International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 06 ISSUE 08 Pages: 150-160
OCLC – 1368736135

Journal Website: http://sciencebring.com/index.php/ijasr

Research Article

AI-Assisted Defect Detection and Test Prioritization in Software Quality Engineering

Submission Date: June 08, 2026, **Accepted Date:** July 05, 2026,
Published Date: August 18, 2026
Crossref doi: https://doi.org/10.37547/ijasr-06-08-05

Mio Watanabe
Department of Computer Science and Machine Intelligence, Advanced Technology Research Center, Nagoya, Japan

Abstract

Artificial intelligence (AI)-assisted software quality engineering is increasingly concerned with reducing the cost of defect identification while improving the efficiency with which limited testing resources are allocated. This research develops a conceptual framework for AI-assisted defect detection and test prioritization by synthesizing the methodological and technological patterns evident in the provided literature. Although the supplied studies are predominantly situated in augmented reality (AR), simulation-based learning, engineering visualization, embedded monitoring, and technology-enabled experimentation, they collectively demonstrate principles relevant to intelligent quality assurance: automated observation, simulation, data-driven classification, prioritization of critical conditions, and continuous feedback. The proposed framework integrates data acquisition, defect-feature extraction, AI-assisted classification, risk-aware test prioritization, execution feedback, and iterative model refinement. Particular attention is given to the distinction between defect detection and test prioritization, since accurate identification alone does not guarantee efficient testing. The analysis further positions project-risk prediction as a complementary decision-support mechanism, emphasizing how predictive reasoning can improve prioritization decisions (Philip, 2024). The resulting framework provides a conceptual basis for incorporating AI into modern software quality engineering while recognizing limitations associated with heterogeneous data, domain transfer, model interpretability, validation, and the absence of empirical software-testing datasets within the supplied literature.

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 06 ISSUE 08 Pages: 150-160
OCLC – 1368736135

ISSN-2750-1396

KEYWORDS

Artificial Intelligence, Defect Detection, Test Prioritization, Software Quality Engineering, Machine Learning, Predictive Analytics, Automated Testing, Risk-Based Testing, Simulation, Quality Assurance

INTRODUCTION

Software quality engineering has progressively moved from conventional end-stage testing toward continuous quality assurance in which defects, risks, and test requirements are evaluated throughout the development lifecycle. The increasing complexity of software systems makes exhaustive execution of all available test cases impractical, particularly when applications contain large numbers of functional paths, interfaces, configurations, and dynamically changing components. Consequently, two related but distinct problems become central: identifying potentially defective behaviour and determining which tests should receive priority.

AI-assisted quality engineering addresses these problems by using computational models to extract patterns from available testing and operational data. A defect-detection model may classify an execution as normal or anomalous, whereas a prioritization model determines which tests should be executed first according to predicted risk, failure probability, historical evidence, or business importance. These functions are complementary. A highly accurate detector may still provide limited practical value if the testing system executes low-risk tests before critical scenarios. Conversely, an effective prioritization mechanism can improve resource utilization even when defect predictions remain imperfect.

The provided literature does not directly constitute a conventional software-testing corpus. Instead, it contains studies concerning AR-based learning, simulation, visualization, embedded monitoring, and engineering applications. Nevertheless, several technological principles are transferable to software quality engineering. For example, simulation-based learning demonstrates the value of controlled environments for representing complex conditions (Kim et al., 2021), while engineering visualization studies demonstrate how computational representations can expose relationships that are difficult to observe directly (Huang et al., 2017). Similarly, wireless monitoring and STM8-based systems illustrate the importance of continuous acquisition of operational information (Liu, 2021).

The relevance of predictive decision support is also apparent in risk-oriented computational approaches. Philip (2024) demonstrates how AI-driven prediction can support more accurate decision-making by estimating project risks before they become critical. In software quality engineering, the same conceptual principle can be applied to testing: predicted defect risk can

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 06 ISSUE 08 Pages: 150-160
OCLC – 1368736135

become an input for determining test priority rather than relying exclusively on static test ordering.

Accordingly, this paper proposes a conceptual AI-assisted defect detection and test prioritization framework. Its objectives are to establish a theoretical relationship between intelligent defect detection and risk-based test ordering, synthesize transferable technological principles from the provided studies, identify the principal components of an AI-assisted testing pipeline, and critically examine its expected benefits and limitations.

LITERATURE REVIEW

The provided literature demonstrates a broad progression from conventional technological interaction toward intelligent, simulation-supported, and data-oriented systems. Bogusevschi et al. (2020) examined combined virtual reality and virtual laboratory environments for physics education, illustrating how computational environments can reproduce experimental conditions. From a quality-engineering perspective, simulation is valuable because software behaviours can similarly be exercised under controlled conditions before deployment. The limitation is that a simulated environment can only represent the conditions encoded into its model; therefore, simulation quality directly affects the reliability of subsequent conclusions.

Cai et al. (2021) investigated AR-supported physics learning and its relationship with students' self-efficacy and conceptions of learning. While the application domain differs from software testing, the study demonstrates that technological systems can alter how users interact with complex information. This principle is relevant to quality engineering because AI-generated defect indicators should not simply replace human judgment; rather, they should present actionable evidence that enables engineers to make better testing decisions.

The engineering visualization research of Huang et al. (2017) is particularly relevant to computational quality analysis. Visualization and interaction with finite element analysis in AR demonstrate the value of transforming complex computational outputs into interpretable representations. In software testing, defect probability, execution traces, dependency relationships, and test-risk scores can similarly be transformed into interpretable decision-support information.

Garon et al. (2016) explored real-time high-resolution 3D data using HoloLens, highlighting the importance of real-time data acquisition. An AI-assisted testing environment similarly requires timely collection of execution logs, test outcomes, code-change information, and runtime indicators. Without sufficiently current data, a prioritization model may recommend tests based on obsolete system conditions.

Simulation and computational learning are also evident in Kim et al. (2021), whose work on trustworthy building fire detection used simulation-based learning. The significance for

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 06 ISSUE 08 Pages: 150-160
OCLC – 1368736135

ISSN-2750-1396

software quality engineering lies in the concept of learning from controlled representations of potentially critical conditions. A testing system can construct representative failure scenarios and use their characteristics to improve classification. However, simulated conditions must be validated against real-world behaviour to avoid a model that performs well synthetically but poorly during actual execution.

Harun, Tuli, and Mantri (2020) examined AR-supported experimentation and analyzed its impact on learning. Their work reinforces the importance of interactive technology in representing technical processes, while Hedenqvist et al. (2021) demonstrated the application of AR to mechanics learning. Together, these studies suggest that complex technical states can be represented through computational environments, supporting the broader theoretical foundation for AI-assisted analysis.

The remaining studies reinforce this technological pattern. Cui et al. (2020) investigated an STM8-based greenhouse shutter controller, while Liu (2021) addressed wireless low-power temperature and humidity monitoring using STM8. These systems emphasize automated sensing, data acquisition, and device-level monitoring. Such principles are transferable to software observability, where execution information can be continuously captured and transformed into quality indicators.

Long et al. (2021) used AR and simulation-based assets for a mechanical stress experiment, providing another example of integrating simulated representations with interactive analysis. Lu et al. (2022) examined electromagnetic–thermal coupling, demonstrating the need to consider interactions among multiple system factors. Software defects likewise rarely arise from a single isolated variable; they may result from interactions among components, configurations, dependencies, and execution states.

Mota et al. (2018) addressed AR mobile application development, while Osadchyi et al. (2021) discussed AR technologies for STEM education. These works support the broader argument that intelligent technological environments require integration between computational functionality and user-facing interaction. Finally, Sahin and Yilmaz (2020) examined AR technology's effect on student achievement and attitudes, indicating that technological effectiveness depends not only on technical capability but also on how users interact with and interpret the system.

A significant research gap emerges from this synthesis. The supplied studies provide substantial evidence concerning simulation, visualization, automated monitoring, and intelligent technological interaction, but they do not directly evaluate AI-based software defect detection or test prioritization. Therefore, the present framework should be interpreted as a theoretically grounded conceptual synthesis rather than as an empirically validated software-testing model. This distinction is essential for avoiding unsupported claims of performance.

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 06 ISSUE 08 Pages: 150-160
OCLC – 1368736135

METHODOLOGY

3.1 Research Design and Conceptual Foundation

This research adopts a conceptual research-and-review methodology. Rather than constructing a new empirical dataset, it synthesizes technological principles from the provided references and maps them onto the software quality-engineering problem. The methodology is therefore structured around functional decomposition: data acquisition, feature representation, defect detection, risk estimation, test prioritization, execution feedback, and model refinement.

The framework assumes that a software testing environment continuously produces heterogeneous observations. These may include historical test outcomes, execution traces, changed modules, error indicators, dependency information, runtime anomalies, and previous defect records. The conceptual architecture converts these observations into feature representations and then assigns each testing condition a defect likelihood and priority score.

3.2 Data Acquisition Layer

The first component is a data-acquisition layer. The importance of continuous monitoring is consistent with Liu's (2021) work on wireless temperature and humidity monitoring and Cui et al.'s (2020) controller-oriented architecture. Although their domain is physical systems, the transferable principle is that reliable decision-making requires systematic collection of current observations.

In a software environment, the acquisition layer can collect test execution results, failure logs, runtime exceptions, code-change information, and historical defect indicators. Data should be timestamped and associated with relevant software components so that the AI model can distinguish recent behaviour from outdated evidence.

3.3 Feature Extraction and Representation

Raw execution data are generally unsuitable for direct prioritization. The system therefore converts observations into meaningful features. Examples include historical failure frequency, recent code modification intensity, dependency centrality, execution frequency, severity of previous failures, and similarity to previously defective scenarios.

The conceptual importance of representation is supported by engineering visualization studies. Huang et al. (2017) showed how complex computational analysis can be represented interactively. Similarly, a software quality model should transform complex execution information into structured representations that can be analyzed by machine-learning models and interpreted by engineers.

3.4 AI-Assisted Defect Detection

The defect-detection component estimates whether an observed software condition is likely

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 06 ISSUE 08 Pages: 150-160
OCLC – 1368736135

to contain an error. A general classification model can be expressed conceptually as:

$$D(x) = P(\text{Defect} \mid x)$$

where x represents the extracted feature vector and D(x) represents the predicted probability of a defect.

The model may use historical execution data to distinguish normal and abnormal patterns. Simulation-based learning provides a theoretical basis for constructing controlled examples of critical conditions (Kim et al., 2021). However, synthetic or simulated examples should not be assumed to represent all production failures.

A practical example is a regression-testing system in which a recently modified payment component generates execution behaviour similar to previously defective versions. The detector can assign a relatively high defect probability to associated test scenarios. This prediction does not prove the presence of a defect; it indicates that the test deserves greater investigative attention.

3.5 Risk Estimation

Defect probability alone is insufficient for test prioritization. A low-probability failure in a safety-critical component may deserve greater attention than a high-probability defect in a low-impact feature. Therefore, the framework incorporates risk as a multidimensional concept.

A conceptual risk score can be represented as:

$$R_i = P_i \times I_i \times E_i$$

where P_i is predicted defect probability, I_i represents potential impact, and E_i represents exposure or relevance of the affected functionality.

This reasoning aligns conceptually with AI-driven project risk prediction, where predictive models support improved decision-making under uncertainty (Philip, 2024). In software quality engineering, risk prediction can therefore complement defect classification by translating model outputs into actionable testing priorities.

3.6 Test Prioritization

The prioritization layer ranks available test cases according to estimated risk and expected testing value. A generalized priority function may be represented as:

$$\text{Priority}(T_i) = w_1 D_i + w_2 R_i + w_3 C_i + w_4 F_i$$

where D_i denotes defect likelihood, R_i risk, C_i change relevance, and F_i historical failure evidence; the w terms represent configurable weights.

This approach avoids treating all tests equally. For example, if 1,000 regression tests are available but only 150 concern recently modified and historically unstable components, those tests can receive higher priority. The remaining tests are not discarded; they are scheduled according to their relative priority.

3.7 Simulation and Feedback

Simulation provides a controlled environment for evaluating model responses before deployment.

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 06 ISSUE 08 Pages: 150-160
OCLC – 1368736135

ISSN-2750-1396

The literature repeatedly demonstrates the utility of simulated or computational environments for technical experimentation (Bogusevschi et al., 2020; Long et al., 2021). In software testing, simulation can be used to generate representative failure scenarios and examine whether the prioritization model responds appropriately.

After test execution, actual outcomes are returned to the learning system. Correct predictions strengthen the evidence supporting the model, while false positives and false negatives expose weaknesses. This feedback cycle is essential because software systems evolve continuously.

3.8 Human Oversight and Interpretability

AI should function as decision support rather than an unquestionable authority. Visualization research demonstrates the importance of making complex computational results understandable to users (Huang et al., 2017). Accordingly, a quality-engineering dashboard should expose not only the priority score but also the principal factors contributing to it.

For example, a test could be ranked highly because the associated module was recently modified, has a high historical failure rate, and interacts with a critical subsystem. Such explanations allow engineers to validate whether the recommendation is technically reasonable.

Results

The conceptual synthesis produces several principal findings. First, defect detection and test prioritization should be treated as interconnected but distinct functions. Detection estimates where defects may exist, whereas prioritization determines where limited testing effort should be concentrated. Combining both functions provides a stronger quality-engineering strategy than relying on either independently.

Second, the literature indicates that automated observation and computational representation are foundational requirements for intelligent decision systems. Monitoring-oriented work demonstrates the value of continuous data acquisition (Liu, 2021), while visualization and simulation studies demonstrate how complex system conditions can be represented computationally (Huang et al., 2017; Kim et al., 2021). In software testing, these principles support the creation of data pipelines capable of continuously updating defect and risk estimates.

Third, risk-aware prioritization is theoretically more useful than defect probability alone. A model that predicts failure likelihood without considering impact can misallocate testing resources. Incorporating risk therefore creates a decision layer between prediction and execution. This principle is consistent with predictive risk modelling as a mechanism for improving decision-making under uncertainty (Philip, 2024).

Fourth, simulation provides an important intermediate validation mechanism. The reviewed studies show that simulated environments can support experimentation and learning (Bogusevschi et al., 2020; Long et al.,

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 06 ISSUE 08 Pages: 150-160
OCLC – 1368736135

2021). For software quality engineering, simulation can be used to evaluate whether AI models identify representative failure conditions before exposing them to production systems.

Finally, the framework indicates that human interpretability remains essential. AI-generated priorities must be sufficiently transparent for quality engineers to challenge incorrect predictions. Therefore, the proposed architecture is best characterized as human-supervised intelligent testing rather than fully autonomous testing.

Because the provided references do not contain a controlled software-testing experiment, these findings represent analytical and conceptual outcomes rather than measured improvements in defect-detection accuracy, test execution time, or fault-detection effectiveness. Empirical validation remains necessary.

Discussion

The proposed framework contributes theoretically by integrating three traditionally separable quality-engineering activities: anomaly or defect detection, risk prediction, and test prioritization. The literature demonstrates that intelligent systems become more useful when computational observations are transformed into actionable information. Simulation, visualization, and monitoring provide the technological foundation, while predictive modelling provides the decision mechanism.

The principal practical implication is resource allocation. Testing teams rarely possess unlimited execution time, computational capacity, or engineering attention. AI-assisted prioritization can therefore focus early testing on scenarios with stronger evidence of potential failure or higher business impact. This does not eliminate exhaustive testing; rather, it changes the sequence in which tests are executed.

A second implication concerns continuous quality engineering. Monitoring-oriented systems illustrate the value of current data (Liu, 2021), while simulation-based approaches demonstrate the ability to evaluate complex conditions in controlled environments (Kim et al., 2021). Combining these concepts enables a feedback-oriented testing architecture in which newly observed failures influence subsequent priorities.

Nevertheless, several trade-offs must be recognized. A model trained on historical failures may reproduce historical biases and fail to detect novel defect classes. Similarly, simulation can improve controlled experimentation but cannot guarantee complete representation of production behaviour. This limitation parallels the broader problem of transferring computational representations to real-world conditions.

False positives present another challenge. Excessive high-risk predictions can cause test suites to become unnecessarily concentrated around certain components. Conversely, false negatives may cause critical tests to be postponed. The prioritization algorithm must therefore be evaluated not only by classification

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 06 ISSUE 08 Pages: 150-160
OCLC – 1368736135

ISSN-2750-1396

accuracy but also by its effect on practical testing outcomes.

Interpretability is another major limitation. A sophisticated model may generate strong predictions while providing weak explanations. Such opacity can reduce engineers' confidence and make debugging difficult. Visualization-oriented research suggests that complex computational information becomes more useful when it is presented in interpretable forms (Huang et al., 2017).

The framework also has a data-dependency problem. AI-assisted quality engineering requires sufficiently rich and reliable historical data. New projects, rapidly changing applications, or systems with limited defect histories may not provide enough information for dependable prediction. In such circumstances, deterministic testing rules and expert judgment remain important.

Finally, the present study is limited by the nature of its evidence base. The supplied references predominantly address AR, simulation, engineering education, monitoring, and computational visualization rather than direct AI-based software testing. Consequently, the proposed relationships are theoretically transferable but cannot be interpreted as empirical evidence that a particular AI model will outperform conventional testing techniques. Future empirical research should evaluate the framework using real software repositories, historical defect datasets, continuous integration environments, and measurable testing outcomes.

Conclusion

AI-assisted defect detection and test prioritization provide a promising conceptual direction for modern software quality engineering because they address two complementary questions: where defects are likely to occur and which tests should be executed first. The synthesis presented in this research indicates that effective intelligent testing can be structured around automated data acquisition, feature representation, AI-based defect estimation, risk modelling, test prioritization, simulation, feedback, and human oversight.

The reviewed literature supports the technological foundations of this framework through evidence concerning computational simulation, real-time monitoring, interactive visualization, and technology-assisted experimentation. These principles can be transferred conceptually to software quality engineering, although direct empirical validation is still required. Risk-oriented prediction further strengthens the framework by allowing defect probability to be combined with impact and exposure when determining test priority (Philip, 2024).

The principal contribution of the study is therefore a unified conceptual model rather than a claim of measured algorithmic superiority. Future research should implement the framework using real-world software-testing datasets and compare machine-learning models according to defect-detection precision, recall,

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 06 ISSUE 08 Pages: 150-160
OCLC – 1368736135

ISSN-2750-1396

fault-detection effectiveness, execution cost, prioritization efficiency, and interpretability. Additional research should investigate adaptive models capable of responding to changing software architectures and previously unseen defect patterns. Such empirical development would provide the evidence necessary to transform the proposed conceptual framework into a validated AI-assisted quality-engineering methodology.

REFERENCES

1. Bogusevschi, D., Muntean, C. H., & Muntean, G. M. (2020). Teaching and learning physics using 3D virtual learning environment: A case study of combined virtual reality and virtual laboratory in secondary school. Journal of Computers in Mathematics and Science Teaching,39,518.http://www.newtonproject.eu/wp-content/uploads/2019/07/SITE2019_DBogusevschi_ Camera Ready.pdf
2. Cai, S., Liu, C., Wang, T., Liu, E., & Liang, J. (2021). Effects of learning physics using Augmented Reality on students' self-efficacy and conceptions of learning. British Journal of Educational Technology,52, 235–251.
3. COMSOL Multiphysics 5.6. 2020. Available at:https://www.comsol.com/model/thermoelectric-cooler-30611
4. Cui, Y., Zhao, H., & Zhao, L. (2020). Design of greenhouse shuttercontroller based on STM8.Agricultural Technology and Equipment, 01, 21–23.
5. Garon, M., Boulet, P., Doironz, J., Beaulieu, L., & Lalonde, J.(2016). Real-time high resolution 3D data on the HoloLens.In2016 IEEE International Symposium on Mixed and Augmented Reality (ISMAR-Adjunct)(pp. 189–191).
6. Harun, Tuli, N., & Mantri, A. (2020). Experience Fleming's rule inelectromagnetism using augmented reality: Analyzing impacton students learning.
7. Procedia Computer Science,172,660–668.
8. Hedenqvist, C., Romero, M., & Vinuesa, R. (2021). Improving the learning of mechanics through augmented reality. Technology Knowledge Learning,
9. Huang, J. M., Ong, S.K., & Nee, A. Y. C. (2017). Visualization and interaction of finite element analysis in augmented reality. Computer-Aided Design,84,1–14.
10. Kim, Y., Kim, H., Lee, S., & Kim, W. (2021). Trustworthy buildingfire detection framework with simulation-based learning. IEEE Access.
11. Liu, W. (2021). Design of wireless low-power temperature and humidity monitoring terminal based on STM8.Electrotechnics,18,42–44.
12. Long,X.,Chen,Y.,&Zhou,J.(2021).Augmentedrealityexperimentonmechanical stress of block on arch with simulation-based asset. In2021 4th International Conference on Information Systems and Computer Aided Education, September 24–26, 2021, Dalian, China. ACM, New York, NY, USA, 5 pages.
13. Lu, Y., Chen, J., Li, J., & Xu, W. (2022). A study on theele ctromagnetic–thermal coupling

International Journal of Advance Scientific Research
(ISSN – 2750-1396)
VOLUME 06 ISSUE 08 Pages: 150-160
OCLC – 1368736135

effect of cross-slotfre quency selective surface. Materials, 15, 640.

14. Mota, J. M., Rube, I. R., Dodero, J. M., & Sánchez, I. A. (2018).Augmented reality mobile app development for all. Computers and Electrical Engineering,65, 250–260.
15. Osadchyi, V. V., Valko, N. V., & Kuzmich, L. V. (2021). Usingau gmented reality technologies for STEM education organization. Journal of Physics Conference Series, 1840,012027.
16. Sahin, D., & Yilmaz, R. M. (2020). The effect of augmented reality technology on middle school students' achievements and attitudes towards science education. Computers &Education,144, 103710, 1–11.
17. 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
18. 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.