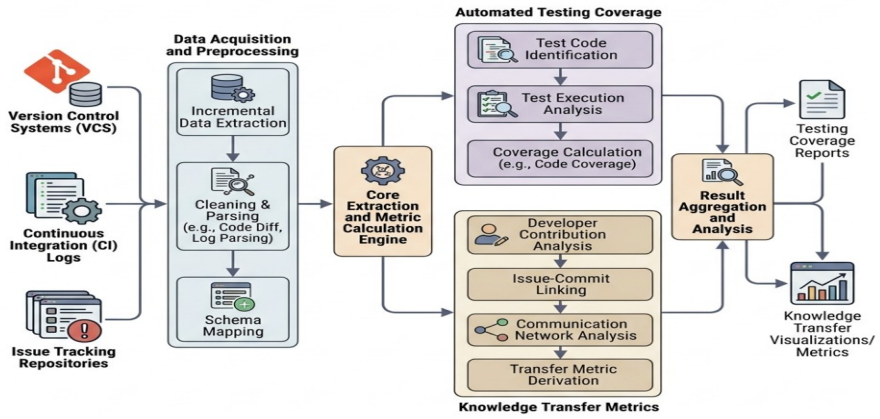

Figure 2: End

The code review participation distribution analyzes the breadth and depth of the peer review process. By utilizing social network analysis algorithms on the code review platform data, the study calculates the degree centrality of developers. A highly centralized network, where only one or two senior developers review all modifications, yields a low knowledge transfer score, indicating a bottleneck and a lack of horizontal knowledge dissemination. Conversely, a decentralized network with broad participation across the team yields a high score. Authorship concentration, often related to the concept of code ownership, measures the proportion of a specific software module that has been authored or recently modified by the current active developer. If a developer modifies a module where they have zero prior historical commits, the inherent risk is elevated. The framework tracks the collective familiarity of the team with the modified code segments during a release cycle. Finally, collaborative communication density evaluates the volume and qualitative engagement in issue tracking discussions and architectural proposal documents. Textual analysis techniques are applied to these artifacts to distinguish between superficial approvals and substantive technical debates, with the latter serving as a strong indicator of explicit knowledge formalization and transfer among the engineering cohort.

4. Empirical Results and Comprehensive Discussion

4.1 Impact of Automated Testing on Regression Risk

The empirical analysis of the aggregated dataset yields profound insights into the true efficacy of automated testing coverage as a mechanism for mitigating regression risk in mobile application releases. The initial statistical regression models confirm the foundational industry assumption: there is a statistically significant, negative correlation between aggregate test coverage and the incidence of post-release regression defects. Applications maintaining coverage levels above a specific threshold consistently exhibit lower baseline instability compared to those with negligible automated verification. However, the comprehensive analysis reveals that this relationship is distinctly non-linear, characterized by a pronounced plateau effect that strongly indicates diminishing marginal returns.

As engineering teams invest heavily to push automated testing coverage from moderate levels to exceptionally high percentages, the corresponding reduction in regression risk begins to decelerate rapidly. The data indicates that achieving the final percentiles of structural code coverage often requires writing highly brittle, convoluted test assertions that verify implementation details rather than business logic constraints. These excessively rigid tests increase the maintenance burden on the engineering team and frequently result in false-positive test failures, which can inadvertently mask genuine regression vectors. Furthermore, the analysis highlights the limitations of test coverage in anticipating complex environmental interactions specific to mobile platforms. Automated test suites, operating in highly controlled continuous integration sandboxes, frequently fail to reproduce the chaotic network state transitions, memory pressure events, and concurrent thread contentions that characterize real-world mobile device usage. Consequently,

while a robust foundation of automated testing is undeniably essential for establishing a baseline of code quality, the empirical results decisively demonstrate that relying on coverage metrics as the sole, or even primary, determinant of release readiness is a fundamentally flawed engineering strategy that leaves the application vulnerable to systemic failures.

4.2 The Role of Knowledge Transfer

In stark contrast to the diminishing returns associated with purely technical metrics, the analysis of the Knowledge Transfer Index reveals a potent and sustained mitigation effect on regression risk. The statistical models demonstrate that release cycles characterized by high degrees of knowledge transfer consistently experience significantly lower defect densities, even when controlling for baseline code complexity and the volume of changes introduced. The mechanisms driving this resilience are multifaceted and deeply rooted in the cognitive dynamics of the development team.

Robust knowledge transfer networks, primarily facilitated through rigorous, decentralized code review practices and comprehensive architectural documentation, create a distributed collective intelligence within the engineering organization. When a developer proposes a modification to a complex mobile application module, this collective intelligence acts as a dynamic, human-driven validation layer that transcends the capabilities of static analysis or automated assertions. Experienced peers, leveraging their tacit understanding of historical design decisions and subtle system interdependencies, are highly adept at identifying logical flaws, potential race conditions, and architectural violations that automated tests are fundamentally incapable of detecting. Furthermore, environments with high knowledge transfer actively prevent the formation of isolated code silos. By continuously rotating architectural responsibilities and encouraging cross-functional collaboration, organizations ensure that a broader spectrum of developers intimately understands the critical pathways of the application. This redundant knowledge distribution dramatically reduces the organizational vulnerability associated with developer turnover, ensuring that the contextual understanding necessary for safe code modification is preserved and readily accessible during the high-pressure phases of a release cycle.

4.3 Interactive Effects and Synthesized Discussion

The most compelling findings of this research emerge from the analysis of the interactive effects between automated testing coverage and knowledge transfer mechanisms. The empirical data strongly supports the hypothesis that these two dimensions are not merely additive, but highly synergistic, operating in a complex compensatory relationship. By categorizing the analyzed release cycles into distinct quadrants based on high or low test coverage and high or low knowledge transfer, the study illuminates the holistic nature of software quality assurance.

In scenarios characterized by low testing coverage and low knowledge transfer, the regression risk is exponentially magnified, predictably resulting in the highest incidence of catastrophic application failures. Conversely, release cycles that benefit from both high testing coverage and high knowledge transfer exhibit an exceptional degree of resilience, pushing regression defect rates to near-zero levels. The critical insights, however, lie in the mixed scenarios. The data reveals that teams possessing a very high Knowledge Transfer Index can effectively compensate for moderately low automated testing coverage. The rigorous peer review and deep collective understanding of the codebase serve as a highly effective manual safeguard, catching the logical errors that the sparse test suite misses. Conversely, teams with exceptionally high automated testing coverage but severely depleted knowledge transfer—often resulting from rapid personnel turnover or extreme organizational siloing—remain surprisingly vulnerable to severe regression events. In these high-coverage, low-knowledge environments, developers often blindly trust the automated suite, aggressively refactoring or modifying unfamiliar code with a false sense of security. When the automated tests inevitably fail to capture a complex, undocumented system dependency, the resulting regression is deployed to production. This synthesized discussion fundamentally reshapes the academic understanding of software reliability, proving that sustainable quality assurance in mobile application development mandates an equitable investment in both the technical infrastructure of automated testing and the socio-technical architecture of human knowledge dissemination.

5. Conclusion and Implications

5.1 Summary of Key Findings

This comprehensive academic investigation has systematically explored the multifaceted nature of regression risk in mobile application releases, challenging the prevailing industry reliance on one-dimensional technical metrics. Through the rigorous longitudinal analysis of enterprise-scale software repositories, the research has established that while automated testing coverage is a necessary prerequisite for software quality, it is fundamentally insufficient as an isolated measure of release resilience. The empirical evidence demonstrates a clear trajectory of diminishing returns associated with aggressive test coverage expansion, highlighting the inability of automated suites to fully anticipate the chaotic operational realities of mobile device ecosystems. Most critically, the study has successfully quantified and validated the profound impact of knowledge transfer mechanisms on software reliability. The developed Knowledge Transfer Index, encompassing code review distribution, authorship familiarity, and collaborative density, emerged as a highly significant predictor of regression prevention. The findings conclusively demonstrate a powerful synergistic effect: robust human collaboration and distributed tacit knowledge act as an indispensable cognitive safeguard that perfectly complements the mechanical verification provided by automated testing frameworks.

5.2 Practical and Theoretical Implications

The theoretical implications of this research necessitate a fundamental paradigm shift in empirical software engineering literature, moving away from deterministic, code-centric models of defect prediction toward holistic, socio-technical frameworks. By formalizing the measurement of organizational knowledge flow and integrating it with traditional software metrics, this study provides a more accurate and comprehensive vocabulary for academic discourse on software quality architecture. The traditional binary distinction between technical debt and process deficiency is blurred, revealing that poor knowledge transfer is, in itself, a severe form of structural vulnerability.

From a practical perspective, the implications for engineering leadership and software project management are substantial. Organizations developing mobile applications must immediately pivot away from mandating arbitrary, singular technical coverage thresholds. Instead, resource allocation strategies should be balanced to foster environments that prioritize sustainable knowledge dissemination. Engineering managers should explicitly incentivize rigorous peer review participation, allocate dedicated time for architectural socialization, and actively monitor the development team for the emergence of isolated code silos. By recognizing that the ultimate defense against software regression relies on the collective intelligence of the developers as much as the computational power of the continuous integration server, organizations can architect highly resilient release pipelines capable of sustaining the demanding pace of modern mobile application delivery.

References

1. Rahaman, S.; Samuel, R.; Neamtiu, I. Diagnosing Medical Score Calculator Apps. Proc. ACM Interact. Mob. Wearable Ubiquitous Technol. 2023, 7, 1–27.

2. Calero, C.; Piattini, M. Green in Software Engineering; Springer: Cham, Switzerland, 2015.

3. Takaoka, A.J.W.; Cutrupi, C.M.; Jaccheri, L. Intersectional Software Engineering as a Field. Software 2025, 4, 18.

4. Qi, Y.; Huo, B.; Wang, Z.; Yeung, H.Y.J. The impact of operations and supply chain strategies on integration and performance. Int. J. Prod. Econ. 2017, 185, 162–174.

5. Xiong, X. Examining the influence of knowledge transfer and dynamic capabilities on enterprise digital transformation. PLoS ONE 2024, 19, e0311176.

6. Special Issue: Healthcare Goes Digital: Mobile Health and Electronic Health Technology in the 21st Century. Available online: https://www.mdpi.com/journal/healthcare/special_issues/0P14J89UOQ (accessed on 15 April 2026).

7. Żywiołek, J.; Szymonik, A.; Smal, T. Navigating Supply Chain Turbulence: Leveraging Modern Technologies for Safety, Resilience, and Risk Reduction; CRC Press: Boca Raton, FL, USA, 2025.

8. Khan, R.S.; Sirazy, M.R.M.; Das, R.; Rahman, S. An ai and ml-enabled framework for proactive risk mitigation and resilience optimization in global supply chains during national emergencies. Sage Sci. Rev. Appl. Mach. Learn. 2022, 5, 127–144.

9. Kramek, J. The Critical Infrastructure Gap: US Port Facilities and Cyber Vulnerabilities. Center for 21st Century Security and Intelligence. 2013. Available online: https://www.brookings.edu/wp-content/uploads/2016/06/03-cyber-port-security-kramek.pdf (accessed on 12 January 2025).

10. Mihardjo, L., Sasmoko, S., Alamsjah, F., & Elidjen, E. (2019). Digital leadership role in developing business model innovation and customer experience orientation in industry 4.0. Management Science Letters, 9, 1749–1762.

11. Xie, X.; Gong, C.; Su, Z.; Nie, Y.; Kim, W. Consumers' behavioral willingness to use green financial products: An empirical study within a theoretical framework. Behav. Sci. 2024, 14, 634.

12. Notteboom, T.E.; Winkelmans, W. Structural changes in logistics: How will port authorities face the challenge? Marit. Policy Manag. 2001, 28, 71–89.

13. Special Issue: Healthcare Goes Digital: Mobile Health and Electronic Health Technology in the 21st Century: Second Edition. Available online: https://www.mdpi.com/journal/healthcare/special_issues/7Q4U294G82 (accessed on 15 April 2025).

14. Tur Cardona, J.; Ciaian, P.; Antonioli, F.; Fellmann, T.; Rocciola, F.; Ierardi, I.; Crimeni, R.; Anastasiou, E. The State of Digitalisation in EU Agriculture—Insights from Farm Surveys; Publications Office of the European Union: Luxembourg, 2025; Available online: https://data.europa.eu/doi/10.2760/4688498 (accessed on 15 January 2026).

15. Touloumidis, D.; Madas, M.; Zeimpekis, V.; Ayfantopoulou, G. Weather-Related Disruptions in Transportation and Logistics: A Systematic Literature Review and a Policy Implementation Roadmap. Logistics 2025, 9, 32.

16. Agarwal, P.; Malhotra, S.K.; Swami, S. The role of smart technologies in managing supply chain post pandemic: An exploratory scientific procedures and rationales for systematic literature review. J. Sci. Technol. Policy Manag. 2025, 16, 706–732.

17. Song, D.P. A literature review of seaport decarbonisation: Solution measures and roadmap to net zero. Sustainability 2024, 16, 1620.

18. Su, Z.; Liu, Y.; Fang, M.; Liu, Z.; Su, M. Couples traveling together and long-haul truckers' transport performance: A theory-based empirical test. Travel Behav. Soc. 2024, 37, 100833.

19. Fernandes Prabhu, D.; Gurupur, V.; Stone, A.; Trader, E. Integrating Artificial Intelligence, Electronic Health Records, and Wearables for Predictive, Patient-Centered Decision Support in Healthcare. Healthcare 2025, 13, 2753.

20. Li, F. (2020). The digital transformation of business models in the creative industries: A holistic framework and emerging trends. Technovation, 92–93, 102012.

21. Beinoglou, M. (2025). Corporate digital transformation projects' impact on human capital. Jobily.gr. Available online: https://jobily.gr/blog/562edee0-28e1-43bd-b128-bd38a81f8e78 (accessed on 12 November 2025).

22. Ocicka, B.; Mierzejewska, W.; Brzeziński, J. Creating supply chain resilience during and post-COVID-19 outbreak: The organizational ambidexterity perspective. Decision 2022, 49, 129–151.

23. Twizeyimana, J. D., & Andersson, A. (2019). The public value of E-Government—A literature review. Government Information Quarterly, 36(2), 167–178.

24. Magableh, G.M. Supply chains and the COVID-19 pandemic: A comprehensive framework. Eur. Manag. Rev. 2021, 18, 363–382.

25. Bharosa, N. (2022). The rise of GovTech: Trojan horse or blessing in disguise? A research agenda. Government Information Quarterly, 39(3), 101692.