



THE AI–ANALYST–TESTER TRIANGLE: A NEW FRAMEWORK FOR INTELLIGENT SOFTWARE QUALITY MANAGEMENT

Darshan Dandavathi¹, Prof. Swati Kanmadi²

¹ Senior Business Analyst

II Create Software, Dharwad, Karnataka, India

² Assistant Professor, Department of BBA

Chetan College of Commerce, BBA & BCA, Hubli, Karnataka, India

ABSTRACT :

The rapid integration of artificial intelligence (AI) into software engineering is transforming how software requirements are interpreted, test cases are generated, defects are identified, and quality decisions are made. However, the use of AI-enabled tools does not automatically ensure software reliability, security, usability, or compliance. AI-generated outputs may contain inaccuracies, overlook contextual requirements, reproduce biases, or create misleading test results. These challenges demonstrate the need for a structured framework that combines computational intelligence with human analytical judgment and systematic software testing.

This paper proposes the **AI–Analyst–Tester Triangle (AAT Triangle)** as a conceptual framework for intelligent software quality management. The framework integrates three complementary components: AI as an intelligent assistant for generating insights and automating repetitive quality activities; the analyst as the interpreter of business requirements, operational context, risks, and quality objectives; and the tester as the independent evaluator responsible for validating software behaviour, detecting defects, and verifying acceptance criteria. The framework emphasizes continuous feedback, evidence-based decision-making, risk prioritization, traceability, and human accountability.

The study adopts a conceptual research design based on an analytical review of selected literature, software quality standards, AI testing guidance, and responsible AI frameworks. It develops a proposed process model, identifies potential performance indicators, formulates research hypotheses, and outlines an empirical validation strategy. The paper argues that intelligent software quality management should be understood not as the replacement of human expertise by AI, but as a coordinated process in which machine-generated intelligence, analytical interpretation, and independent verification reinforce one another. The proposed framework offers a foundation for future empirical research and practical implementation in software development organizations.

Keywords: Artificial intelligence, software quality management, software testing, business analysis, human–AI collaboration, defect detection, intelligent test automation, quality assurance, risk-based testing.

1. Introduction

Software quality has become a strategic concern for organizations that depend on digital platforms, enterprise applications, cloud services, mobile applications, and data-driven business systems. Software failures can cause financial losses, operational interruptions, security vulnerabilities, customer dissatisfaction, and reputational damage. Consequently, software quality management requires more than the identification of coding errors; it involves the systematic evaluation of whether software satisfies functional requirements, non-functional expectations, stakeholder needs, and applicable quality criteria.

Traditional software quality practices rely on collaboration among business analysts, developers, quality assurance professionals, testers, project managers, and users. Business analysts interpret stakeholder requirements, testers examine software behaviour, and quality managers coordinate processes for preventing and controlling defects. Although these responsibilities are complementary, they may become fragmented when teams work with incomplete documentation, rapidly changing requirements, compressed release schedules, or disconnected tools.

Artificial intelligence introduces opportunities to improve these activities. AI-enabled systems can assist with requirement classification, test-case generation, defect prediction, log analysis, test prioritization, regression testing, and the summarization of test results. Machine-learning techniques can identify patterns in historical defects, while generative AI can propose test scenarios and explain potential failure conditions.

A systematic review by Ajorloo et al. (2024) examined machine-learning methods used in software testing, including supervised, unsupervised, reinforcement, and hybrid learning approaches. The review illustrates the breadth of research on AI-supported testing and the importance of evaluating both the benefits and limitations of individual techniques. However, the availability of intelligent tools does not eliminate the need for contextual interpretation, independent verification, or accountable quality decisions.

AI-generated test cases may be technically plausible but fail to represent actual business rules. An analyst may correctly interpret a requirement but overlook an unusual interaction between system components. A tester may execute a comprehensive test suite but fail to question whether the test data represent realistic operational conditions. These examples indicate that quality failures can arise not only from weaknesses in individual activities but also from gaps between them.

This paper addresses that coordination problem by proposing the AI–Analyst–Tester Triangle. The model treats AI, the analyst, and the tester as three interconnected contributors to a continuous quality management process.

The central argument is that software quality can be strengthened when AI-generated insights are interpreted by an analyst, converted into verifiable quality objectives, independently evaluated by a tester, and subsequently used to improve both AI-supported analysis and future testing.

1.1 Background of the Study

The evolution of software engineering has progressed from predominantly manual verification towards automated testing, continuous integration, continuous delivery, and increasingly intelligent quality engineering. Automation has improved execution speed and repeatability, while AI offers additional capabilities for processing large amounts of requirements, source code, execution logs, and historical defect information.

Nevertheless, software quality is multidimensional. ISO/IEC 25010:2023 provides a product quality model containing nine quality characteristics and associated subcharacteristics. It can support the specification, evaluation, and measurement of software and ICT product quality throughout the product lifecycle (International Organization for Standardization [ISO], 2023).

AI-supported quality activities must therefore be aligned with explicit quality objectives rather than judged solely by the number of generated test cases or the speed of automated execution.

Similarly, the National Institute of Standards and Technology's Artificial Intelligence Risk Management Framework emphasizes the identification, assessment, and management of AI-related risks throughout the AI lifecycle (Tabassi, 2023). This reinforces the need to examine not only whether AI can generate useful outputs but also whether those outputs are reliable, appropriate, explainable, and subject to effective oversight.

1.2 Problem Statement

Organizations adopting AI for software testing may encounter five interconnected problems:

1. **Contextual limitations:** AI-generated recommendations may not fully reflect stakeholder intent, organizational policies, or domain-specific constraints.
2. **Analytical fragmentation:** Requirements, risk assessments, test scenarios, and defect reports may be managed through separate workflows.
3. **Verification gaps:** AI-generated test cases or defect classifications may be accepted without sufficient independent validation.
4. **Accountability ambiguity:** Teams may be uncertain about who is responsible for validating AI outputs and approving release decisions.
5. **Insufficient feedback integration:** Lessons from test execution may not consistently improve requirements analysis, AI prompts, test-generation rules, or future quality planning.

These issues suggest the need for a coordinated framework that integrates AI capabilities, analytical judgment, and systematic testing within a common quality management structure.

1.3 Research Aim

The primary aim of this study is to develop a conceptual framework for intelligent software quality management through the coordinated interaction of artificial intelligence, the analyst, and the tester.

1.4 Research Objectives

The study pursues the following objectives:

1. To examine the potential role of AI in software quality management.
2. To identify the complementary responsibilities of AI systems, analysts, and software testers.
3. To propose the AI–Analyst–Tester Triangle as an integrated quality management framework.
4. To establish a process for requirements interpretation, risk assessment, test generation, validation, and continuous improvement.
5. To identify performance indicators for evaluating the proposed framework.
6. To formulate testable hypotheses for future empirical investigation.
7. To identify implementation challenges, governance requirements, and directions for further research.

1.5 Research Questions

- **RQ1:** How can AI, analysts, and testers be integrated into a unified software quality management process?
- **RQ2:** What responsibilities should each component perform to reduce quality gaps and improve traceability?
- **RQ3:** How can feedback between AI-supported analysis and independent testing contribute to continuous quality improvement?
- **RQ4:** Which measurable indicators can be used to evaluate the effectiveness of the proposed framework?
- **RQ5:** What organizational, technical, and governance factors may influence successful implementation?

2. Review of Literature

2.1 Artificial Intelligence in Software Testing

AI applications in software testing include defect prediction, test-case generation, test-suite optimization, automated user-interface testing, log analysis, and the identification of anomalous software behaviour.

Machine-learning systems can use historical execution data, source-code characteristics, issue histories, and defect records to support testing decisions. Generative AI can also assist with converting natural-language requirements into candidate test cases, proposing boundary conditions, and summarizing failures.

Ajorloo et al. (2024) reviewed machine-learning approaches in software testing and classified methods across several learning paradigms. Their review provides a basis for understanding AI as a family of approaches rather than a single technology. Different techniques have different data requirements, strengths, evaluation methods, and limitations.

AI adoption should therefore be evaluated in relation to the specific quality task. A model that accurately predicts defect-prone modules may not necessarily generate valid test cases, while a generative system that produces readable test scenarios may still fail to identify the most critical business risks.

The AAT Triangle responds to this diversity by assigning AI the role of an intelligent assistant rather than treating any single AI tool as an autonomous quality authority.

2.2 The Role of the Analyst in Software Quality

The analyst connects stakeholder expectations with system requirements, process models, acceptance criteria, and business rules. This role is especially important when requirements are incomplete, ambiguous, contradictory, or dependent on organizational context.

For example, an AI system may generate a test case for an online payment process that checks whether a successful transaction produces a confirmation message. An analyst may recognize that the requirement must also specify transaction limits, authorization rules, duplicate payment prevention, refund conditions, and exception handling.

The analyst contributes contextual understanding and prioritization. The role includes interpreting requirements, resolving ambiguity, identifying dependencies, assessing business impact, and ensuring that quality activities reflect actual stakeholder needs.

Within the AAT Triangle, the analyst also reviews AI-generated suggestions, identifies unsupported assumptions, and translates validated requirements into measurable acceptance criteria.

2.3 The Role of the Tester

The tester evaluates whether software behaviour conforms to defined requirements and quality expectations. Testing includes test design, execution, defect identification, regression verification, exploratory testing, and the assessment of evidence.

The tester's role extends beyond executing predetermined scripts. Independent evaluation is necessary because AI-generated outputs may contain omissions, false assumptions, duplicated scenarios, or invalid assertions.

The International Software Testing Qualifications Board's AI testing guidance addresses challenges associated with AI-based systems, including non-deterministic behaviour, bias, transparency, explainability, and data dependence. Its current AI testing syllabus also emphasizes the lifecycle-specific evaluation of AI-based systems (International Software Testing Qualifications Board [ISTQB], 2026).

This distinction is important. There are two related but different quality challenges:

- **Using AI to test conventional software:** AI assists with activities such as test generation, prioritization, and defect analysis.
- **Testing AI-based software:** The system under test contains AI components whose behaviour may be probabilistic, data-dependent, or difficult to predict using conventional expected-output assertions.

The AAT Triangle can support both situations, provided that the testing strategy explicitly addresses the relevant risks.

2.4 Human–AI Collaboration

Human–AI collaboration involves the allocation of tasks between computational systems and people according to their respective capabilities. AI can process information rapidly, identify patterns, and generate alternatives. Human professionals contribute contextual judgment, ethical reasoning, accountability, and the ability to challenge outputs that appear plausible but lack sufficient evidence.

Effective collaboration requires clear decision rights. If people routinely accept AI-generated recommendations without scrutiny, the organization may create automation bias. Conversely, if AI outputs require extensive manual correction, the expected efficiency benefits may not materialize.

The proposed framework therefore distinguishes between generating a recommendation, validating its relevance, and approving a quality decision. These are separate activities and should be evaluated separately.

2.5 Software Quality Standards and Risk Management

ISO/IEC 25010:2023 provides a reference model for specifying and evaluating product quality. Its characteristics can help teams translate broad quality expectations into explicit objectives, acceptance criteria, and evaluation measures (ISO, 2023).

The NIST AI Risk Management Framework provides complementary guidance for managing AI-related risks. Its emphasis on governance, context, measurement, and risk management is relevant when AI-generated outputs influence software release decisions (Tabassi, 2023).

These resources offer useful foundations for the AAT Triangle. However, the proposed triangle is not a replacement for a formal quality standard or AI risk management framework. Instead, it is a coordination model that can be used alongside such standards to organize the work of AI systems, analysts, and testers.

2.6 Synthesis of the Literature

The reviewed literature and standards support three observations.

First, AI offers multiple ways to assist software testing, but the value of each technique depends on its task, data, evaluation criteria, and operating context.

Second, software quality requires alignment between requirements, testing activities, and defined quality objectives.

Third, AI-related risks require oversight and evaluation rather than unquestioned reliance on generated outputs.

The AAT Triangle combines these observations into a proposed operational framework. Its principal contribution is the explicit coordination of AI-generated intelligence, analyst-led interpretation, and tester-led verification within one feedback-oriented process.

3. Research Gap

Existing work on AI-supported software testing examines techniques such as machine-learning-based defect prediction, automated test generation, test-suite optimization, and intelligent test execution. Quality standards provide models for defining product quality, while AI governance frameworks address risks arising from the development and use of AI.

However, these areas do not automatically provide an operational arrangement for coordinating AI outputs with analyst judgment and independent tester verification across the software quality lifecycle.

The research gap addressed in this paper is therefore **the need for a clearly specified, human-accountable coordination model linking AI-assisted quality activities, business and requirements analysis, and independent software testing through traceable feedback.**

This gap is a conceptual proposition derived from the scope of the literature reviewed here. It should not be interpreted as proof that no similar framework exists. A systematic review of relevant databases would be required to establish the novelty of the triangle comprehensively.

The proposed framework contributes by:

1. Defining three distinct but interdependent quality roles.
2. Establishing the information exchanged between those roles.
3. Introducing a feedback mechanism connecting test outcomes to analysis and AI-supported activities.
4. Linking workflow effectiveness to measurable quality and efficiency indicators.
5. Making human responsibility and verification explicit in AI-assisted quality decisions.

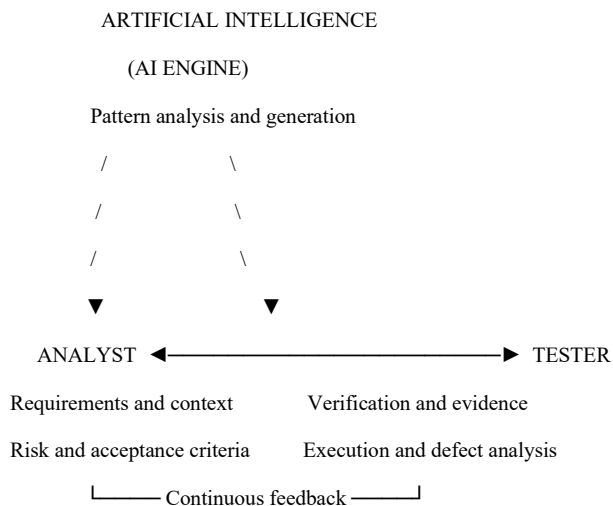
4. Conceptual Framework: The AI–Analyst–Tester Triangle

4.1 Conceptual Foundation

The AI–Analyst–Tester Triangle is a proposed framework for coordinating intelligent assistance, contextual interpretation, and independent verification.

The triangle represents three mutually reinforcing functions rather than three isolated departments. Each component contributes a different form of capability, and the quality management process becomes stronger when the outputs of one component are evaluated and improved by the others.

Figure 1. Conceptual representation of the AAT Triangle



Source: Author's proposed conceptual model.

The diagram represents the three principal functions. The continuous feedback process connects test evidence with subsequent requirements analysis and improvements to AI-supported activities.

4.2 Component One: Artificial Intelligence

AI functions as the computational intelligence component of the framework.

Its principal responsibilities include:

- Analysing requirements documents and identifying possible ambiguities.
- Generating candidate test cases and boundary conditions.
- Suggesting negative, exception, and regression scenarios.
- Supporting defect clustering and classification.
- Identifying potentially high-risk modules from historical evidence.
- Summarizing test execution logs.
- Suggesting regression test prioritization.
- Identifying possible gaps in requirement-to-test traceability.
- Producing draft quality reports for human review.

AI outputs should be treated as recommendations requiring appropriate validation. The reliability of these recommendations depends on the quality of the input data, the model's capabilities, the task definition, and the criteria used to evaluate results.

AI should not independently authorize a release merely because a generated report indicates that testing is complete.

4.3 Component Two: The Analyst

The analyst functions as the contextual interpretation and quality-planning component.

Principal responsibilities include:

- Clarifying stakeholder requirements.
- Identifying missing, inconsistent, or ambiguous requirements.
- Translating business needs into measurable acceptance criteria.
- Evaluating AI-generated scenarios against business rules.
- Prioritizing quality risks according to operational impact.
- Identifying dependencies among requirements and software components.
- Ensuring traceability between requirements, test cases, and defects.
- Determining whether proposed quality metrics are relevant.
- Reviewing unresolved risks before release recommendations are made.

The analyst is responsible for contextual relevance rather than merely checking whether an AI-generated response is grammatically correct or technically plausible.

4.4 Component Three: The Tester

The tester functions as the independent verification and evidence-generation component.

Principal responsibilities include:

- Reviewing test scenarios and test-data suitability.
- Designing and executing functional and non-functional tests.
- Verifying acceptance criteria.
- Evaluating AI-generated test cases for validity and completeness.
- Recording defects with reproducible evidence.
- Conducting regression and exploratory testing.
- Verifying corrections and assessing possible side effects.
- Reporting residual risks and unresolved defects.
- Maintaining evidence for release decisions.

The tester must retain the authority to reject a proposed test case, challenge a misleading AI conclusion, or escalate an unresolved quality risk.

4.5 Interaction Among the Three Components

The three components exchange information through a continuous process.

AI to analyst: AI identifies patterns, suggests scenarios, and highlights possible requirement gaps. The analyst evaluates their relevance and correctness.

Analyst to tester: The analyst converts validated requirements and priorities into acceptance criteria, risk assessments, and test objectives.

Tester to AI: Test outcomes, defect classifications, and coverage gaps can provide feedback for refining future AI-supported recommendations, subject to data quality, privacy, and governance controls.

Tester to analyst: Defects and unexpected behaviour may reveal incorrect assumptions, incomplete requirements, or missing acceptance criteria.

Analyst to AI: Clarified requirements, approved terminology, and contextual constraints improve the information available for subsequent AI-assisted activities.

This interaction distinguishes the AAT Triangle from a simple sequence in which AI produces test cases and humans merely execute them. The proposed model requires reciprocal evaluation and feedback.

4.6 Core Principles of the Framework

The framework is based on seven principles:

1. **Human accountability:** Responsibility for quality decisions remains clearly assigned to authorized people.
2. **Independent verification:** AI-generated outputs must be tested or reviewed according to their risk and intended use.
3. **Requirement traceability:** Quality activities should be linked to requirements, acceptance criteria, and recorded evidence.

4. **Risk-based prioritization:** Testing effort should reflect the probability and impact of potential failures.
5. **Continuous feedback:** Defects and test outcomes should inform future analysis and testing.
6. **Transparency:** The basis of significant recommendations and quality decisions should be documented.
7. **Proportional oversight:** Higher-risk software and AI-assisted decisions require more rigorous review.

These principles are intended to guide implementation rather than constitute a claim that the proposed framework has already been empirically validated.

5. Proposed Operating Model

The AAT Triangle can be implemented through an eight-stage process.

Stage 1: Requirement Collection

Stakeholder requirements, user stories, process documents, existing specifications, and relevant business rules are collected.

Primary responsibility: Analyst.

AI contribution: Document summarization, classification, and identification of potentially ambiguous statements.

Output: An initial requirements baseline.

Stage 2: Requirement Interpretation

The analyst evaluates the requirements, resolves ambiguity, and defines expected behaviour.

Primary responsibility: Analyst.

AI contribution: Suggestions for missing scenarios, conflicting statements, and boundary conditions.

Output: Reviewed requirements and measurable acceptance criteria.

Stage 3: Risk Assessment

The team identifies potential failures and evaluates their likelihood, impact, detectability, and business consequences.

Primary responsibility: Analyst, with input from the tester and relevant stakeholders.

AI contribution: Analysis of historical defect patterns and suggestions for prioritizing high-risk areas.

Output: A documented risk register and prioritized testing objectives.

Stage 4: Test Design

Candidate test scenarios are created for normal operations, boundary conditions, invalid inputs, exception handling, integration behaviour, and relevant non-functional characteristics.

Primary responsibility: Tester.

AI contribution: Generation of candidate scenarios and suggestions for test-data variation.

Output: A reviewed test suite linked to requirements and risks.

Stage 5: Test Validation and Execution

The tester reviews test validity, executes approved tests, and records results.

Primary responsibility: Tester.

AI contribution: Test-script assistance, execution-log analysis, and candidate defect identification.

Output: Test execution evidence and preliminary defect reports.

Stage 6: Defect Analysis

Defects are classified, prioritized, and assessed for their likely causes and business implications.

Primary responsibility: Analyst and tester, supported by developers and relevant specialists.

AI contribution: Defect clustering, duplicate-report detection, and possible root-cause suggestions.

Output: A prioritized defect register and corrective-action plan.

Stage 7: Quality Review and Release Recommendation

The team evaluates test coverage, unresolved defects, residual risks, and acceptance criteria.

Primary responsibility: Authorized release decision-maker, informed by the analyst and tester.

AI contribution: Consolidation of evidence and identification of incomplete records.

Output: A documented release recommendation.

Stage 8: Feedback and Continuous Improvement

The team reviews escaped defects, false positives, missed scenarios, test effectiveness, and the quality of AI-generated recommendations.

Primary responsibility: Joint responsibility across the quality team.

AI contribution: Trend analysis and suggestions for improving future testing.

Output: Updated test assets, revised requirements where necessary, and documented process improvements.

Table 1. Role allocation within the AAT Triangle

Quality activity	AI	Analyst	Tester
Requirement analysis	Suggests patterns and gaps	Interprets and validates	Identifies testability issues
Risk identification	Supports pattern detection	Evaluates business impact	Evaluates failure exposure
Test generation	Generates candidate cases	Validates business relevance	Reviews technical testability
Test execution	Assists with automation	Clarifies expected outcomes	Executes and verifies
Defect management	Suggests classification	Assesses business priority	Records and reproduces defects
Release assessment	Summarizes evidence	Evaluates requirement fulfilment	Reports test results and residual risks
Continuous improvement	Analyses trends	Revises context and priorities	Improves test assets

Source: Author's proposed framework.

The table describes proposed responsibilities. Actual allocation should reflect organizational structure, risk level, technical capability, and applicable governance requirements.

6. Research Methodology

6.1 Research Design

This study adopts a conceptual and descriptive-analytical research design. It develops a proposed framework by synthesizing selected research on AI-supported testing, software quality standards, AI risk management, and professional testing guidance.

The purpose of the present paper is framework development, not the reporting of a completed field experiment. Accordingly, no survey responses, experimental measurements, organizational case-study data, or statistical test results are claimed.

6.2 Research Approach

The conceptual development follows four activities:

1. Identification of the principal challenges associated with AI-assisted software quality management.
2. Examination of literature and standards relevant to AI-supported testing and quality evaluation.
3. Synthesis of complementary responsibilities for AI, analysts, and testers.
4. Development of a workflow, proposed hypotheses, and evaluation metrics for subsequent empirical testing.

The literature discussion is selective rather than a formally completed systematic literature review. A future systematic review should define database coverage, search strings, inclusion and exclusion criteria, screening procedures, and a reproducible quality-appraisal method.

6.3 Sources of Information

The paper draws on selected scholarly literature and institutional resources, including:

- Peer-reviewed research on machine learning in software testing.
- ISO/IEC 25010:2023 for software product quality.
- NIST AI Risk Management Framework 1.0 for AI risk governance.
- ISTQB guidance on testing AI-based systems and AI-related testing practices.

These sources provide a conceptual foundation for the proposed framework. They do not constitute empirical evidence that the AAT Triangle outperforms alternative approaches.

6.4 Proposed Empirical Validation

The next research phase should evaluate the framework in real or controlled software development environments.

A suitable design would compare two groups:

- **Comparison group:** Conventional software quality management using the organization's existing testing practices.
- **Intervention group:** Software quality management using the AAT Triangle, with documented AI assistance, analyst validation, independent testing, and feedback.

References:

1. Ajorloo, S., Jamarani, A., Kashfi, M., Haghi Kashani, M., & Najafizadeh, A. (2024). A systematic review of machine learning methods in software testing. *Applied Soft Computing*, 162, Article 111805. <https://doi.org/10.1016/j.asoc.2024.111805>
2. Amalfitano, D., Faralli, S., Rossa-Hauck, J. C. R., Matalonga, S., & Distanto, D. (2024). Artificial intelligence applied to software testing: A tertiary study. *ACM Computing Surveys*, 56(3), Article 58. <https://doi.org/10.1145/3616372>
3. Escalante-Viteri, A., & Mauricio, D. (2025). Artificial intelligence in software testing: A systematic review of a decade of evolution and taxonomy. *Algorithms*, 18(11), Article 717. <https://doi.org/10.3390/a18110717>
4. International Organization for Standardization. (2023). *ISO/IEC 25010:2023: Systems and software engineering—Systems and software Quality Requirements and Evaluation (SQuaRE)—Product quality model*. <https://www.iso.org/standard/78176.html>
5. International Software Testing Qualifications Board. (2026). *Certified Tester AI Testing (CT-AI), Version 2.0*. <https://istqb.org/certifications/certified-tester-ai-testing-ct-ai/>
6. McMinn, P. (2004). Search-based software test data generation: A survey. *Software Testing, Verification and Reliability*, 14(2), 105–156. <https://doi.org/10.1002/stvr.294>
7. Myers, G. J., Sandler, C., & Badgett, T. (2011). *The art of software testing* (3rd ed.). John Wiley & Sons.
8. Pecorelli, F., Grano, G., Palomba, F., Gall, H. C., & De Lucia, A. (2024). Toward granular search-based automatic unit test case generation. *Empirical Software Engineering*, 29, Article 71. <https://doi.org/10.1007/s10664-024-10451-x>
9. Pressman, R. S., & Maxim, B. R. (2020). *Software engineering: A practitioner's approach* (9th ed.). McGraw Hill.