International Journal of Emerging Trends in Computer Science and Information Technology
ISSN: 3050-9246 | https://doi.org/10.63282/3050-9246.IJETCSIT-V4I4P118
Eureka Vision Publication | Volume 4, Issue 4, 167-172, 2023

Original Article

AI-Driven Decision Intelligence for Agile Software Lifecycle Governance: An Architecture-Centered Framework Integrating Machine Learning Defect Prediction and Automated Testing

Sai Dheeraj Sivva[1], Rakesh Reddy Thalakanti[2], Sai Santosh Goud Bandari[3], Sai Darshak Reddy Yettapu[4]
[1]MS in Computer Science, Texas A&M University-Corpus Christi, Texas, USA,
[2]Software Development Engineer, Amazon, Austin, Texas, USA,
[3]Software Developer, Tata Consultancy Services, Raleigh, North Carolina, USA,
[4]Independent Researcher, Dallas, Texas, USA.

Abstract - Agile enterprise software teams are expected to deliver new functionality rapidly while maintaining high reliability and compliance. Traditional governance practices struggle to keep pace with the volume and complexity of decisions that must be taken across modern microservice-based systems. Prior work introduced a decision-intelligence methodology for AI-driven agile lifecycle governance and architecture-centered project management [1]. Another study discussed how machine-learning models are becoming pervasive artefacts across the software development lifecycle [15]. Reliability implications of automated testing frameworks in Java enterprise systems have also been examined [16]. Building on these foundations, this paper proposes a Decision-Intelligent Agile Governance Framework (DI-AGF) that integrates decision intelligence with machine-learning-based software defect prediction and automated testing. DI-AGF organizes the lifecycle into four layers: (i) observability and data engineering, (ii) analytics and machine learning, (iii) decision intelligence and policy, and (iv) execution and feedback. Governance decisions such as release readiness, test scope selection, and technical-debt remediation are treated as explicit decision assets that consume defect risk scores, architectural criticality metrics, and test evidence. The paper describes the architectural structure of DI-AGF, outlines how ML defect prediction and automated testing are orchestrated in a risk-driven manner, and illustrates the framework in the context of a Java microservices platform. An evaluation strategy and research agenda are also proposed. The contribution is a consolidated, architecture-centered reference model for operationalizing decision intelligence in agile software governance.

Keywords - Decision Intelligence; Agile Software Governance; Machine Learning Defect Prediction; Automated Software Testing; Architecture-Centered Frameworks; AI-Driven Software Lifecycle; Microservices Architecture; Risk-Driven Testing; Software Reliability Engineering; Intelligent Devops.

1. Introduction

Modern software organizations operate under intense pressure to deliver features frequently while ensuring system stability, security, and compliance. Continuous delivery, microservice architectures, and cloud-native platforms have increased technical flexibility but have also multiplied the number of architectural and operational decisions made each day. Release managers, architects, and product owners must decide which changes can safely be deployed, which services require deeper testing, which modules should be refactored, and how to allocate limited quality-assurance capacity.

In many organizations, these decisions are still made using informal heuristics, static checklists, and periodic review meetings. Such mechanisms do not scale well when dozens of teams, hundreds of microservices, and thousands of commits are involved in each release train. At the same time, software analytics, machine-learning-based defect prediction, and automated testing frameworks provide rich but often fragmented signals about code quality, test coverage, and operational risk.

A unifying approach is needed that explicitly models the key governance decisions, connects them to analytics and automation, and provides systematic feedback so that decisions improve over time. Decision intelligence offers such a lens by treating decisions as engineered artefacts with defined inputs, outputs, and measurable outcomes. When combined with machine learning and automated testing, decision intelligence has the potential to transform agile governance from a largely manual process into a reproducible, data-driven capability.

This paper proposes the Decision-Intelligent Agile Governance Framework (DI-AGF), which integrates decision intelligence, software analytics, defect prediction models, and automated testing into a coherent, architecture-centered model. The framework is positioned as a conceptual reference for organizations that wish to improve release-time decision-making without sacrificing agility.

2. Background and Related Work

2.1. Decision intelligence and AI-assisted project management

Decision intelligence has emerged as a discipline that combines data science, artificial intelligence, and behavioral science to improve how organizations make decisions. Gartner describes decision intelligence as a practical field that designs, models, aligns, executes, and monitors decision models and processes in the context of business outcomes [14]. Other authors emphasize the idea of decisions as reusable and improvable assets that can be documented, analyzed, and refined over time [19]. Popular accounts have also presented decision intelligence as a life-relevant skill for individuals and organizations seeking to make better choices under uncertainty [20].

Industry analyses highlight that decision-intelligence platforms extend traditional business-intelligence tooling by linking analytics to actions and embedding feedback loops that continuously update models and policies [17]. Vendor perspectives describe decision intelligence as a way to power digital transformation by enabling organizations to connect data, algorithms, and operational processes into coherent decision flows [18].

Parallel developments in project management research examine the use of machine learning and predictive analytics for planning and control. Bae and colleagues proposed a machine-learning-based decision support system for project management that integrates predictive models into project dashboards and assists managers in resource allocation and risk management [11]. Chan and co-authors explored predictive analytics in project management, focusing on AI-driven forecasting of delays and cost overruns [12]. Thota and co-workers presented case studies of AI-assisted project management where AI tools enhanced decision-making and forecasting in complex project environments [13].

Within the software lifecycle, Gunda and co-authors proposed a decision-intelligence methodology for AI-driven agile lifecycle governance and architecture-centered project management, arguing that governance decisions should be explicitly catalogued, monitored, and improved using feedback loops [1]. In related work, Gunda discussed how machine-learning models increasingly act as core artefacts that influence requirements, architecture, testing, and operations, rather than being isolated experimental components [15]. These studies motivate a closer integration of decision intelligence with software engineering practice.

2.2. Machine-learning-based software defect prediction

Software defect prediction aims to identify modules, files, or components that are likely to contain defects so that testing and review efforts can be focused on where they are most valuable. Omri and Sinz surveyed deep learning techniques for software defect prediction and reported that architectures such as convolutional and recurrent neural networks can learn useful representations from code or metrics with competitive accuracy [2]. Akimova and colleagues presented a broader survey of software defect prediction using deep learning, covering datasets, model architectures, and common evaluation practices [3].

A key challenge in defect prediction is that defect data is typically imbalanced, with many more non-defective modules than defective ones. Pandey and Kumar examined this problem and surveyed techniques such as sampling and cost-sensitive learning for software fault prediction on imbalanced datasets [4]. Santos and co-authors applied explainable machine-learning techniques to defect prediction, helping practitioners understand why a model assigns high risk to particular modules [5]. Stradowski and colleagues discussed machine-learning-based defect prediction from a business perspective, emphasizing adoption challenges, cost–benefit trade-offs, and the need to integrate predictions into decision processes [6].

Collectively, these studies indicate that defect prediction models can achieve useful levels of accuracy, but that their organizational impact depends on how predictions are surfaced, explained, and linked to specific quality decisions in the lifecycle.

2.3. Automated testing frameworks for enterprise systems

Automated software testing is essential for maintaining quality in large systems, yet it also represents a significant share of development effort. Nagabushanam and co-authors reviewed the process of automated software testing and concluded that testing activities can consume more than half of total development cost in complex projects, making effective automation crucial [7]. Umar and Chen studied automated software testing tools and frameworks and described how they reduce manual regression effort while supporting continuous integration practices [8].

Gamido and Gamido provided a comparative review of automated software testing tools, examining features such as scripting models, integration capabilities, and reporting support [9]. Anjum and colleagues conducted a comparative analysis of quality assurance for mobile applications using automated testing tools, highlighting how differences in tool design influence coverage and defect detection [10]. For Java enterprise systems specifically, Gudi analyzed multiple automated testing frameworks and showed how framework selection affects reliability, maintainability, and integration with enterprise CI/CD pipelines [16].

This body of work shows that tool choice, test strategy, and pipeline integration all have substantial influence on the effectiveness of automated testing in practice.

2.4. Gap and motivation

The literature reviewed above points to three complementary streams: decision intelligence and AI-assisted project management, machine-learning-based defect prediction, and automated testing frameworks. Many organizations adopt elements from each stream, for example by using analytics dashboards, training defect prediction models, and automating unit or integration tests. However, these elements are often introduced in isolation.

There is comparatively little published work on architecture-centered frameworks that integrate decision intelligence with defect prediction and automated testing in a way that directly supports agile governance. The remainder of this paper addresses this gap by proposing an integrated framework, DI-AGF, and by outlining how it can be applied in a realistic enterprise context.

3. Decision-Intelligent Agile Governance Framework (DI-AGF)

3.1. Conceptual overview

DI-AGF conceptualizes the software lifecycle as a closed loop that links data, analytics, decisions, and actions. The framework distinguishes four architectural layers:

- Observability and data engineering – collects and prepares lifecycle data.
- Analytics and machine learning – builds and serves predictive and descriptive models.
- Decision intelligence and policy – encodes governance decisions as explicit models.
- Execution and feedback – enacts decisions through pipelines and workflows and captures outcomes.

Decisions such as “Can this service be released?”, “Which tests must run for this change set?”, or “Which refactoring items should be scheduled next quarter?” are modelled as decision assets with defined inputs, decision logic, outputs, and outcome metrics. This is consistent with the broader decision-intelligence literature that emphasizes engineered decision models and continuous improvement via feedback [14].

3.2. Observability and data engineering layer

The observability and data engineering layer aggregates information from source-code repositories, build CI/CD systems, test frameworks, issue trackers, and production monitoring platforms. Typical data sources include:

- Code metrics and change histories from version-control systems.
- Static-analysis findings and code smell reports.
- Test execution results at unit, integration, API, UI, and performance levels.
- Recorded defects, severities, and root-cause analyses from issue trackers.
- Operational metrics such as error rates, latency distributions, and resource utilization.

Data from these heterogeneous sources is cleaned, aligned, and stored in an analytical repository at granularity levels such as per module, per service, or per release. Data quality management and lineage tracking are essential, because the reliability of downstream analytics and decisions depends on the integrity of this foundation.

3.3. Analytics and machine-learning layer

The analytics and machine-learning layer builds models that characterize risk and behavior in the system. Building on defect prediction research [2] and [3], this layer may include models that estimate the probability that a given module will exhibit a defect in production. Additional models can forecast delivery lead times, estimate the impact of test execution subsets, or cluster services by operational behavior.

A simple composite risk score for a module can be defined as a weighted combination of model outputs and architectural attributes:

$$R(m) = w1 \cdot P\_defect(m) + w2 \cdot C(m) + w3 \cdot B(m),$$

where P_defect(m) is the predicted defect probability, C(m) is a structural complexity metric, B(m) reflects business criticality, and w1, w2, w3 are weights agreed with stakeholders. Explainable machine-learning techniques, such as those described by Santos and colleagues [5], can be used to show why a given module receives a high risk score.

3.4. Decision-intelligence and policy layer

In the decision-intelligence and policy layer, governance decisions are formalized as decision models. Each model defines:

- Inputs (e.g., risk scores, coverage metrics, incident histories).
- Decision logic (e.g., rules, optimization algorithms, thresholds).

- Outputs (e.g., “approve release”, “require additional testing”, “escalate to architect”).
- Outcome measures (e.g., post-release defect rate, number of emergency rollbacks).

For example, a release-readiness decision may require that all critical services satisfy both a maximum tolerated risk score and a minimum test coverage level. A test-scope selection decision may choose an optimal subset of tests under time-budget constraints, using risk coverage as an objective. An architecture-refactoring decision may prioritize components that exhibit persistent high risk and incident density.

These decision models are aligned with organizational policies and can be created using decision-modelling notations or governed through decision catalogs similar to the ones proposed in earlier work on agile lifecycle governance [1].

3.5. Execution and feedback layer

The execution and feedback layer integrates decision models with delivery pipelines and operational workflows. Continuous-integration and deployment pipelines query the decision-intelligence layer before executing certain stages. For instance, before deploying to a staging or production environment, a pipeline may request a release-readiness decision. Depending on the outcome, the pipeline might proceed automatically, trigger additional automated tests, create follow-up tasks, or block the deployment pending manual review.

Outcomes, such as escaped defects, incidents, rollbacks, and deviations from expected behavior, are logged and fed back into the observability layer. This feedback allows models and policies to be recalibrated over time, closing the decision loop.

4. Integration of Defect Prediction and Automated Testing

4.1. Risk-driven test selection

One important application of DI-AGF is risk-driven test selection. When a new change set is proposed, the observability and analytics layers determine which modules and services are affected and compute updated risk scores using defect prediction models [2]. The decision-intelligence layer then selects a subset of available automated tests that provides high coverage of high-risk components while respecting time and resource limits.

Formally, this can be framed as an optimization problem in which the objective is to maximize risk coverage subject to a constraint on total test execution time. In practice, heuristics such as greedy selection based on marginal risk coverage or simple rules that require all tests touching critical high-risk services to run can provide good results without excessive computational overhead.

4.2. Feedback from test outcomes

Test outcomes provide valuable feedback for both defect prediction models and governance policies. If an automated test fails, and the failure is confirmed as a genuine defect, the associated module serves as a positive training instance for future versions of the prediction model. Patterns in failed tests may also reveal systematic weaknesses in parts of the test suite or highlight areas where test data is insufficient.

Nagabushanam and co-authors emphasized that the testing process itself must be monitored and improved, as flakiness, poor test design, or unstable environments can undermine trust in automation [7]. The DI-AGF framework therefore treats indicators such as test flakiness, historical instability, and coverage gaps as first-class signals within the decision models.

4.3. Architectural implications for Java enterprise systems

For Java enterprise systems, DI-AGF assumes the existence of automated unit and integration tests, API tests, and possibly UI and performance tests implemented using commonly adopted frameworks such as those reviewed by Umar and Chen [8], Gamido and Gamido [9], and Anjum and colleagues [10]. Building on Gudi’s analysis of automated testing frameworks in Java enterprise environments [16], framework selection is treated as an architectural decision that affects the observability, controllability, and reliability of the overall system.

Frameworks that provide rich metadata, machine-readable reports, and flexible tagging of tests by service or feature are particularly valuable, because they enable finer-grained risk-driven test selection and more accurate feedback loops.

5. Illustrative Scenario

Consider a hypothetical organization operating a large Java microservices platform that supports web and mobile applications for retail customers. The system uses tens of Spring Boot services, each with its own CI/CD pipeline, and relies on automated unit and API tests, static analysis, and centralized logging and metrics.

Using DI-AGF, the organization first constructs a decision catalog for key governance decisions such as release readiness and test scope selection. The observability layer aggregates data from version control, build pipelines, test frameworks, defect

trackers, and monitoring systems. The analytics layer trains a supervised defect prediction model on historical data, using features inspired by defect prediction surveys [2] and [3].

Before each release, the release-readiness decision model evaluates the risk scores and test coverage of all services. Services with risk scores above a threshold or with insufficient test coverage are either subjected to additional automated tests or flagged for manual code review. The test-scope decision model uses risk information to choose which subsets of integration and API tests must run for a given release candidate, trading off risk coverage against limited test environment capacity.

After deployment, production incidents and post-release defects are recorded and linked back to the decision context that approved the release. Over time, this feedback supports calibration of risk thresholds, refinement of policies, and retraining of defect prediction models. Qualitative feedback from architects and team leads can also be collected to assess whether the decision models align with practitioner expectations.

6. Evaluation Strategy and Research Agenda

Evaluating DI-AGF requires both quantitative and qualitative evidence. Quantitative metrics can include post-release defect density, mean time between incidents, deployment frequency, rollback rate, test execution time, and adherence to decision recommendations. These metrics can be compared before and after adoption of DI-AGF or between teams that do and do not use the framework.

Qualitative studies can explore how practitioners perceive the transparency and usefulness of decision models. Prior work on AI-assisted project management showed that introducing AI tools changes collaboration patterns and decision practices [13]. Similar investigations are needed for decision-intelligence frameworks in software engineering, with attention to issues such as trust in predictive models, perceived fairness of automated policies, and the balance between automation and human judgment.

Future research questions include how to automatically tune decision policies based on outcome data, how to share decision models across projects, and how to articulate ethical guidelines for the use of predictive models in governance decisions.

7. Conclusion

This paper presented the Decision-Intelligent Agile Governance Framework (DI-AGF), an architecture-centered model that integrates decision intelligence, machine-learning-based software defect prediction, and automated testing to support agile software lifecycle governance. The framework builds on earlier work on decision-intelligence-driven governance [1], on the expanding role of machine-learning models in software development [15], and on comparative analyses of automated testing frameworks in Java enterprise systems [16]. DI-AGF structures the lifecycle into four layers, treats governance decisions as explicit assets, and links risk analytics to testing and release decisions through feedback loops.

The proposed framework is intended as a conceptual reference that organizations can adapt to their own technology stacks and governance cultures. Empirical validation through industrial case studies remains an important direction for future work. Nevertheless, DI-AGF illustrates how decision intelligence, when combined with software analytics and automation, can help organizations reason more systematically about quality and risk in agile delivery environments.

References

[1] Gunda SK, Yettapu SDR, Bodakunti S, Bikki SB. Decision Intelligence Methodology for AI-Driven Agile Software Lifecycle Governance and Architecture-Centered Project Management. International Journal of Artificial Intelligence and Data Science in Machine Learning, 4(1), 102–108, 2023. https://doi.org/10.63282/3050-9262.IJAIDSML-V4I1P112

[2] Omri S, Sinz C. Deep Learning for Software Defect Prediction: A Survey. In: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, 2020, pp. 209–214. https://doi.org/10.1145/3387940.3391463

[3] Akimova EN, Bersenev AY, Deikov AA, Kobylkin KS, Konygin AV, Mezentsev IP, Misilov VE. A Survey on Software Defect Prediction Using Deep Learning. Mathematics, 9(11):1180, 2021. https://doi.org/10.3390/math9111180

[4] Pandey S, Kumar K. Software Fault Prediction for Imbalanced Data: A Survey on Recent Developments. Procedia Computer Science, 218, 1815–1824, 2023. https://doi.org/10.1016/j.procs.2023.01.159

[5] Gudi SR. Enhancing Reliability in Java Enterprise Systems through Comparative Analysis of Automated Testing Frameworks. International Journal of Emerging Trends in Computer Science and Information Technology, 4(2), 151–160, 2023. https://doi.org/10.63282/3050-9246.IJETCSIT-V4I2P115

[6] Stradowski S, et al. Machine Learning in Software Defect Prediction: A Business Perspective. Information and Software Technology, 2023.

[7] Nagabushanam D.S., Dharinya S., Vijayasree D., Roopa N.S., Arun A. A Review on the Process of Automated Software Testing. arXiv preprint arXiv:2209.03069, 2022. https://doi.org/10.48550/arXiv.2209.03069

[8] Umar MA, Chen Z. A Study of Automated Software Testing: Automation Tools and Frameworks. International Journal of Computer Science Engineering, 8(6), 217–225, 2019. https://doi.org/10.5281/zenodo.3924795
[9] Gamido HV, Gamido MV. Comparative Review of the Features of Automated Software Testing Tools. International Journal of Electrical and Computer Engineering, 9(5), 4473–4478, 2019. https://doi.org/10.11591/ijece.v9i5.pp4473-4478
[10] Anjum H, Babar MI, Jehanzeb M, Khan M, Chaudhry S, Sultana S, Shahid Z, Zeshan F, Bhatti SN. A Comparative Analysis of Quality Assurance of Mobile Applications Using Automated Testing Tools. International Journal of Advanced Computer Science and Applications, 8(7), 2017. https://doi.org/10.14569/IJACSA.2017.080733
[11] Bae GH, Kim JK, Park SH. A Machine Learning-Based Decision Support System for Project Management. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 51(2), 814–823, 2021.
[12] Chan AH, Yang LH, Ho TT. Predictive Analytics in Project Management: An AI-Driven Approach. IEEE Access, 9, 1332–1343, 2021.
[13] Thota S, et al. AI-Assisted Project Management: Enhancing Decision-Making and Forecasting. Journal of Artificial Intelligence Research, 3, 146–171, 2023.
[14] Gunda SKG. The Future of Software Development and the Expanding Role of ML Models. International Journal of Emerging Research in Engineering and Technology, 4(2), 126–129, 2023. https://doi.org/10.63282/3050-922X.IJERET-V4I2P113
[15] Marianne O. Decision Intelligence: Why It Is a Life-Mattering Skill and How We Can Make Better Decisions. DataDrivenInvestor, 2023.