INTERNATIONAL MULTI DISCIPLINARY JOURNAL FOR RESEARCH & DEVELOPMENT
eISSN 2394-6334 Impact factor: 7,854 Volume 13, issue 08 (2026)

AI-Driven Automated Defect Prediction and Software Testing for Quality Improvement

Budi Santoso
Department of Artificial Intelligence, Institute of Computational Technology, Jakarta, Indonesia

Abstract: The increasing complexity of contemporary software systems has intensified the need for automated mechanisms capable of identifying defects before they propagate into later stages of development and deployment. Conventional testing approaches remain valuable, but their effectiveness can be constrained by the scale of software artifacts, heterogeneous execution environments, and the increasing number of possible test conditions. This research develops a conceptual AI-driven framework for automated defect prediction and software testing that integrates software-data preprocessing, defect-risk prediction, intelligent test prioritization, automated test execution, feedback analysis, and continuous model refinement. The methodology positions defect prediction as a risk-estimation problem in which historical and structural characteristics of software artifacts are transformed into actionable testing priorities. The theoretical design also draws on principles of adaptive control, feedback-driven decision making, and intelligent automation represented in the supplied literature. Particular attention is given to the transferability of these principles into software quality engineering, where prediction and testing should operate as a closed feedback loop rather than as isolated activities. The proposed framework is evaluated conceptually through its expected effects on defect identification, testing efficiency, prioritization, regression coverage, and quality improvement. The analysis indicates that AI can provide the greatest value when prediction outputs are directly connected to test-selection and execution mechanisms. However, model bias, inadequate training data, false positives, explainability, and domain shift remain significant limitations. The study concludes that AI-driven testing should augment rather than completely replace engineering judgment and should be implemented through continuously monitored, feedback-oriented quality processes.

Keywords: Artificial Intelligence, Defect Prediction, Automated Software Testing, Machine Learning, Deep Learning, Quality Assurance, Test Prioritization, Software Quality Engineering, Predictive Analytics

INTRODUCTION

Software systems are increasingly characterized by large codebases, frequent releases, distributed architectures, and continuous modification. Under these conditions, software quality assurance cannot depend exclusively on manually designed test cases or sequential testing activities. A defect that remains unidentified during development can propagate through subsequent software versions, increase debugging effort, and affect system reliability. Consequently, defect prediction and automated testing have become complementary quality-engineering activities: prediction estimates where defects are likely to occur, while testing provides the mechanism through which those risks can be examined.

The central research problem addressed in this study is the absence of an integrated conceptual mechanism connecting defect-risk estimation with automated test decision making. A prediction model that merely labels software components as defective or non-defective has limited operational value if its output does not

influence testing priorities. Conversely, an automated testing system can execute large numbers of tests without necessarily concentrating computational resources on the software regions presenting the greatest predicted risk. An effective AI-driven approach therefore requires an explicit relationship between prediction, prioritization, execution, observation, and learning.

The supplied literature, although primarily concerned with intelligent control, robotics, adaptive mechanisms, and neural-network-based control, provides useful theoretical analogies for this integration. Miller demonstrated the use of neural networks for real-time control of a biped walking robot, illustrating the broader principle that learned models can transform system observations into control decisions (Miller, 1994). Similarly, adaptive control research demonstrates the importance of adjusting system behavior according to changing conditions rather than relying exclusively on fixed control rules (Jatsun et al., 2015). These principles can be conceptually transferred to software testing, where changing code, defect patterns, and test outcomes require dynamic prioritization.

The objective of this research is therefore to design a conceptual AI-driven automated defect prediction and software-testing framework. The study specifically investigates how an AI prediction layer can be connected to test-case prioritization, automated execution, defect classification, and feedback-based model improvement. The framework is intended for software quality environments in which testing resources are limited and where testing must be increasingly selective, repeatable, and responsive.

The research is significant because it treats defect prediction and automated testing as components of a unified quality-improvement system. Instead of viewing AI as an isolated classification technology, the proposed approach considers AI as a decision-support mechanism embedded within the testing lifecycle. This perspective is consistent with the principle of adaptive interaction observed in human-centred robotic control, in which system behavior is influenced by ongoing interaction and dynamic feedback (Shi et al., 2021).

LITERATURE REVIEW

2.1 Intelligent Models and Automated Decision Making

The supplied literature establishes a broad theoretical foundation for intelligent, adaptive, and automated system behavior. Miller's work on real-time neural-network control demonstrates how computational models can process system information and generate decisions in an operational environment (Miller, 1994). Although the application concerns robotic locomotion rather than software engineering, the underlying principle is relevant to AI-based testing: an intelligent model can convert observed characteristics into decisions without requiring every decision rule to be explicitly programmed.

Jatsun et al. investigated adaptive control for an exoskeleton performing sit-to-stand motion, emphasizing system adaptation during a changing operational task (Jatsun et al., 2015). In software testing, an analogous principle is the dynamic modification of test priorities according to defect predictions and newly observed failures. A fixed testing sequence may remain operational, but an adaptive sequence can allocate greater attention to components whose estimated risk changes over time.

The distinction between fixed and adaptive decision making is particularly important for continuous software development. Software changes can alter defect distributions, dependencies, and test relevance. Consequently, a quality-assurance mechanism based entirely on historical static priorities may become progressively less effective.

2.2 Feedback and Closed-Loop Control as a Theoretical Foundation

Several supplied studies focus on modeling, measurement, control, and interaction. Zhou et al. presented a precision measuring mechanism intended for closed-loop control of a biologically inspired lower-limb exoskeleton (Zhou et al., 2018). The concept of closed-loop operation is particularly relevant to AI-driven testing because a testing system should not terminate its operation after generating a prediction. Instead, test results should return to the analytical layer and influence subsequent predictions and testing decisions.

A conceptual software-quality loop can therefore be expressed as:

Software artifacts → Feature extraction → Defect prediction → Test prioritization → Automated execution → Failure analysis → Feedback → Model refinement

This structure differs fundamentally from a conventional one-directional testing workflow. Prediction becomes an active component of test management rather than merely a reporting function.

The literature also demonstrates that complex systems benefit from explicit consideration of interaction and changing conditions. Human-centred adaptive control incorporates interaction dynamics into control behavior (Shi et al., 2021). By analogy, software testing can incorporate interactions among code modules, test cases, historical failures, dependency structures, and changing versions.

2.3 Adaptive and Environment-Aware System Behavior

Wu and Popović investigated terrain-adaptive bipedal locomotion control, demonstrating the conceptual importance of adapting system behavior to environmental conditions (Wu and Popovi´c, 2010). In software quality engineering, the equivalent environment includes programming languages, architectural structures, development practices, deployment configurations, and release patterns.

A defect prediction model trained on one software environment may not perform identically when transferred to another. This creates a domain-adaptation problem. Consequently, AI-based testing should incorporate mechanisms for monitoring prediction reliability rather than assuming that a model remains permanently accurate.

The adaptive principle is also visible in research involving the optimization of robotic mechanisms. Zhang et al. examined optimization of a rotational asymmetric parallel mechanism for rehabilitation applications (Zhang et al., 2020). In the software-testing context, optimization can be interpreted as allocating finite testing resources to the most informative or risky test activities.

2.4 Automation, System Modeling, and Research Gap

Research on robotic exoskeletons further demonstrates the value of system modeling before automated intervention. Veneman et al. addressed design and evaluation of an exoskeleton robot for interactive gait rehabilitation (Veneman et al., 2007), while Esquenazi et al. examined the ReWalk powered exoskeleton as a mechanism for restoring ambulatory function (Esquenazi et al., 2012). These works differ substantially from software engineering, but they reinforce a broader engineering principle: intelligent automation requires structured modeling of the target system and evaluation of its operational behavior.

Shi et al. provided a review of lower-limb rehabilitation exoskeleton robots and emphasized the evolution of robotic mechanisms toward increasingly sophisticated rehabilitation functions (Shi et al., 2019). Jatsun et al. investigated theoretical and experimental characteristics of a mobile robotic gait system, demonstrating the relationship between theoretical modeling and experimental validation (Jatsun et al., 2015).

The major research gap emerging from the supplied literature is therefore not a direct gap in software-testing research, because the references themselves primarily address robotics and intelligent control. Rather, the gap addressed in this paper is the conceptual transfer of adaptive, neural, measurement-oriented, and closed-loop engineering principles into an integrated AI-based software-testing architecture. This distinction is important because the framework proposed here is a research model rather than a claim that the supplied robotic studies directly evaluated software-testing systems.

METHODOLOGY

3.1 Research Design

This study adopts a conceptual framework-development methodology. The objective is to construct an AI-driven architecture that connects defect prediction with automated testing through a feedback loop. The methodology is organized into six functional stages: software-data acquisition, feature engineering, defect-risk prediction, intelligent test prioritization, automated execution and diagnosis, and feedback-based model refinement.

The framework is designed around a risk score (R_i) for software component (i):

[

R_i = f(X_i)

]

where (X_i) represents the feature vector associated with the component and (f) represents the trained AI prediction function.

The resulting risk value does not constitute a definitive statement that a defect exists. Instead, it represents the model's estimated likelihood or priority level. This distinction is essential because prediction uncertainty must be incorporated into subsequent testing decisions.

3.2 Software Data Acquisition

The first stage collects software artifacts and testing information. Relevant information may include source-code characteristics, change history, complexity indicators, previous failure records, test execution outcomes, and component-level dependency information.

The objective is not to collect every available attribute but to identify features that can provide meaningful discrimination between higher- and lower-risk software components. A hypothetical system might identify ten modified modules in a release and assign substantially higher predicted risk to three modules because they contain extensive changes and have experienced repeated historical failures.

Data quality is a major methodological concern. Incomplete or inconsistently labeled defect records can introduce systematic bias into the prediction model. Therefore, preprocessing should include duplicate removal, missing-value analysis, normalization where required, and consistency checking of defect labels.

3.3 Feature Engineering

Feature engineering transforms raw software information into representations suitable for machine learning.

Static characteristics can describe the internal structure of software, whereas dynamic features can represent behavior observed during execution.

The proposed framework can classify features into four conceptual groups: structural, historical, behavioral, and testing-related features. Structural features describe software complexity and dependency relationships. Historical features represent previous modifications and defect occurrences. Behavioral features capture execution characteristics. Testing-related features represent prior test outcomes and coverage behavior.

This multidimensional representation is preferable to reliance on a single complexity measure because software defects can emerge from interactions among structural and behavioral properties.

3.4 AI-Based Defect Prediction

The prediction layer can employ supervised machine-learning or deep-learning models depending on dataset size, feature structure, and interpretability requirements. A binary classification formulation can be expressed as:

[

P(D_i=1|X_i)=\sigma(f(X_i))

]

where (D_i=1) indicates the predicted presence of a defect and (\sigma) converts the model output into a probability-like score.

For a large software organization, a deep-learning architecture could learn nonlinear relationships among software characteristics. However, a more complex model is not automatically superior. If explainability is essential, a simpler model may provide more actionable evidence to engineers.

The conceptual role of neural computation is supported by Miller's demonstration that neural networks can be incorporated into real-time control systems (Miller, 1994). For software testing, the equivalent role is to transform multidimensional software observations into predictions that influence subsequent decisions.

3.5 Intelligent Test Prioritization

The most important methodological contribution is the transformation of prediction into testing action. If (R_i) denotes defect risk and (C_j) denotes the estimated cost of executing test (j), test prioritization can be represented conceptually as:

[

Priority_j = \frac{Risk_j \times InformationValue_j}{Cost_j}

]

The equation is not intended as a universal optimization formula but as a decision model illustrating the relationship among risk, expected information, and testing resources.

High-risk and high-information tests should be executed earlier when testing time is constrained. For example, if a release contains 1,000 available regression tests but only 200 can be executed within a deployment

window, the AI system should prioritize tests associated with high-risk modified components rather than selecting the 200 tests arbitrarily.

This adaptive allocation reflects the broader optimization and adaptive-control principles represented in the supplied engineering literature (Zhang et al., 2020; Jatsun et al., 2015).

3.6 Automated Test Execution and Defect Diagnosis

After prioritization, selected tests are executed automatically through the available software-testing infrastructure. Execution outputs include pass/fail status, runtime behavior, error messages, logs, and environmental information.

The diagnostic layer groups failures according to probable root causes. Multiple failed tests may originate from one underlying defect, while a single test can sometimes fail because of an environmental condition rather than a software defect. Therefore, automated diagnosis should not equate every failed test with a unique defect.

A hypothetical example illustrates this distinction. Suppose a payment service is modified and the AI model assigns it high risk. The prioritization engine executes payment, authentication, transaction-validation, and rollback tests early. If three tests fail because of a common database interaction error, the diagnosis layer should attempt to consolidate the failures rather than reporting three independent defects.

3.7 Feedback and Model Refinement

The final stage returns execution outcomes to the prediction system. Confirmed defects, false positives, false negatives, and newly observed software characteristics can become future training information.

This creates a continuous feedback architecture:

[

Prediction \rightarrow Testing \rightarrow Observation \rightarrow Learning \rightarrow Prediction

]

The importance of feedback is theoretically consistent with closed-loop control mechanisms described in the supplied literature (Zhou et al., 2018). It also reflects adaptive system behavior in which decisions are modified according to changing observations (Shi et al., 2021).

The feedback mechanism should include safeguards against uncontrolled learning. Automatically incorporating every test result into training data could amplify erroneous labels. Human validation may therefore remain necessary for ambiguous failures.

3.8 Evaluation Framework

The proposed system should be evaluated using defect-prediction and testing-efficiency measures. Relevant indicators include precision, recall, F1-score, false-positive rate, defect detection rate, test-execution time, testing-resource utilization, and regression-test reduction.

A useful evaluation should compare at least two configurations: conventional test prioritization and AI-driven

prioritization. The comparison should determine whether AI identifies a greater proportion of confirmed defects within an equivalent testing budget.

The methodology also requires cross-version validation. A model that performs well only on the software version used during training cannot be considered sufficiently robust for practical deployment.

RESULTS

The conceptual evaluation of the proposed architecture indicates that the principal benefit of AI-driven testing is not simply increased automation but improved allocation of testing attention. When defect prediction is integrated with test prioritization, testing resources can theoretically be concentrated on software components presenting higher estimated risk. This provides a stronger quality-engineering mechanism than treating prediction and testing as independent processes.

The first expected finding is improved early defect identification. High-risk components can be tested earlier in the release cycle, allowing development teams to investigate potentially serious defects before deployment. The effectiveness of this benefit depends directly on prediction precision and recall. Excessive false positives would consume testing resources, whereas excessive false negatives could create a false perception of quality.

The second finding concerns testing efficiency. Automated prioritization can reduce unnecessary execution of low-value tests during constrained testing windows. However, reduction in execution volume should not be interpreted automatically as improved quality. A smaller test set is beneficial only when it preserves or improves meaningful defect-detection capability.

The third finding is the value of feedback. A closed-loop architecture allows actual test outcomes to modify future predictions. This can enable the system to respond to changing defect patterns rather than relying indefinitely on historical assumptions. The principle resembles adaptive engineering systems in which observed performance influences subsequent decisions (Jatsun et al., 2015; Shi et al., 2021).

The fourth finding concerns model transparency. Deep models may identify complex patterns, but software engineers require sufficient evidence to understand why a component has received a high-risk score. Consequently, the practical value of prediction depends not only on statistical accuracy but also on interpretability and engineering usefulness.

Finally, the framework suggests that AI-based software testing should be evaluated as a socio-technical system. The prediction model, testing infrastructure, software engineers, defect-management processes, and deployment environment collectively determine quality outcomes. The model cannot be considered independently from the system in which it operates.

DISCUSSION

The proposed framework demonstrates that AI-driven defect prediction is most valuable when it is connected directly to testing decisions. A prediction-only architecture creates an informational output, whereas a prediction-testing-feedback architecture creates an operational quality mechanism. This distinction represents the central theoretical contribution of the study.

The adaptive nature of the proposed framework is consistent with the broader engineering principles represented by the supplied literature. Miller's neural-network control research illustrates how learned representations can support operational decisions (Miller, 1994). Applied conceptually to software quality, the

same principle suggests that historical and behavioral software information can be transformed into risk-sensitive testing decisions. The framework therefore extends the idea of intelligent control from physical-system operation to software-quality management.

The proposed approach also aligns conceptually with closed-loop measurement and control. Zhou et al. examined a precision measuring mechanism associated with closed-loop control, emphasizing the relationship between measurement and subsequent control behavior (Zhou et al., 2018). In software testing, test results constitute the feedback signal. Without this signal, the prediction model cannot systematically respond to newly discovered defect patterns.

The reference Deep Learning-Based Automated Software Testing for Improved Quality Assurance is particularly relevant to the proposed architecture because it captures the intended relationship between deep learning, automated testing, and quality assurance. Within this study, that work is used as a conceptual supporting source for the integration of learning-based prediction with automated quality-assurance activities rather than as empirical evidence for specific performance values.

Nevertheless, several trade-offs must be recognized. First, predictive accuracy depends on training-data quality. Historical defects are not necessarily representative of future defects. Development practices, programming technologies, and architectural structures can change, producing distribution shifts. Second, false positives may cause testing teams to over-investigate components that are actually stable. Third, false negatives remain especially problematic because an apparently low-risk component may escape adequate testing.

Another limitation concerns explainability. A highly complex model may outperform a simpler model under certain conditions while providing less transparent reasoning. In safety-critical or regulated environments, explainability may be as important as raw predictive performance.

The supplied robotic literature also highlights the importance of adaptation to operational conditions. Terrain-adaptive control demonstrates the broader engineering principle that a system should respond to environmental variation rather than operate under rigid assumptions (Wu and Popovi´c, 2010). Software systems similarly evolve across versions and environments. Therefore, model monitoring should be treated as an ongoing requirement rather than a one-time validation activity.

The practical implication is that organizations should not implement AI testing as a replacement for conventional quality assurance. Instead, AI should augment existing testing infrastructure by identifying risk, prioritizing tests, detecting patterns, and providing feedback. Human engineers remain necessary for interpreting ambiguous failures, validating labels, reviewing unusual predictions, and determining release readiness.

The framework further indicates that future research should conduct empirical experiments using real software repositories. Such experiments should compare multiple prediction algorithms, investigate cross-project generalization, quantify the relationship between prediction accuracy and testing efficiency, and assess the consequences of false-positive and false-negative predictions under realistic release constraints.

CONCLUSION

This research presented a conceptual AI-driven framework for automated defect prediction and software testing aimed at improving software quality. The framework integrates data acquisition, feature engineering, AI-based risk prediction, intelligent test prioritization, automated execution, defect diagnosis, and feedback-

based model refinement.

The primary contribution is the conceptual integration of prediction and testing into a closed-loop quality-assurance process. Rather than treating defect prediction as an isolated classification task, the framework converts predicted risk into testing priorities and subsequently uses observed test outcomes to refine future predictions. This architecture draws theoretical support from adaptive control, neural-network decision making, optimization, measurement, and closed-loop engineering principles represented in the supplied literature.

The analysis indicates that AI can potentially improve defect detection efficiency and testing-resource allocation, particularly where software systems are large and testing windows are constrained. However, these benefits are conditional on reliable data, appropriate model validation, continuous monitoring, and careful treatment of uncertainty.

The future scope of this research includes empirical implementation across real software projects, comparative evaluation of machine-learning and deep-learning architectures, explainable AI for defect prediction, cross-project defect prediction, automated root-cause analysis, and integration with continuous integration and continuous delivery environments. Ultimately, the most effective AI-driven testing systems are likely to be those that combine computational intelligence with adaptive feedback and human engineering judgment.

REFERENCES

1. A. Esquenazi, M. Talaty, A. Packel and M. Saulino, “The ReWalk powered exoskeleton to restore ambulatory function to individuals with thoracic-level motor-complete spinal cord injury,” Am. J. Phys. Med. Rehabilit., vol. 91, no. 11, pp. 911–921, 2012.

2. A. Yatsun and S. Jatsun, “Modeling quasi-static gait of a person wearing lower limb exoskeleton,” in International Conference on Industrial Engineering, Cham: Springer, 2018, pp. 565–575.

3. D. Shi, W. Zhang, W. Zhang and X. Ding, “A review on lower limb rehabilitation exoskeleton robots,” Chin. J. Mech. Eng., vol. 32, no. 1, pp. 1–11, 2019.

4. D. Shi, W. Zhang, W. Zhang, L. Ju and X. Ding, “Human-centred adaptive control of lower limb rehabilitation robot based on human–robot interaction dynamic model,” Mech. Mach. Theory, vol. 162, pp. 104340, 2021.

5. J. C. Wu and Z. Popovi´c, “Terrain-adaptive bipedal locomotion control,” ACM Trans. Graphics (TOG), vol. 29, no. 4, pp. 1–10, 2010.

6. J. F. Veneman, R. Kruidhof, E. E. Hekman, R. Ekkelenkamp, E. H. Van Asseldonk and H. Van Der Kooij, “Design and evaluation of the LOPES exoskeleton robot for interactive gait rehabilitation,” IEEE Trans. Neural Syst. Rehabil. Eng., vol. 15, no. 3, pp. 379–386, 2007.

7. J. Pratt and G. Pratt, “Intuitive control of a planar bipedal walking robot,” in Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146), IEEE, 1998, Vol. 3, pp. 2014–2021.

8. L. Zhou, W. Chen, J. Wang, S. Bai, H. Yu and Y. Zhang, “A novel precision measuring parallel mechanism for the closed-loop control of a biologically inspired lower limb exoskeleton,”

IEEE/ASME Trans. Mechatr. vol. 23, no. 6, pp. 2693–2703, 2018.

9. S. H. Collins, A. Ruina, "A bipedal walking robot with efficient and human-like gait,", in Proceedings of the 2005 IEEE international conference on robotics and automation, IEEE, 2005, pp. 1983–1988.

10. S. H. Collins, M. Wisse and A. Ruina, "A three-dimensional passive-dynamic walking robot with two legs and knees," Int. J. Robotics Res., vol. 20, no. 7, pp. 607–615, 2001.

11. S. Jatsun, L. Vorochaeva, A. Yatsun and A. Malchikov, "Theoretical and experimental studies of transverse dimensional gait of five-link mobile robot on rough surface," in 2015 10th International Symposium on Mechatronics and its Applications (ISMA), 2015, pp. 1–6.

12. S. Jatsun, S. Savin, A. Yatsun and R. Turlapov, "Adaptive control system for exoskeleton performing sit-to-stand motion," in 2015 10th International Symposium on Mechatronics and Its Applications (ISMA). IEEE, 2015, pp. 1–6.

13. W. T. Miller, "Real-time neural network control of a biped walking robot," IEEE Control Syst. Mag., vol. 14, no. 1, pp. 41–48, 1994.

14. W. Zhang, W. Zhang, X. Ding and L. Sun, "Optimization of the rotational asymmetric parallel mechanism for hip rehabilitation with force transmission factors," J. Mech. Robotics, vol. 12, no. 4, 2020.

15. Philip, P. G. (2024). Artificial Intelligence-Driven Project Risk Prediction Models: Enhancing Decision-Making Accuracy in Large-Scale Infrastructure Projects . American Journal of Technology, 3(1), 52–69. https://doi.org/10.58425/ajt.v3i1.571

16. 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.