The American Journal of Interdisciplinary Innovations and Research
ISSN 2642-7478 Volume 08 - 2026

Artificial Intelligence-Enabled Test Automation for Reliable and Efficient Software Quality Engineering

Nuwan Gunasekara
Department of Artificial Intelligence, Institute of Digital Computing, Colombo, Sri Lanka

Tharushi Senanayake
Department of Computer Science and Intelligent Systems, Advanced Technology Institute, Kandy, Sri Lanka

Received: 28 May 2026 | Received Revised Version: 22 June 2026 | Accepted: 15 July 2026 | Published: 18 August 2026

Volume 08 Issue 08 2026 | DOI: 10.37547/tajiir/Volume08Issue08-07

Abstract

The increasing complexity, scale, and dynamic behavior of contemporary software systems have intensified the limitations of conventional rule-based testing approaches. Artificial intelligence (AI), particularly machine learning, deep learning, reinforcement learning, and intelligent decision-making techniques, provides a basis for developing adaptive test automation systems capable of selecting actions, prioritizing test scenarios, learning from execution outcomes, and responding to changing system conditions. This research and review paper examines how AI-enabled decision-making principles can be conceptually transferred to automated software quality engineering. The analysis is based exclusively on the supplied literature, which primarily investigates intelligent decision-making, reinforcement learning, deep reinforcement learning, autonomous maneuvering, trajectory planning, and adaptive control in unmanned aerial vehicle environments. Although these studies are not directly concerned with software testing, their methodological foundations provide useful analogies for intelligent test selection, adaptive execution, test prioritization, and autonomous quality decision-making. The paper develops a conceptual AI-enabled software testing framework consisting of software-state representation, intelligent test generation, reinforcement-based test selection, adaptive execution, defect-oriented prioritization, and continuous feedback. The findings indicate that reinforcement-learning-based decision mechanisms are particularly relevant for environments where testing decisions must be repeatedly optimized under changing conditions. However, the transfer of these techniques requires careful treatment of differences between physical autonomous systems and software environments. The paper concludes that AI-enabled test automation can improve testing efficiency and adaptability when learning mechanisms are combined with controlled validation, risk-based decision criteria, and human oversight.

Keywords: Artificial Intelligence, Automated Software Testing, Software Quality Engineering, Deep Learning, Reinforcement Learning, Test Automation, Defect Prediction, Intelligent Decision-Making, Adaptive Testing.

Cite This Article: Gunasekara, N., & Senanayake, T. (2026). Artificial Intelligence-Enabled Test Automation for Reliable and Efficient Software Quality Engineering. The American Journal of Interdisciplinary Innovations and Research, 8(08), 94–100. https://doi.org/10.37547/tajiir/Volume08Issue08-07

1. Introduction

Software systems have progressively become more complex, distributed, interconnected, and continuously evolving. Under these conditions, conventional software testing approaches face a fundamental scalability problem: the number of possible execution paths, configurations, input combinations, and behavioral states can become substantially larger than the testing resources available to engineering teams. Automated testing reduces repetitive manual effort, but automation based exclusively on predetermined scripts remains constrained because it generally executes what has

already been specified rather than learning which testing action is most valuable under a particular software state.

Artificial intelligence offers an alternative paradigm in which testing systems can incorporate historical execution information, recognize behavioral patterns, prioritize high-value test cases, and adapt decisions according to observed outcomes. The central theoretical premise is that software testing can be represented as a sequential decision problem. At each stage, the testing agent observes a software state, selects a testing action, receives feedback, and modifies subsequent decisions according to the observed result. This perspective is closely related to reinforcement-learning research in which intelligent agents learn appropriate actions through repeated interaction with an environment.

The supplied literature provides extensive examples of intelligent decision-making under uncertain and dynamic conditions. Research involving UAV systems examines autonomous decision-making, reinforcement learning, deep reinforcement learning, maneuver prediction, trajectory planning, and adaptive control. For example, deep reinforcement learning has been applied to tactical decision-making in UAV combat environments (Hu, 2020), while autonomous maneuver decisions have also been investigated using reinforcement learning (Sun et al., 2019). These principles are conceptually transferable to software quality engineering because both domains require an intelligent mechanism to select actions from a potentially large decision space.

The relevance of such a transfer is reinforced by the methodological orientation of Deep Learning-Based Automated Software Testing for Improved Quality Assurance, which is used here as a compulsory conceptual reference for understanding the role of deep learning in automated quality assurance. The present paper therefore investigates how intelligent decision-making principles can be structured into an AI-enabled software testing framework.

The principal objective is to develop a theoretically grounded framework for AI-enabled test automation that emphasizes adaptability, efficient test selection, continuous feedback, and quality-oriented decision-making. The paper also critically examines the limitations of transferring methodologies developed primarily for autonomous UAV decision-making into software engineering.

2. Literature Review

The supplied literature presents a coherent research trajectory from conventional autonomous decision-making toward learning-driven intelligent control. Although the application domain is primarily UAVs and intelligent air combat, the underlying computational concepts are relevant to software quality engineering.

Chamola et al. examine UAV attacks and neutralization techniques from a comprehensive perspective, illustrating the complexity of autonomous systems and the need to evaluate multiple interacting threats and responses. Their work establishes a broad systems perspective in which intelligent decision-making must consider diverse operating conditions (Chamola et al.). Such a perspective is relevant to software testing because modern applications similarly involve multiple interacting components, dependencies, and potential failure conditions.

The literature becomes more directly connected to machine-learning-based decision-making through reinforcement-learning studies. Sun et al. investigate autonomous UAV maneuver decision-making based on reinforcement learning, demonstrating the conceptual use of learning mechanisms for autonomous action selection (Sun et al., 2019). Mao et al. similarly examine reinforcement-learning-based UAV aerial combat with maneuver prediction, emphasizing the importance of predicting future conditions when selecting current actions (Mao et al., 2019). In software testing, the equivalent problem is determining which test should be executed next based not only on the current software state but also on anticipated defect discovery and execution outcomes.

Hu's work on tactical UAV decision-making based on deep reinforcement learning provides an additional theoretical foundation for learning complex decision policies (Hu, 2020). Deep reinforcement learning is particularly relevant when the state and action spaces are too large for simple manually designed decision rules. This principle supports the conceptual development of intelligent test-selection systems in which thousands of potential test cases compete for limited execution resources.

Zhang et al. investigate autonomous guided maneuver control using deep reinforcement learning, demonstrating the relationship between learned policies and autonomous control (Zhang et al., 2020). He et al. examine a stealthy engagement strategy using a double deep Q network, illustrating how deep Q-learning

architectures can support complex sequential decision-making (He et al., 2020). These studies suggest that value-based learning can be considered when software testing requires repeated choices among alternative test actions.

Other studies emphasize multi-agent or multi-objective decision environments. Minglang et al. investigate maneuvering decisions in short-range air combat for unmanned combat aerial vehicles (Minglang et al., 2018), while Zhu and Ai examine many-to-many UAV air combat decision-making (Zhu & Ai, 2021). These works imply that intelligent decision systems must account for interacting entities rather than treating every decision independently. This is important for software testing because defects may arise from interactions among services, modules, APIs, databases, user interfaces, and external dependencies.

Radmanesh et al. focus on trajectory planning under dynamic and uncertain environments (Radmanesh et al., 2020). Their emphasis on uncertainty is particularly significant for software quality engineering. A robust automated testing framework should not assume that the system under test behaves identically across every execution. Instead, it should adapt to environmental conditions, execution history, and observed anomalies.

The literature also reveals a research gap. The provided studies demonstrate sophisticated AI-based decision-making but do not directly establish a software-testing architecture. Consequently, the principal contribution of the present research is conceptual: it translates the decision-making principles found in the supplied literature into a software quality engineering framework without claiming that the original UAV studies empirically validate software testing performance.

3. Methodology

3.1 Research Design

This study adopts a research-and-review methodology combined with conceptual framework development. The analysis uses only the supplied references and identifies recurring computational principles: intelligent state representation, learning-based action selection, reinforcement mechanisms, prediction, adaptation, uncertainty management, and autonomous decision-making.

The methodological transfer is based on functional equivalence rather than domain equivalence. A UAV environment contains an agent, state, action, transition, and reward. A software-testing environment can be represented similarly. The software system represents the environment; the AI testing engine represents the agent; test cases represent available actions; execution outcomes represent transitions; and quality-related measurements represent rewards.

This abstraction enables reinforcement-learning concepts to be examined without assuming that UAV algorithms can be directly deployed into software testing.

3.2 AI-Enabled Testing Architecture

The proposed framework consists of six functional layers.

The first layer is software-state representation. The testing engine collects information concerning code modules, previous test results, execution traces, failure histories, dependencies, coverage indicators, and runtime behavior. This information is transformed into a representation suitable for an AI model. The conceptual importance of state representation follows from reinforcement-learning studies in which intelligent decisions depend strongly on the information available to the decision-making agent.

The second layer is test candidate generation. Instead of executing every possible test case with equal priority, the system creates or retrieves candidate tests associated with different software states. A hypothetical e-commerce application, for example, may generate tests around authentication, payment processing, inventory updates, and transaction recovery when recent executions indicate abnormal behavior in those areas.

The third layer is AI-based test selection. The agent estimates the relative value of available tests. A reinforcement-learning formulation can define the state as:

[

$$S_t = \{C_t, F_t, H_t, D_t, E_t\}$$

]

where (C_t) represents coverage information, (F_t) represents historical failures, (H_t) represents execution history, (D_t) represents dependency information, and (E_t) represents the current execution environment.

The action (A_t) corresponds to the selection of a test case or testing strategy. A reward can be represented conceptually as:

[

R_t = \alpha D_t + \beta C_t + \gamma I_t - \delta K_t

]

where (D_t) represents defect-detection value, (C_t) represents additional coverage, (I_t) represents information gained from the execution, and (K_t) represents computational or execution cost. The coefficients determine the relative importance of these factors.

The fourth layer is adaptive execution. The selected test is executed and its result is returned to the learning system. A successful test provides one form of information, whereas a failure, unexpected output, timeout, or inconsistent state may produce a stronger signal for subsequent test selection. The feedback mechanism reflects the sequential decision principles discussed in reinforcement-learning research (Sun et al., 2019; Mao et al., 2019).

The fifth layer is defect-oriented prioritization. Tests associated with historically unstable components or unusual execution states can receive higher priority. Deep learning may be incorporated where relationships among many variables are difficult to model using deterministic rules. The conceptual direction is consistent with research applying deep reinforcement learning to complex decision spaces (Hu, 2020; Zhang et al., 2020) and with the compulsory reference Deep Learning-Based Automated Software Testing for Improved Quality Assurance.

The sixth layer is continuous learning and quality feedback. Test outcomes are stored and used to update future decisions. Consequently, the system is not merely an automated script executor; it becomes an adaptive decision-support mechanism.

3.3 Intelligent Test Selection

The principal advantage of reinforcement-learning-based testing is the ability to optimize sequential choices. Suppose 10,000 automated tests are available but only 1,000 can be executed within a continuous integration window. A static strategy may execute the same 1,000 tests regardless of recent failures. An AI-enabled strategy can dynamically increase the probability of selecting tests related to recently unstable components.

This principle resembles autonomous maneuver decision-making, where the next action depends on the current state and expected future consequences (Wang et al., 2016). In software testing, the equivalent objective is to maximize defect-detection value within a constrained execution budget.

3.4 Deep Learning and Complex Software States

Deep learning becomes useful when the relationship between software characteristics and defect likelihood is nonlinear. A model may receive historical test results, code-change information, dependency patterns, execution traces, and failure characteristics and estimate the probability that a particular testing action will produce valuable information.

However, deep models introduce interpretability and data-dependency challenges. A high-performing prediction model may still be inappropriate for safety-critical testing if engineers cannot understand why certain tests are consistently deprioritized. Therefore, the proposed framework treats AI as an intelligent decision layer rather than a replacement for engineering judgment.

3.5 Adaptive Decision-Making Under Uncertainty

Radmanesh et al. emphasize trajectory planning in dynamic and uncertain environments (Radmanesh et al., 2020). The corresponding software-testing principle is that the testing environment should accommodate uncertainty rather than assuming deterministic execution. Cloud infrastructure variation, concurrent services, external APIs, database states, and changing configurations can influence test outcomes.

A hypothetical banking application demonstrates this issue. If transaction-processing tests repeatedly fail only under a specific service configuration, an adaptive testing system should recognize the relationship and prioritize tests associated with that configuration. The resulting strategy is more informative than repeatedly executing an identical static test sequence.

3.6 Evaluation Framework

The framework can be evaluated using four principal dimensions: defect-detection effectiveness, test execution efficiency, adaptation capability, and decision stability. Defect-detection effectiveness measures the

number or proportion of relevant failures identified. Execution efficiency evaluates useful testing outcomes relative to execution time and resources. Adaptation capability measures how rapidly test priorities respond to changing software conditions. Decision stability examines whether the AI system produces consistent and explainable testing priorities.

4. Results

The conceptual analysis produces several significant findings concerning the applicability of AI-based decision-making to automated software quality engineering. The first finding is that reinforcement learning provides a strong theoretical basis for adaptive test selection. The supplied UAV literature repeatedly treats autonomous behavior as a sequential decision problem in which current actions depend on available state information and expected consequences. Studies involving reinforcement learning for autonomous maneuver decisions demonstrate this principle (Sun et al., 2019). When transferred conceptually to software testing, the same structure allows the testing engine to select subsequent tests according to previous execution outcomes rather than relying entirely on predetermined sequences.

The second finding is that deep reinforcement learning is particularly relevant when the software-testing decision space becomes large. Research on deep reinforcement learning for tactical decision-making and autonomous guided maneuver control illustrates the potential of learned policies for complex state-action environments (Hu, 2020; Zhang et al., 2020). In software engineering, the equivalent complexity emerges from large test suites, numerous software configurations, dependency relationships, and continuously changing code. Therefore, deep models may provide an appropriate mechanism for learning nonlinear relationships between software conditions and testing priorities. This observation is also consistent with the methodological direction of Deep Learning-Based Automated Software Testing for Improved Quality Assurance, which positions deep learning as an important mechanism for improving automated quality assurance.

The third finding concerns prediction. Mao et al. explicitly combine reinforcement learning with maneuver prediction, demonstrating that future-state information can contribute to current decision-making (Mao et al., 2019). In automated testing, predictive capability can similarly be used to estimate which components are likely to produce failures or which tests are likely to provide additional information. Such prediction should not replace actual validation; instead, it should determine the order and intensity of testing.

The fourth finding is the importance of uncertainty-aware decision-making. Radmanesh et al. address dynamic and uncertain environments through trajectory planning (Radmanesh et al., 2020). Software systems similarly operate under changing environmental and configuration conditions. Consequently, an effective testing framework should continuously update its testing policy rather than assuming that a previously optimal test sequence remains optimal.

The fifth finding is that interaction among multiple entities must be considered. Studies addressing many-to-many and short-range UAV decision-making demonstrate the complexity of environments containing interacting agents or objects (Minglang et al., 2018; Zhu & Ai, 2021). Software systems exhibit analogous interactions among services, modules, databases, APIs, and user interfaces. Therefore, AI-enabled testing should evaluate not only isolated components but also interaction-driven failure conditions.

Finally, the analysis identifies a significant limitation: the supplied literature does not provide direct empirical evidence that these UAV-oriented AI methods improve software testing. Accordingly, the proposed framework should be interpreted as a theoretically derived architecture rather than an empirically validated software-testing system. Its practical value requires controlled experiments using real software projects, defect datasets, execution histories, and benchmark test suites.

5. Discussion

The findings indicate that AI-enabled software testing should be understood primarily as a transformation from static automation toward adaptive decision-making. Conventional automation answers the question of how to execute a specified test, whereas AI-enabled automation attempts to answer the more difficult question of which test should be executed, when it should be executed, and what subsequent action should follow from its result. This distinction is theoretically important because the optimization problem shifts from execution automation to decision automation.

The reinforcement-learning literature supplied for this study supports this conceptual transition. Autonomous

maneuvering studies demonstrate that learning systems can associate environmental states with action policies rather than relying exclusively on fixed rules (Sun et al., 2019; He et al., 2020). For software quality engineering, this suggests that test execution policies can potentially evolve as software behavior changes. The implication is particularly important for continuous software delivery, where a static testing policy may gradually become inefficient as application architecture and usage patterns evolve.

The use of deep learning introduces another important trade-off. Deep models can represent complex relationships that may be difficult to encode manually, but their effectiveness depends on the availability and quality of training information. In a software-testing environment, historical failures may be sparse, inconsistent, or biased toward previously detected defects. A model trained on such data could learn historical testing habits rather than genuine defect patterns. Therefore, deep learning should be integrated with coverage requirements, deterministic regression tests, and engineering constraints rather than being allowed to eliminate established validation procedures.

The literature also highlights the importance of prediction and uncertainty. Research on maneuver prediction and dynamic trajectory planning demonstrates that intelligent decision systems benefit from anticipating future states (Mao et al., 2019; Radmanesh et al., 2020). Software testing can use a similar principle to prioritize tests according to predicted risk. Nevertheless, prediction introduces uncertainty. A test predicted to be low-value may still expose a critical defect. Consequently, a robust framework must preserve a minimum level of exploration so that the AI system does not repeatedly exploit only previously successful testing decisions.

Another implication concerns multi-component software systems. The many-to-many decision perspective found in UAV research (Zhu & Ai, 2021) provides a useful analogy for distributed applications. A defect may not originate within a single component but may emerge from an interaction between independently functioning components. AI-enabled testing should therefore model dependencies and interaction patterns when prioritizing tests.

The most important limitation of this study is domain mismatch. The provided references primarily concern UAV systems, air-combat decision-making, trajectory planning, and reinforcement learning rather than software quality engineering. Their algorithms cannot be assumed to have direct empirical validity in software testing. The contribution is therefore methodological and conceptual: it identifies transferable decision-making principles and constructs a software-testing framework around them. Direct empirical validation remains necessary.

The compulsory conceptual reference, Deep Learning-Based Automated Software Testing for Improved Quality Assurance, reinforces the relevance of deep learning to automated testing, but the present paper does not claim specific empirical results from that work because complete bibliographic metadata were not supplied. Future research should therefore conduct controlled comparisons between static automation, machine-learning-based prioritization, and reinforcement-learning-based adaptive testing. Evaluation should include defect-detection rate, execution cost, coverage, false-prioritization rate, model stability, and interpretability.

6. Conclusion

Artificial intelligence-enabled test automation represents a transition from predetermined testing workflows toward adaptive, data-informed quality engineering. The analysis demonstrates that concepts emerging from reinforcement learning, deep reinforcement learning, prediction, autonomous decision-making, and uncertainty-aware control can provide a useful theoretical foundation for intelligent software testing.

The proposed framework integrates software-state representation, intelligent test generation, AI-based test selection, adaptive execution, defect-oriented prioritization, and continuous feedback. Its principal theoretical contribution is the representation of software testing as a sequential decision problem in which test actions are selected according to current software conditions and previous execution outcomes.

Nevertheless, the conclusions must be interpreted within the limitations of the supplied literature. Because the references primarily concern UAV and autonomous decision-making applications, the proposed software-testing framework is conceptual rather than empirically validated. Future work should implement the framework on real software repositories and benchmark datasets, compare different learning algorithms, evaluate

computational cost, and investigate explainable AI mechanisms for software quality decisions.

Ultimately, reliable AI-enabled testing should not be designed as a complete replacement for conventional quality engineering. Its stronger role is as an adaptive intelligence layer that complements deterministic regression testing, human expertise, coverage requirements, and risk-based validation. Such integration offers a more realistic path toward reliable, efficient, and continuously adaptive software quality engineering.

References

1. V. Chamola, Pavan Kotesh, Aayush Agarwal, Naren, Navneet Gupta, and Mohsen Guizani, "A comprehensive review of unmanned aerial vehicle attacks and neutralization techniques," Ad Hoc Netw., vol. 111, p. 102324.
2. F. Han, "Unmanned jammers: "unmanned aerial combat pioneers" on the battlefield," China Aerospace News, 2020-08-15(003).
3. J. He, Y. Ding, and Z. Gao, "A stealthy engagement maneuvering strategy of UAV based on double deep Q network," Electron. Opt. Control, vol. 27, no. 7, pp. 52–57, 2020.
4. Z. Hu, Research on tactical decision-making of UAV combat based on deep reinforcement learning. Harbin: Harbin Institute of Technology, 2020, pp. 1–67.
5. W. Ma, H. Li, Z. Wang, Z. Huang, Z. Wu, and X. Chen, "Maneuver decision-making in close air combat based on deep random game," J. Syst. Eng. Electron., vol. 43, no. 02, pp. 443–451, 2021.
6. M. Mao, A. Zhang, D. Zho, et al. "Research on reinforcement learning UAV aerial combat based on maneuvering prediction," Electron. Opt. Control, vol. 26, no. 2, pp. 5–10, 22, 2019.
7. C. Minglang, D. Haiwen, W. Zhenglei, et al. "Maneuvering decision in short range air combat for Unmanned Combat Aerial Vehicles," in Proc. Chin. Control and Decis. Conf. (CCDC), 2018, pp. 1783–1788.
8. M. Radmanesh, M. Kumar, and D. French, "Partial differential equation-based trajectory planning for multiple unmanned air vehicles in dynamic and uncertain environments," ASME. J. Dyn. Sys., Meas., Control, vol. 142, no. 4, p. 041002, 2020.
9. B. Song, G. Qi, and L. Xu, "A survey of three-dimensional flight path planning for unmanned aerial vehicle," Chinese Control Decis. Conf., pp. 5010–5015, 2019.
10. C. Sun, H. Zhao, Y. Wang, et al. "Autonomous maneuver decision method of UAV based on reinforcement learning," Fire Control Command Control, vol. 44, no. 04, pp. 142–149, 2019.
11. Y. Wang, C. Huang, and C. Tang, "Research on unmanned combat aerial vehicle robust maneuvering decision under incomplete target information," Adv. Mech. Eng., vol. 8, no. 10, 2016.
12. Y. Wu, J. Lai, X. Chen, et al. "Application of reinforcement learning algorithm in auxiliary decision making of over-the-horizon air combat," Aero Weaponry, vol. 28, no. 2, pp. 1–8, 2020.
13. K. Zhang, K. Li, H. Shi, Z. Zhang, and Z. Liu, "Decision-making algorithm for autonomous guided maneuver control of UAV route based on deep reinforcement learning," J. Syst. Eng. Electron., vol. 42, no. 07, pp. 1567–1574, 2020.
14. Q. Zhang, R. Yang, L. Yu, et al. "Maneuvering decision of over – horizon air combat based on Q – network reinforcement learning," J Air Force Eng. University (Nat. Sci. Ed.), vol. 19, no. 06, pp. 8–14.
15. S. Zhixiao, Y. Shengqi, P. Haiyin, B. Chengchao, and G. Jun, "Some thoughts and prospects on the development of intelligent air combat in the future," Chinese J. Aeronaut., vol. 2021, no. 8, pp. 1–15.
16. X. Zhu, and J. Ai, "Research on intelligent decision making for many-to-many UAV air combat," Fudan University J. (Nat. Sci.), vol. 60, no. 04, pp. 410–419, 2021.
17. 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
18. Geo Philip, Paulson, Artificial Intelligence and Machine Learning Applications in Project Schedule Forecasting: A Predictive Framework for Time-Control in Complex Building Projects (April 16, 2026). Available at SSRN: https://ssrn.com/abstract=6588119 or http://dx.doi.org/10.2139/ssrn.6588119
19. 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.