The American Journal of Engineering and Technology
ISSN 2689-0984 Volume 08 - 2026

AI-Powered Test Automation Frameworks for Next-Generation Software Quality Engineering

Dr. Tomas Kazlauskas
Department of Artificial Intelligence Engineering, Vilnius Institute of Digital Technologies, Lithuania

Received: 28 May 2026 | Received Revised Version: 20 June 2026 | Accepted: 15 July 2026 | Published: 17 August 2026

Volume 08 Issue 08 2026 | Crossref DOI: 10.37547/tajet/Volume08Issue08-03

Abstract

Agile software development emphasizes rapid iteration, continuous integration, frequent releases, and incremental delivery, making regression testing a central software quality challenge. Conventional regression testing approaches often depend on manually selected test suites, static prioritization rules, and repeated execution of tests that provide limited incremental fault-detection value. This paper develops a conceptual intelligent regression testing framework that applies artificial intelligence (AI) techniques to test selection, prioritization, execution, failure classification, and continuous learning within Agile development pipelines. The methodological foundation combines supervised learning, representation learning, historical test-result analysis, change-impact assessment, and feedback-driven optimization. Because the supplied literature primarily concerns AI-based detection and classification in biomedical signal-processing applications rather than software testing, the paper explicitly treats these studies as methodological evidence for transferable AI patterns rather than direct empirical evidence for regression testing. The framework consequently emphasizes feature extraction, automated classification, adaptive prediction, and real-time decision support. A conceptual evaluation indicates that AI-assisted regression testing can improve the alignment between code changes and test execution priorities, reduce redundant execution, and create feedback loops capable of adapting to changing Agile projects. However, model drift, insufficient historical data, explainability, false prioritization, and integration complexity remain significant constraints. The analysis positions intelligent regression testing as an adaptive decision-support layer rather than a complete replacement for conventional testing practices.

Keywords: Artificial Intelligence, Regression Testing, Agile Software Development, Test Automation, Machine Learning, Test Prioritization, Continuous Integration, Intelligent Software Testing, Predictive Analytics.

Cite This Article: Dr. Tomas Kazlauskas. (2026). AI-Powered Test Automation Frameworks for Next-Generation Software Quality Engineering. The American Journal of Engineering and Technology, 8(08), 24–30. Retrieved from https://theamericanjournals.com/index.php/tajet/article/view/8320

1. Introduction

1.1 Problem Statement

Agile software development has transformed software delivery from periodic release cycles into continuous and incremental engineering processes. Features are frequently added, modified, refactored, and integrated, requiring testing activities to operate at a comparable speed. Regression testing becomes particularly demanding because modifications in one component may affect existing functionality in another component. Executing the entire regression suite after every change provides broad coverage but can introduce substantial execution time, infrastructure consumption, and feedback delays. Conversely, executing only a small manually selected subset may accelerate delivery while increasing the probability that important failures remain undetected.

Artificial intelligence provides a potential mechanism for addressing this trade-off by converting historical testing information into predictive and adaptive decisions. Instead of treating all test cases as equally important, an intelligent framework can estimate the probability that an individual test will expose a defect, identify tests associated with recently modified components, and

dynamically prioritize execution. The underlying concept is consistent with AI systems demonstrated in other domains, where automated feature extraction and classification replace purely manual decision processes. For example, deep learning has been applied to automated signal classification and detection, demonstrating how learned representations can support complex decision-making tasks (Abdelhameed, Daoud, & Bayoumi, 2018; Acharya, Oh, Hagiwara, Tan, & Adeli, 2018).

The relevance of AI-driven automation to modern software quality engineering is particularly evident when testing must operate continuously. Ramamurthy (2023) describes AI-driven test automation as an emerging framework for modern software quality engineering, supporting the positioning of intelligent mechanisms within automated testing processes. The principal research problem therefore concerns how AI can be incorporated into regression testing without compromising reliability, explainability, and test coverage.

1.2 Research Objectives

This paper aims to develop a conceptual framework for intelligent regression testing in Agile environments. The objectives are to examine how AI can support regression-test selection and prioritization, define an adaptive architecture connecting software changes with historical testing information, establish a methodological workflow for model-based testing decisions, and critically assess the benefits and limitations of AI-based regression automation.

1.3 Scope and Significance

The proposed framework focuses on regression testing within continuous Agile development. Its functional scope includes change analysis, test-case representation, predictive prioritization, automated execution, failure classification, and feedback-based model updating. It does not claim that AI independently guarantees software correctness. Rather, AI is positioned as an intelligent optimization layer operating alongside established testing practices.

The significance of this approach lies in its potential to reduce the mismatch between software-development velocity and testing capacity. AI-based systems in other technically complex domains demonstrate that automated feature learning and classification can transform large, multidimensional data into actionable predictions (Ahmedt-Aristizabal et al., 2018; Ansari et al., 2019). The same general computational principle can be conceptually transferred to software quality data, provided that domain-specific validation is performed.

2. Literature Review

The supplied literature predominantly examines AI and deep-learning applications for seizure detection, classification, localization, and prediction. Although these studies do not directly investigate software regression testing, they provide methodological evidence concerning automated classification, feature representation, predictive modeling, and real-time decision support.

Abdelhameed, Daoud, and Bayoumi (2018) demonstrate the use of a deep convolutional autoencoder for automated seizure detection. The methodological significance for intelligent regression testing is the transformation of complex raw information into learned representations. Software-testing environments similarly generate multidimensional information, including source-code changes, test histories, execution duration, failure frequencies, dependencies, and previous defect associations. An intelligent testing system can therefore use representation-learning principles to convert these heterogeneous variables into features suitable for predictive decision-making.

Acharya, Hagiwara, and Adeli (2018) examine automated seizure prediction, while Acharya et al. (2018) apply a deep convolutional neural network to automated seizure detection and diagnosis. Collectively, these studies demonstrate the distinction between detecting an existing event and predicting a future event. This distinction is directly relevant to regression testing. A testing system that merely detects failures after execution is reactive; an intelligent regression framework should additionally estimate which tests are likely to fail before execution and prioritize them accordingly.

The studies by Achilles et al. (2016, 2018) emphasize convolutional neural networks for real-time seizure detection. Their relevance is conceptual rather than domain-specific: real-time decision environments require computational methods capable of generating useful classifications without unacceptable delay. Agile continuous-integration pipelines impose a similar temporal requirement. A regression-testing model that takes longer to generate predictions than the saved testing time would have limited practical value.

Ahmed et al. (2018) present a review of focal and non-focal epilepsy localization, illustrating how AI-oriented systems can organize complex classification problems into meaningful categories. Ahmedt-Aristizabal et al. (2018) further investigate deep facial analysis and computer vision for epilepsy evaluation, while another study by the same authors addresses deep classification of epileptic signals (Ahmedt-Aristizabal et al., 2018). These works support the broader theoretical proposition that automated classification can assist expert decision-making when the underlying data contain patterns that are difficult to identify consistently through manual inspection.

Akut (2019) examines a wavelet-based deep-learning approach for epilepsy detection. The combination of signal transformation and deep learning is particularly informative for regression testing because software data may also require preprocessing before predictive modeling. Raw test logs, source-code changes, stack traces, execution histories, and dependency information cannot necessarily be supplied directly to every learning algorithm. Appropriate feature engineering or representation learning is therefore an essential architectural stage.

Ansari et al. (2019) investigate neonatal seizure detection using deep convolutional neural networks. The study further demonstrates the applicability of automated deep-learning classification to complex and noisy information. For regression testing, analogous challenges arise because historical test results can contain flaky tests, incomplete logs, environment-specific failures, and changing software dependencies.

The major research gap is therefore evident. The supplied literature provides strong examples of AI-based detection, prediction, classification, and real-time analysis, but it does not provide direct empirical validation of AI-based regression testing. Consequently, the proposed framework should be interpreted as a theoretically grounded architecture rather than as a claim that biomedical AI results directly prove software-testing performance. Ramamurthy (2023) provides the most direct conceptual connection to AI-driven test automation and supports the integration of intelligent mechanisms into software quality engineering.

3. Methodology

3.1 Conceptual Research Design

This research adopts a framework-development methodology. Rather than presenting fabricated experimental measurements, it derives an intelligent regression-testing architecture from established AI principles represented in the supplied literature and applies those principles conceptually to Agile software-testing workflows.

The proposed architecture contains six connected stages: change analysis, feature construction, predictive modeling, test prioritization, automated execution, and feedback-based learning. The architecture is designed around a closed-loop principle. Each software modification produces new testing evidence, which is subsequently incorporated into the system to improve future predictions.

3.2 Change-Impact Analysis

The first stage identifies what has changed between two software versions. Relevant features may include modified files, classes, functions, modules, dependency relationships, code complexity indicators, historical defect associations, and previous test-to-component relationships.

For example, if a payment-service module is modified, the framework should not treat all 5,000 regression tests equally. Tests directly associated with payment processing, authentication, database transactions, and dependent interfaces should receive greater initial attention. The purpose is not to eliminate unrelated tests permanently but to establish an evidence-based execution order.

3.3 Intelligent Feature Representation

The second stage transforms software and testing information into machine-readable representations. Features can include test execution frequency, previous failure rate, average execution time, last failure timestamp, affected source components, historical defect density, and relationships between tests and modified modules.

The conceptual rationale follows the representation-learning principle demonstrated in AI classification studies. Abdelhameed, Daoud, and Bayoumi (2018) demonstrate how deep learning can derive useful representations from complex inputs, while Acharya et al. (2018) demonstrate automated classification using learned patterns. In software testing, such representations can allow the model to recognize combinations of variables associated with regression failures.

3.4 Predictive Test-Prioritization Model

The predictive layer estimates a risk score for every candidate regression test. A conceptual formulation can be represented as:

$$\text{Risk Score}(T_i) = f(C_i, H_i, D_i, F_i, R_i)$$

where C_i represents change impact, H_i represents historical failure behavior, D_i represents dependency relevance, F_i represents execution characteristics, and R_i represents recent test outcomes.

A supervised classification model may categorize tests as high-, medium-, or low-priority. Alternatively, a ranking model can generate a continuous ordering. The latter is preferable when the objective is to execute the highest-value tests first rather than merely classify them.

3.5 Automated Execution Layer

After prioritization, the framework passes tests to the continuous-integration environment. High-risk tests are executed first, allowing developers to receive earlier feedback. Tests with lower predicted relevance remain available for subsequent execution to preserve overall coverage.

This design reflects the real-time emphasis evident in AI detection research. Achilles et al. (2016, 2018) demonstrate the importance of computational approaches suitable for real-time detection. In Agile testing, the equivalent requirement is rapid feedback after code integration.

3.6 Failure Classification

The framework should distinguish between genuine regression failures, environmental failures, infrastructure problems, and potentially flaky tests. This stage prevents every failed test from being interpreted as evidence of a software defect.

For instance, if a test fails because a dependent service is temporarily unavailable, its failure should not necessarily increase the predicted defect probability of the modified source code. Automated classification can assist this distinction by analyzing historical failure signatures, logs, execution context, and related test behavior.

3.7 Feedback and Continuous Learning

The final stage feeds execution outcomes back into the model. Correct predictions reinforce existing associations, while incorrect predictions provide training information for future iterations.

This feedback architecture is important because Agile software systems evolve continuously. A static model can become inaccurate when architecture, dependencies, development practices, or test suites change. The adaptive principle is consistent with predictive AI applications in which historical observations are used to improve subsequent decisions (Acharya, Hagiwara, & Adeli, 2018).

3.8 Hypothetical Agile Implementation

Consider an Agile project containing 2,000 automated regression tests. A developer modifies an authentication component. The proposed framework analyzes the code change, identifies associated test cases, evaluates historical failure patterns, assigns risk scores, and executes high-risk authentication and dependent-interface tests first. If one of these tests fails, the pipeline immediately provides feedback to the development team while lower-priority tests continue asynchronously.

The value of the approach is not simply reducing the number of tests. Its principal contribution is changing the execution order according to predicted information value. Ramamurthy (2023) similarly situates AI-driven automation within modern quality-engineering practices, supporting the use of intelligent mechanisms as an augmentation of automated testing.

4. Results

The conceptual analysis indicates that the proposed framework can be organized around four primary outcomes: improved prioritization, reduced redundant execution, adaptive failure analysis, and continuous learning. These outcomes are architectural findings derived from the framework rather than experimentally measured performance claims.

First, AI-based prioritization provides a more dynamic alternative to static regression-test ordering. Conventional strategies frequently use fixed priorities determined by business importance or manually defined categories. An intelligent model can incorporate multiple dimensions simultaneously, including recent source-code changes, historical failures, dependency relationships, and test execution behavior. This allows test ordering to evolve with the software system rather than remaining unchanged throughout the project lifecycle.

Second, the framework provides a mechanism for

reducing redundant early-stage execution. A complete regression suite may contain tests with low probability of detecting defects associated with a particular modification. Executing these tests first provides limited immediate information to developers. By ranking tests according to predicted risk, the framework can concentrate early execution resources on tests with greater expected diagnostic value. The resulting benefit is primarily a reduction in feedback latency rather than an assumption that low-ranked tests can be permanently removed.

Third, automated failure classification can improve the quality of regression feedback. A failed test is not automatically equivalent to a software defect. Infrastructure interruptions, unavailable services, unstable test environments, and flaky behavior can produce failures unrelated to the current code change. An intelligent classifier can use historical patterns to separate these categories, reducing unnecessary developer investigation. The classification principle has methodological parallels with AI systems used to distinguish complex biomedical signal categories (Ahmedt-Aristizabal et al., 2018; Ansari et al., 2019).

Fourth, the feedback mechanism enables continuous model adaptation. Agile development continuously changes both application behavior and the testing environment. Therefore, a model trained once and never updated would progressively lose relevance. Incorporating new test outcomes enables the framework to modify its predictions as new evidence accumulates.

However, the analysis also identifies important constraints. Prediction quality depends on the availability and reliability of historical data. Projects with limited regression history may not provide sufficient observations for effective learning. Furthermore, flaky tests can introduce misleading training signals, while major architectural changes may produce patterns that differ substantially from historical data. Deep-learning approaches can also introduce explainability challenges, particularly when developers need to understand why a specific test was assigned a high or low priority.

Consequently, the framework should not be interpreted as an autonomous replacement for conventional testing. Its most defensible role is as an intelligent decision-support layer that optimizes test sequencing while retaining comprehensive regression execution when project risk requires it.

5. Discussion

The findings suggest that intelligent regression testing should be understood as an optimization problem rather than simply an automation problem. Traditional automation reduces manual execution effort, whereas AI-based automation attempts to determine which automated activities should receive attention first and why. This distinction is important for Agile environments where the volume of automated tests can increase faster than the practical time available for immediate feedback.

The literature provides indirect theoretical support for this architecture. The supplied AI studies demonstrate that complex data can be transformed into representations suitable for automated classification and prediction (Abdelhameed, Daoud, & Bayoumi, 2018; Acharya et al., 2018). Their methodological relevance lies in demonstrating the feasibility of learning patterns from large and multidimensional datasets. In software testing, the equivalent dataset consists of historical test outcomes, code changes, dependencies, execution traces, and defect information. Nevertheless, the difference between biomedical signal classification and software regression prediction must not be ignored. A model successful in one domain cannot be assumed to achieve comparable performance in another.

An important theoretical implication is that regression testing can be represented as a sequential decision problem. At every development iteration, the framework determines which available test should be executed next based on the information currently available. The optimal decision is therefore not necessarily the test with the highest historical failure rate. A test with moderate failure probability but strong dependency on a recently modified component may provide greater immediate value. This supports a multidimensional ranking model rather than a single-variable heuristic.

A practical implication concerns the relationship between AI and human testers. Excessive automation can create organizational risk if developers accept model predictions without questioning them. Explainability should therefore be integrated into the framework. For example, the system could indicate that a test was prioritized because its dependent component changed, its historical failure frequency increased, and similar modifications previously generated failures. Such explanations transform the model from a black-box scheduler into an interpretable decision-support mechanism.

Model drift represents another significant limitation. Agile systems evolve through architectural changes, library upgrades, refactoring, and changes in test suites. Historical correlations can therefore become obsolete. Continuous monitoring and periodic retraining are necessary. The adaptive-learning concept is consistent with the predictive orientation observed in AI-based detection and classification research (Acharya, Hagiwara, & Adeli, 2018; Akut, 2019).

There is also a fundamental trade-off between efficiency and coverage. Aggressive prioritization may shorten feedback time but increase the possibility that a low-ranked test exposes a defect later. The framework should consequently distinguish prioritization from elimination. Prioritization changes execution order; elimination permanently removes tests. The former is considerably safer because complete regression coverage can still be performed within the pipeline or during scheduled validation cycles.

The most important limitation of this study is the absence of direct empirical software-testing references in the supplied bibliography. Ramamurthy (2023) provides a direct conceptual connection to AI-driven test automation, but the remaining references concern biomedical AI applications. Therefore, claims regarding actual reductions in testing time, defect-detection rates, or computational costs cannot responsibly be presented as measured results. Future research should empirically evaluate the proposed framework using real Agile repositories, controlled test suites, historical defects, and standard evaluation measures such as fault-detection effectiveness, test-execution time, prioritization accuracy, and feedback latency.

Overall, the framework provides a theoretically coherent bridge between AI-based prediction and Agile regression testing while recognizing the limitations of transferring methodologies across application domains. Ramamurthy (2023) reinforces the broader direction toward AI-enabled quality engineering, but domain-specific validation remains essential.

6. Conclusion

This paper proposed an intelligent regression testing framework designed for Agile software development. The framework integrates change-impact analysis, feature representation, predictive test prioritization, automated execution, failure classification, and continuous feedback. Its central proposition is that AI should augment regression testing by dynamically determining the sequence and relative importance of test execution rather than attempting to replace comprehensive testing.

Analysis of the supplied literature indicates that automated AI systems can successfully perform complex detection, prediction, and classification tasks in other domains. These methodological principles provide a theoretical foundation for intelligent testing, particularly in the areas of learned representations, predictive classification, real-time decision support, and adaptive feedback. However, because most supplied references address biomedical applications, they cannot serve as direct empirical evidence for software-testing effectiveness.

The framework therefore contributes primarily at the conceptual and architectural level. Its practical value depends on high-quality historical testing data, reliable failure labels, appropriate model selection, explainability, and continuous adaptation. The most appropriate deployment strategy is a hybrid model in which AI prioritizes and analyzes tests while conventional automated and human-driven testing mechanisms preserve coverage and oversight.

Future research should implement the proposed architecture in real Agile continuous-integration environments and compare AI-based prioritization with conventional regression strategies. Particular attention should be given to model drift, flaky-test detection, explainable prioritization, cross-project generalization, computational overhead, and the relationship between prediction accuracy and actual defect-detection effectiveness.

References

1. Abdelhameed, A. M., Daoud, H. G., & Bayoumi, M. (2018).Epileptic seizure detection using deep convolutionalautoencoder. In2018 IEEE International Workshop onSignal Processing Systems (SiPS)(pp. 223–228). IEEE.
2. Acharya, U. R., Hagiwara, Y., & Adeli, H. (2018). Automatedseizure prediction.Epilepsy & Behavior,88, 251–261.
3. Acharya, U. R., Oh, S. L., Hagiwara, Y., Tan, J. H., &Adeli, H. (2018). Deep convolutional neural network forthe automated detection and diagnosis of seizure usingEEG signals.Computers in Biology and Medicine,100,270–278.

4. Achilles, F., Tombari, F., Belagiannis, V., Loesch, A., Noachtar, S.,Navab, N. (2016). Convolutional neural networks for real-timeepileptic seizure detection.Computer Methods inBiomechanics and Biomedical Engineering: Imaging andVisualization,1163, 264–269.
5. Achilles, F., Tombari, F., Belagiannis, V., Loesch, A. M., Noachtar,S., & Navab, N. (2018). Convolutional neural networks forreal-time epileptic seizure detection.Computer Methods inBiomechanics and Biomedical Engineering: Imaging &Visualization,6, 264–269.
6. Ahmed, F. H. Arunkumar, N., Chandima, G., Abbas, K.,et al.(2018). Focal and non-focal epilepsy localization: A review.IEEE Access,6, 49306–49324.
7. Ahmedt-Aristizabal, D., Fookes, C., Nguyen, K., Denman, S.,Sridharan, S., & Dionisio, S. (2018). Deep facial analysis: Anew phase in epilepsy evaluation using computer vision.Epilepsy & Behavior,82,17–24.
8. Ahmedt-Aristizabal, D., Fookes, C., Nguyen, K., & Sridharan, S.(2018). Deep classification of epileptic signals. In2018 40thAnnual International Conference of the IEEE Engineering inMedicine and Biology Society (EMBC)(pp. 332–335). IEEE.
9. Akut, R. (2019). Wavelet based deep learning approach for epilepsydetection, Health information science and systems.HealthInformation Science and Systems,7,8.
10. Ansari, A.H., Cherian, P. J., Caicedo, A., Naulaers, G., De Vos, M.,& Van Huffel, S. (2019). Neonatal seizure detection using deepconvolutional neural networks.International Journal ofNeural Systems,29, 1850011.
11. Ramamurthy, K. (2023). AI-Driven Test Automation Frameworks for the Modern Software Quality Engineering. International Journal of Emerging Trends in Computer Science and Information Technology, 4(4), 257-269.
12. Philip, P. G. (2025). Explainable Artificial Intelligence (XAI) for Project Governance: Improving Transparency and Stakeholder Trust in Automated Project Decision. Journal of Project Management Studies, 2(1), 37–55. https://doi.org/10.58425/jpms.v2i1.570