TRACE-TEST: An Evidence-Linked Software Quality Assurance Framework for AI-Augmented Testing in Regulated Platforms

Rajeew Vishvakarma

Project Manager, American Express(Contract)

Abstract

Artificial intelligence is increasingly used in software testing for test generation, regression prioritization, repair, and defect analysis. Most studies, however, optimize for local gains such as speed, coverage, or prompt quality. From a software quality perspective, that leaves an important gap. Regulated platforms in banking, healthcare, insurance, and public services require testing workflows that improve quality while also preserving traceability, evidence retention, human oversight, and defensible release decisions. This paper develops TRACE-TEST, an evidence-linked framework for software quality assurance in AI-augmented testing. The paper combines a targeted integrative review of recent empirical testing studies and current governance sources with a design-science artefact proposal. The review shows two simultaneous realities. The technical potential of LLM-based testing is rising quickly. Industrial adoption, standardized evidence practices, and post-deployment feedback loops remain immature. TRACE-TEST addresses that gap by linking regulations and controls, requirements, risk classification, AI-generated artefacts, reviewer actions, execution evidence, monitoring signals, and release sign-off into a single quality-assurance chain. The paper makes four contributions. First, it identifies five research gaps that matter most for high-assurance software quality. Second, it synthesizes recent empirical evidence on coverage, mutation effectiveness, traceability, practitioner oversight, and software evolution. Third, it specifies a minimum evidence object and a governed release-assurance workflow. Fourth, it proposes an evaluation model that combines technical testing metrics with quality-assurance metrics such as traceability completeness and evidence readiness. The main contribution is not simply more AI in testing. It is a framework that makes AI-assisted testing more reviewable, measurable, and software-quality aligned in regulated environments.

Keywords: software quality; software quality assurance; AI-augmented testing; software testing; traceability; auditability; human oversight; release governance; risk-based testing

1. Introduction

Artificial intelligence is reshaping software engineering, and software testing is one of the fastest-moving application areas. Recent surveys show a large and growing body of work on large language models in testing. The 2024 survey by Wang et al. synthesizes 102 studies and finds strong concentration in test-case preparation and program repair. At the same time, more recent adoption studies report that testing has not experienced the same production breakthrough seen in coding assistants. Reported real-world benefits remain narrow, and industrial implementations are still limited and uneven [1]-[3]. For software quality research, this creates a clear opportunity. The field now needs frameworks that connect AI-enabled testing gains to quality management, release confidence, and reproducible engineering evidence.

In regulated platforms, that gap matters more than it does in ordinary consumer software. Banking, payments, insurance, healthcare, and public-sector systems do not evaluate a release only by whether

defects were found quickly. They also evaluate whether teams can justify what was tested, why certain risks were prioritized, which reviewer accepted or changed an AI-generated artefact, what evidence was preserved, and how the decision can be reconstructed later. In such settings, testing is not just a quality-control function. It is also a software quality assurance and evidence-production function.

Current governance frameworks reinforce that shift. NIST AI RMF 1.0 treats AI risk management as a lifecycle activity organized around Govern, Map, Measure, and Manage, and ties trustworthy AI to characteristics such as validity, reliability, accountability, transparency, explainability, and resilience [4]. ISO/IEC 42001 defines an AI management system as a structured way to establish, implement, maintain, and continually improve AI governance within an organization [5]. The EU AI Act's record-keeping and human-oversight requirements further push high-risk systems toward automatic logging, traceability, and operational mechanisms for human intervention [6]-[8].

The problem is that most AI-for-testing literature does not yet solve the evidentiary burden that regulated platforms face. Much of the current work focuses on coverage, generation quality, prompt strategies, or developer productivity. Those contributions are valuable, but they do not fully answer software-quality questions such as the following. What minimum evidence must be preserved for an AI-generated test artefact? How should requirement-to-test-to-release traceability be maintained? Which reviewer actions should be logged? When should production incidents or policy changes trigger regression re-prioritization? And how should a release fail assurance even if tests pass but evidence is incomplete?

This paper addresses those questions through TRACE-TEST: an evidence-linked framework for AI-augmented testing in regulated platforms. The paper does not claim a new proprietary benchmark or a fabricated industrial experiment. Instead, it contributes a literature-grounded framework and an explicit evidence architecture that can support real validation studies. The central claim is simple. In regulated settings, the value of AI-augmented testing should be judged not only by automation benefit, but by whether the resulting workflow remains reviewable, traceable, measurable, and auditable across the release lifecycle.

2. Method: targeted integrative review and artefact development

This paper uses a targeted integrative review combined with design-science artefact development. The review is targeted rather than fully systematic. It concentrates on the subset of literature most relevant to regulated AI-assisted testing: official governance sources, recent empirical studies on LLM-based software testing, empirical studies on AI adoption in testing, and recent work on traceability and software evolution. The review window was centered on 2023-2026 because that period captures the rapid shift from early LLM experimentation to more mature empirical and governance-oriented discussion.

Sources were selected using four inclusion rules. First, governance sources had to be official or near-official documents that influence organizational practice, such as NIST AI RMF 1.0, ISO/IEC 42001 materials, and EU AI Act support materials. Second, software-testing papers had to report either empirical results or structured evidence on LLM-enabled testing tasks. Third, papers were prioritized when they offered quantitative signals that could inform metric design, such as statement coverage, branch coverage, mutation score, adoption depth, or traceability F1. Fourth, papers were retained if they materially informed one of the five research gaps developed later in the paper.

The design-science component converts those review findings into a prescriptive artefact. TRACE-TEST is specified as a release-assurance framework rather than as a single model or tool. It defines layers, evidence objects, reviewer actions, and monitoring triggers. This is important because regulated testing failures are often architectural rather than algorithmic. The challenge is not only whether AI can generate a useful test. The challenge is whether the organization can govern the artefact from prompt to sign-off.

Because this paper is positioned as a framework and evidence-architecture contribution, the empirical role of the literature is to anchor design choices and evaluation criteria. The paper therefore reports literature-derived empirical signals, not new experimental findings. This keeps the manuscript methodologically honest and makes the next empirical step explicit.

3. Software quality context for regulated platforms

For the purposes of this paper, a regulated platform is a software-intensive platform operating in a domain where release decisions interact with formal internal controls, external compliance obligations, or sector-level oversight. The definition is intentionally broader than the EU AI Act's high-risk category. A digital banking onboarding flow, payment rails orchestration layer, health information system, or claims workflow engine can all require documented release justification even when the embedded AI feature itself is not formally classified as high-risk. The common property is evidentiary pressure.

NIST AI RMF provides a useful baseline because it is sector-agnostic and lifecycle-oriented. Its Govern, Map, Measure, and Manage functions emphasize that AI risk management is not a one-time technical validation task. It is an ongoing socio-technical discipline [4]. For testing teams, that means testing outputs should connect to intended use, risk treatment, measurement, and change management. That idea also appears in NIST's 2026 monitoring report, which argues that post-deployment monitoring is necessary but still underdeveloped and fragmented in practice [9].

ISO/IEC 42001 extends that logic to organizational management systems. Its core relevance here is not a detailed test method. Its relevance is the expectation that AI-related activities are documented, reviewable, and improved continuously [5]. In other words, ad hoc AI assistance in test design is weak governance. Documented AI assistance inside a controlled workflow is stronger governance.

The EU AI Act sharpens the operational implications. Article 12 requires high-risk AI systems to allow automatic recording of events over the lifetime of the system to support traceability. Article 14 requires effective human oversight, including mechanisms that let qualified people understand, monitor, and intervene when needed [6]-[8]. These provisions do not tell testing teams exactly how to structure a release package. But they clearly indicate the direction: logs, evidence, and human control are not optional niceties. They are design requirements.

4. Empirical signals from recent software quality and testing literature

The recent empirical literature sends a mixed but useful signal. On the one hand, LLM-based testing techniques can be technically strong. Schäfer et al. evaluate TestPilot on 25 npm packages covering 1,684 API functions and report median statement coverage of 70.2% and branch coverage of 52.8%, outperforming Nessie on both metrics [10]. Dakhel et al. show that mutation-aware prompting improves defect-revealing power, detecting up to 28% more faulty human-written snippets and achieving a 93.57% mutation score on synthetic buggy code [11]. Those results show that AI assistance can materially improve test generation and bug exposure when evaluated carefully.

On the other hand, stronger and more recent benchmarks show that technical gains are less stable than early optimism suggested. Huang et al. introduce the ULT benchmark with 3,909 real-world function-level tasks and report average performance that is much lower than on easier or potentially contaminated benchmarks: 41.32% accuracy, 45.10% statement coverage, 30.22% branch coverage, and 40.21% mutation score [12]. Haroon et al. then show that even when baseline performance is strong on original programs, generated tests degrade under software evolution. Across eight LLMs and 22,374 program variants, line coverage reaches 79% and branch coverage 76% on original programs, but under semantic-altering changes pass rate falls to 66% and branch coverage to 60% [13]. These findings are especially important for regulated release workflows, where change stability matters as much as initial performance.

Traceability research adds a second important empirical signal. Alor et al. evaluate LLMs for documentation-to-code traceability and report best-model F1 scores of 79.4% and 80.4% across two datasets, clearly outperforming classical baselines [14]. Yet the same paper shows weaker performance for full relationship explanations, with fully correct explanations ranging from 42.9% to 71.1%. The implication is direct. AI can assist in discovering links, but human review remains necessary when those links must support downstream governance or release decisions.

Practitioner work points in the same direction. Santana et al. interview 15 software testers and find that LLM use is iterative, reflective, and heavily dependent on human validation because of hallucinations, privacy concerns, and inconsistent reasoning [15]. Karhu et al. and related secondary studies similarly find that industrial AI adoption in software testing remains limited, with a gap between broad expectations and observed deployment value [2]-[3]. This means a publishable framework should not assume frictionless adoption. It should explicitly design for oversight, reviewer workload, and evidence capture.

A final signal concerns input quality. Vogelsang et al. show that requirements smells can affect LLM performance in automated traceability tasks, even if the effect is not uniform across all subtasks [16]. For regulated testing, this matters because requirement text is often the starting point for AI-generated test assets. A weak requirements corpus creates a weak prompt foundation. That means requirement quality checks are not a side issue. They are part of the testing control plane.

Table 1A. Recent empirical signals that shape the framework (part 1 of 2).

Study	Study type	Scope	Key quantitative signal	Implication for this manuscript
Wang et al. (2024) [1]	Survey	102 LLM-testing studies	Test-case preparation and program repair dominate	The field is broad, but evidence models are still thin.
Karhu et al. (2025) [2]	Secondary study	Industry-context AI adoption in testing	AI has not yet made a significant breakthrough in testing practice	Frameworks must address adoption friction, not assume maturity.
Karhu et al. (2025) [3]	Secondary study	Barriers and enablers	Lack of perceived value and reference implementations slows adoption	A reusable governed workflow is a practical contribution.
Schäfer et al. (2023)	Empirical evaluation	25 npm packages;	70.2% median	AI test generation can

Study	Study type	Scope	Key quantitative signal	Implication for this manuscript
[10]		1,684 API functions	statement coverage; 52.8% branch coverage	materially improve technical quality.
Dakhel et al. (2023) [11]	Empirical evaluation	Mutation-aware prompting	Up to 28% more faulty snippets detected; 93.57% mutation score on synthetic buggy code	Coverage alone is insufficient; bug-revealing power matters.

Note. Quantitative values are literature-derived and used to motivate framework design and metric selection.

Table 1 continues on the next page.

Table 1B. Recent empirical signals that shape the framework (part 2 of 2).

Study	Study type	Scope	Key quantitative signal	Implication for this manuscript
Huang et al. (2025) [12]	Benchmark study	ULT benchmark; 3,909 tasks	41.32% accuracy; 45.10% statement coverage; 30.22% branch coverage; 40.21% mutation score	Leakage-aware, realistic evaluation is essential.
Alor et al. (2025) [14]	Empirical evaluation	Two doc-to-code datasets	Best F1 = 79.4% and 80.4%; weaker explanation quality	Traceability can be AI-assisted, but explanations still need review.
Santana et al. (2025) [15]	Interview study	15 testers	Iterative use with strong human validation due to hallucinations and inconsistency	Reviewer actions should be first-class evidence.
Vogelsang et al. (2025) [16]	Empirical study	Requirements smells in prompts	Requirement quality affects automated traceability performance	Requirement quality belongs in the testing control plane.
Haroon et al. (2026) [13]	Large-scale empirical study	8 LLMs; 22,374 program variants	Under semantic-altering changes: pass rate 66%; branch coverage 60%	Monitoring and change-driven retesting are necessary.

Note. These values are reproduced from the cited studies and are not original results from the present manuscript.

5. Top research gaps in software quality assurance and framework response

Taken together, the reviewed literature supports five research gaps that matter for regulated AI-augmented testing.

Table 2. Top five research gaps and how TRACE-TEST responds.

Gap	Why the gap matters	TRACE-TEST response
1. Compliance-aware objectives are weak	Most studies optimize for speed, coverage, or generation quality.	Introduce risk weighting, release evidence readiness, and assurance metrics.
2. End-to-end traceability is under-specified	AI outputs are rarely linked to requirement, control, reviewer, execution, and release records.	Define a minimum evidence object and central evidence store.
3. Post-deployment linkage is thin	Testing is still treated mainly as a pre-release activity.	Convert incidents, drift, policy changes, and overrides into retest triggers.
4. Audit-ready release packages are missing	Papers rarely define what evidence must exist before sign-off.	Specify release assurance gate conditions and required artefacts.
5. Human oversight is acknowledged but weakly measured	Practitioner studies show oversight matters, but metrics are inconsistent.	Log accept/edit/reject/escalate decisions and review effort.

6. TRACE-TEST: framework specification for software quality assurance

TRACE-TEST stands for Traceable, Audit-Ready, Compliance-aware, Evidence-linked Testing. It is designed as a governed workflow rather than as a single AI component. The framework treats testing as a defect-detection activity, a software quality assurance activity, and an evidence-production activity. That combined role is what makes it suitable for regulated platforms.

The framework begins with regulations and controls, requirements, change information, and historical quality signals. Those inputs are not merely background context. They define why certain tests matter more than others. TRACE-TEST therefore introduces an explicit risk classification stage before AI artefacts enter human review. This is a key departure from many current AI-testing tools, which optimize for generation quality without formally encoding control severity, customer harm, or release sensitivity.

The AI test-support layer may include generation, prioritization, repair, or defect summarization. TRACE-TEST is intentionally model-agnostic. It does not depend on a particular LLM vendor, prompt strategy, or orchestration method. What it does require is that each AI-produced artefact is treated as a governed object. That means it can be inspected, edited, rejected, or escalated by a reviewer rather than silently consumed.

The human review gate is the control heart of the framework. Every AI-produced artefact receives one of four reviewer dispositions: accept, edit, reject, or escalate. These actions are preserved as evidence. This makes oversight operational rather than symbolic. It also aligns well with current practitioner findings, which show that useful LLM adoption in testing already depends heavily on reflective human validation [15].

Once approved, artefacts move to execution and results capture. TRACE-TEST then persists a minimum evidence object that links the artifact to its requirement, control, risk class, AI metadata, reviewer action, execution result, and release identifier. The evidence store is central because regulated release failures often emerge from missing reconstruction capability rather than from one isolated test defect. If an auditor, risk manager, or internal reviewer cannot reconstruct why an artefact existed and who approved it, the organization has an assurance problem even if the software behaved correctly in the short term.

The framework closes with release assurance and monitoring feedback. Release sign-off depends on both technical evidence and evidence completeness. A release may therefore fail assurance because high-risk requirements are insufficiently covered, because reviewer dispositions are missing, or because incident-driven retest triggers have not been resolved. After release, incidents, drift, policy changes, model changes, and elevated override rates feed back into regression reprioritization and the next evidence package. This directly addresses NIST's concern that post-deployment monitoring remains necessary but weakly standardized [9].

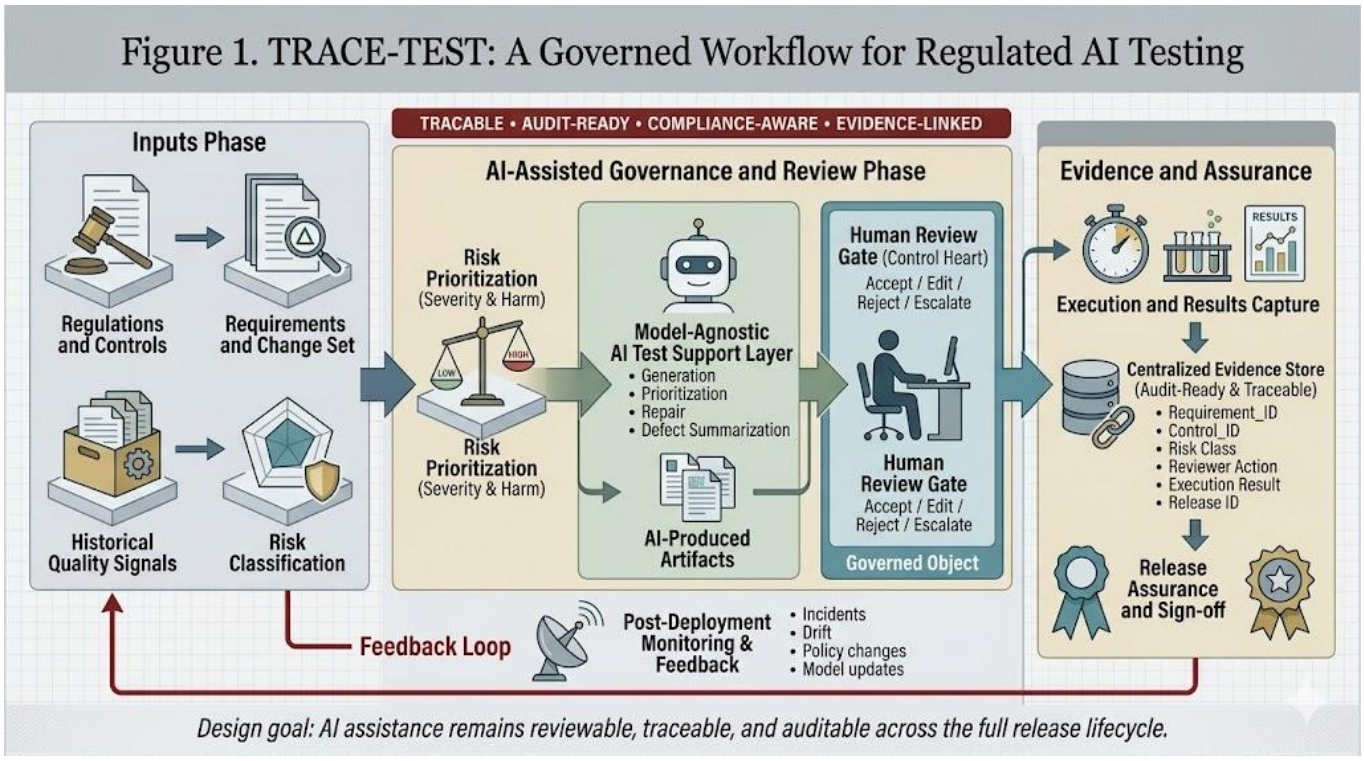

Figure 1. TRACE-TEST: A Governed Workflow for Regulated AI Testing

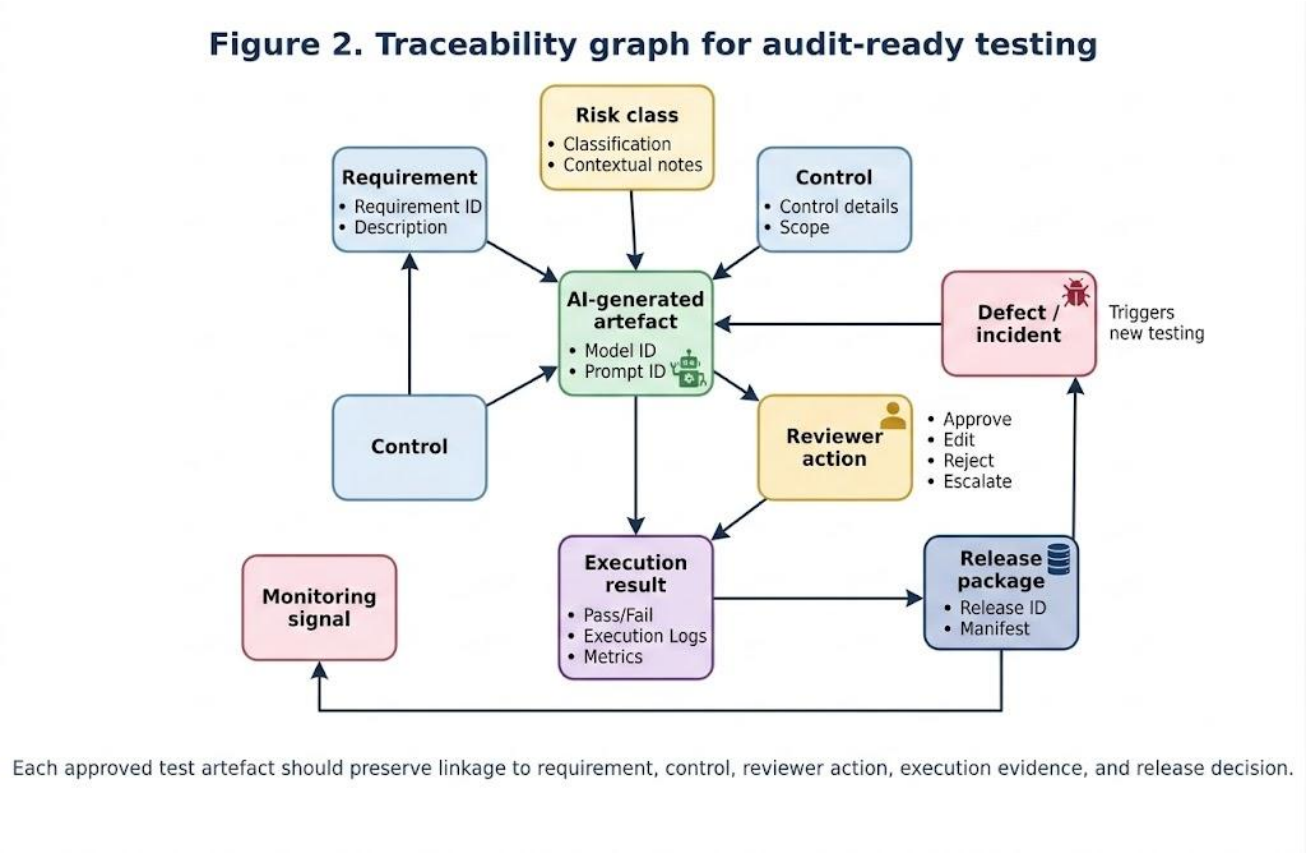

Figure 2. Traceability graph for audit-ready testing.

6.1 Crosswalk to governance requirements

TRACE-TEST is strongest when its components are mapped directly to real governance expectations. Table 3 provides a compact crosswalk from recognized governance requirements to concrete testing implications.

Table 3. Governance crosswalk from NIST AI RMF, ISO/IEC 42001, and the EU AI Act to testing implications.

Source	Governance expectation	Testing implication
NIST AI RMF Govern [4]	Defined policies, roles, accountability	Name reviewer roles; preserve approvals; require evidence completeness before sign-off
NIST AI RMF Map/Measure [4]	Context, risk, and performance measurement	Tag requirements by risk; evaluate both technical and assurance metrics
NIST monitoring report [9]	Post-deployment monitoring remains necessary but immature	Use incidents, drift, policy changes, and override rates as explicit retest triggers
ISO/IEC 42001 [5]	Documented and continually improved AI management system	Embed AI assistance inside versioned testing workflows and evidence retention
EU AI Act Article 12 [6]-[7]	Automatic logging and traceability	Persist artefact lineage, AI metadata, and execution evidence
EU AI Act Article 14 [8]	Effective human oversight	Support accept, edit, reject, and escalate decisions with qualified reviewers

6.2 Minimum evidence object

The minimum evidence object is the operational centerpiece of the framework. It defines the fields that must exist for an AI-assisted testing artefact to remain reconstructable later. Organizations may extend the schema, but deleting core fields would weaken traceability, oversight, or release reconstruction.

Table 4. Minimum evidence object for AI-assisted testing artefacts.

Field	Purpose	Typical type
evidence_id	Unique immutable identifier	String / UUID
requirement_id	Requirement or user-story link	String
control_id	Mapped policy or control identifier	String
risk_class	Criticality / compliance severity tier	Ordinal
artifact_type	Generated test, priority recommendation, repair, summary	Enum
model_version	Model or tool version used	String
prompt_or_context_ref	Prompt template or retrieval context reference	String
reviewer_action	Accept / edit / reject / escalate	Enum
reviewer_role	Qualified human role making the decision	String
execution_result	Pass / fail / skipped / blocked	Enum
defect_or_incident_link	Connected defect or production signal	String
release_id	Release package linkage	String
timestamp	When the evidence object was created or changed	Datetime
retention_status	Retention and archival state	Enum

6.3 Proposed release-readiness logic

A regulated release should not rely only on test pass rate. The framework therefore proposes a composite decision logic that combines technical sufficiency and evidence sufficiency. One practical formulation is:

$$RRI = w1 \cdot RWTE + w2 \cdot TC + w3 \cdot EC + w4 \cdot RER - w5 \cdot PESI$$

where RRI is release readiness index, RWTE is risk-weighted test effectiveness, TC is traceability completeness, EC is evidence completeness, RER is release evidence readiness, and PESI is post-release escape severity index. The formulation is normative rather than validated. Its role is to make release-governance reasoning explicit and measurable.

7. Quality evaluation model and future empirical protocol

A credible evaluation of TRACE-TEST should compare three workflows: a traditional manual testing baseline, a generic AI-assisted testing baseline, and the governed TRACE-TEST workflow. That three-way comparison matters. Without the generic AI baseline, any measured gain could be attributed simply

to adding AI. The real contribution of this paper is narrower. It is the additional value produced by governance, traceability, and evidence.

The paper therefore proposes a mixed metric set. Technical metrics include requirement coverage, defect detection rate, regression effectiveness, and false-positive suggestion rate. Assurance metrics include traceability completeness, evidence completeness, reviewer intervention rate, release evidence readiness, audit-preparation effort, and post-release escape severity. This metric design is grounded in the reviewed literature. Test generation studies justify conventional software metrics [10]-[13], while traceability, practitioner, and governance studies justify the added need for explanation quality, human validation, and evidence preservation [4]-[9], [14]-[16].

To make prioritization more realistic for regulated settings, the paper also proposes risk-weighted test effectiveness (RWTE). RWTE rewards early detection of defects in requirements with higher harm potential or stronger control significance, rather than treating all requirements equally. This better fits domains where missing a high-severity compliance condition is more damaging than missing a cosmetic defect. The paper presents RWTE as a proposed composite measure for future empirical work, not as a validated industry standard.

Finally, future empirical studies using TRACE-TEST should follow recent community guidance for LLM-based software engineering research. Baltes et al. recommend declaring LLM roles, model versions, prompts, tool architectures, baselines, human validation, and limitations in empirical SE studies involving LLMs [17]. That guidance should be built directly into TRACE-TEST reporting practice.

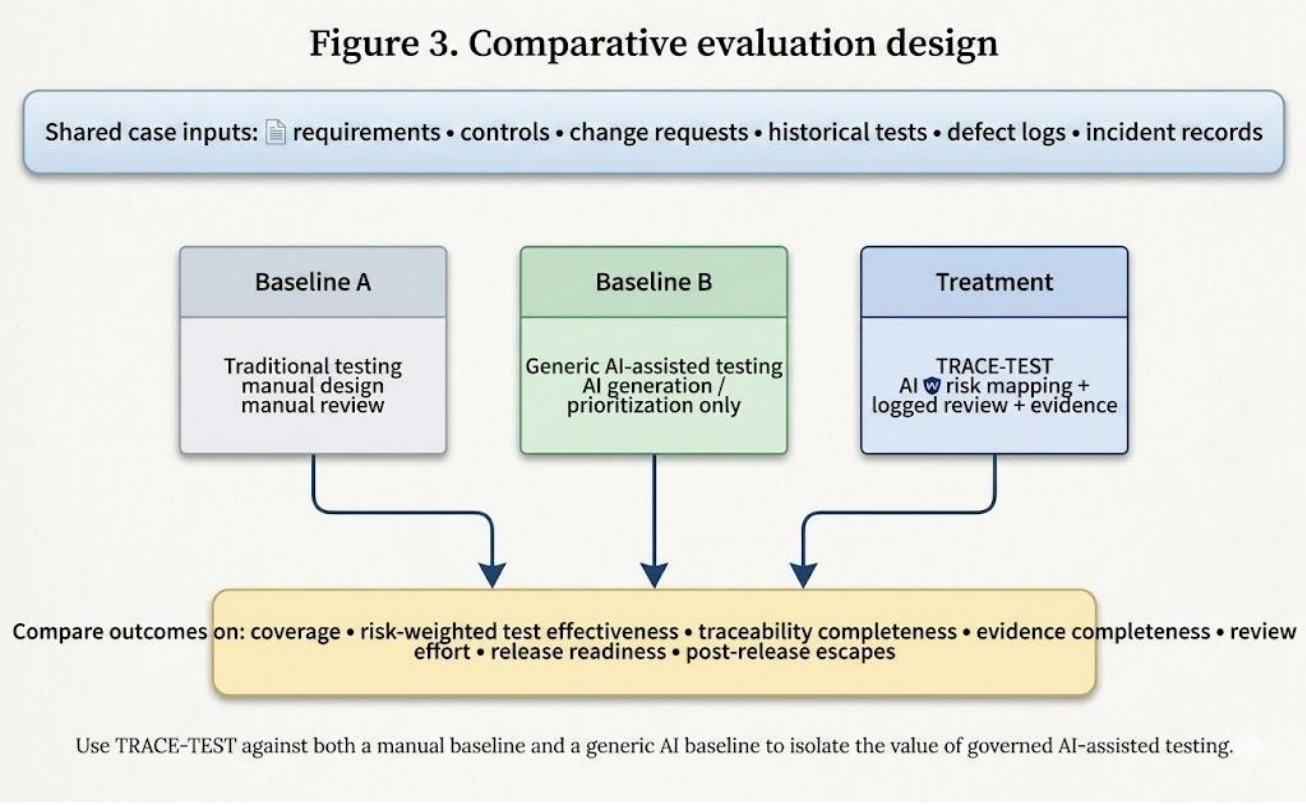

Figure 3. Comparative evaluation design.

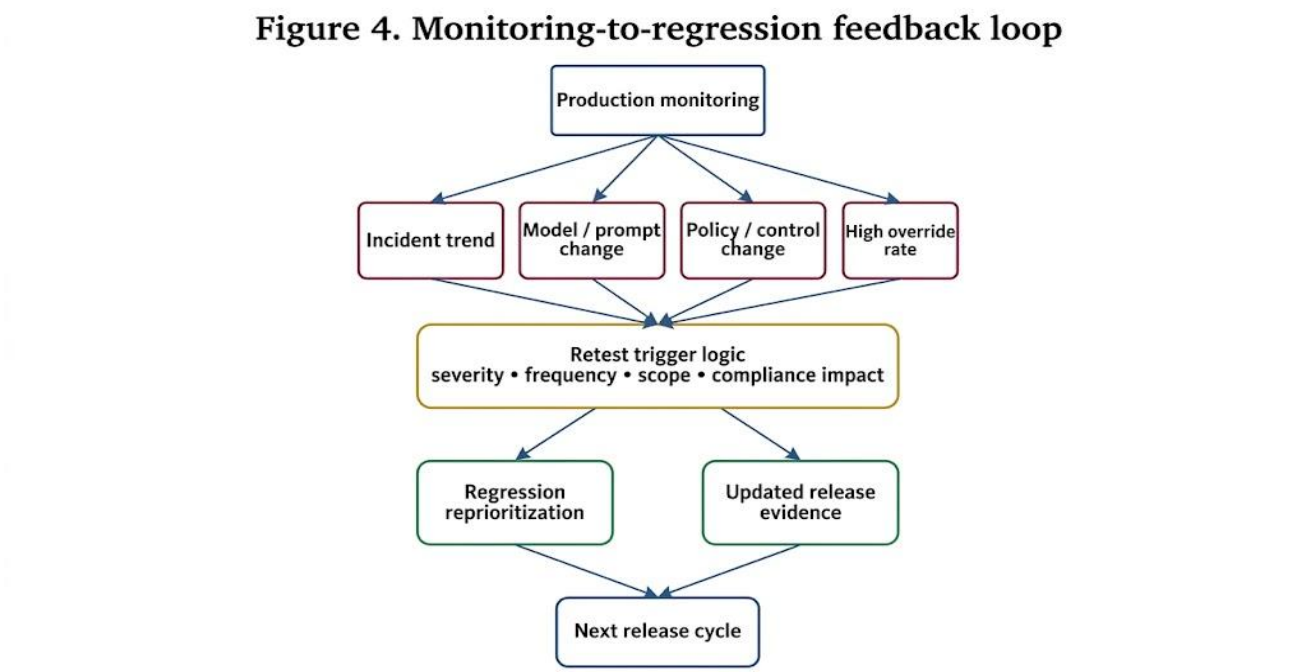

Figure 4. Monitoring-to-regression feedback loop.

Table 5. Proposed evaluation metrics for a future TRACE-TEST validation study.

Metric	Operational definition	Category
Requirement coverage	Percent of in-scope requirements exercised by at least one approved test	Technical
Defect detection rate	Detected defects divided by total known defects in the evaluation window	Technical
Regression effectiveness	Release-relevant defects detected by the selected regression subset	Technical
False-positive suggestion rate	AI suggestions later judged irrelevant, redundant, or incorrect	Technical
Traceability completeness	Approved artefacts linked to requirement, control, and release identifiers	Assurance
Evidence completeness	Approved artefacts with the full evidence object populated	Assurance
Reviewer intervention rate	Edited + rejected + escalated artefacts divided by all AI-produced artefacts	Assurance
Release evidence readiness	Share of mandatory evidence objects available before sign-off	Assurance
Audit-preparation effort	Person-hours required to produce release-review package	Assurance
Post-release escape severity	Weighted severity of defects escaping after release	Assurance

8. Discussion and implications for software quality management

The reviewed evidence supports a careful but optimistic conclusion. AI can improve software-testing tasks. It can help generate tests, improve coverage, detect more mutation-relevant faults, and support trace link discovery. But those gains do not automatically produce high-quality release assurance. The literature also shows benchmark fragility, limited industrial penetration, human-review dependence, and instability under software change [2]-[3], [12]-[16].

This is why the main contribution of this paper is architectural. TRACE-TEST reframes AI-assisted testing as a governed software-quality evidence system. That reframing matters for regulated platforms because it integrates software-testing performance with control evidence, accountability, and post-release response. Put differently, the paper argues that the next frontier is not only better prompts or stronger models. It is better evidence design for software quality assurance.

The paper also has practical implications for software quality managers, test leaders, and engineering leaders. Teams should not begin with end-to-end AI automation. They should begin with a narrow but high-value slice: risk tagging, artefact lineage, reviewer dispositions, and explicit retest triggers. That sequencing can reduce the organizational resistance documented in adoption studies while preserving enough structure for later audits, compliance reviews, and quality-management reporting.

The paper has limitations. It is a framework paper grounded in recent empirical literature, not a new multi-organization field experiment. Some of the cited studies are preprints, reflecting the speed of this research area. In addition, governance expectations vary by domain and jurisdiction. Future work should therefore validate TRACE-TEST in at least one real regulated platform, publish an open evidence schema, and test the framework's ability to reduce audit effort without suppressing testing productivity.

9. Conclusion

AI-assisted testing is no longer a speculative topic. The empirical literature shows both strong technical promise and important limits. Regulated platforms need a response that goes beyond faster generation. They need workflows that remain reviewable, traceable, measurable, and defensible over time.

TRACE-TEST is offered as that response. By connecting controls, requirements, risk classification, AI-generated artefacts, reviewer actions, execution evidence, monitoring signals, and release sign-off, the framework turns AI-augmented testing into an evidence-linked software quality assurance process. The paper's central argument is therefore straightforward. In regulated software, the value of AI-augmented testing should be judged not only by what it automates, but by how well it supports high-quality and audit-ready release assurance.

Declarations

Data Availability: This manuscript is a conceptual and evidence-grounded framework paper. No new dataset was generated, curated, or analyzed for this study. All supporting evidence is drawn from published literature and publicly available sources cited in the reference list.

Competing Interests: The author declares no competing interests.

Funding: This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Author Contributions: Rajeew Vishvakarma conceived the study, developed the framework, conducted the literature synthesis, analyzed the evidence base, prepared the figures and tables, and wrote, reviewed, and revised the manuscript. The author approved the final manuscript.

Corresponding Author: Correspondence to Rajeew Vishvakarma at rajeew.vishvakarma@gmail.com

References

[1] J. Wang, Y. Huang, C. Chen, Z. Liu, S. Wang, and Q. Wang, "Software Testing With Large Language Models: Survey, Landscape, and Vision," IEEE Transactions on Software Engineering, 2024; survey covering 102 studies.

[2] K. Karhu, J. Kasurinen, and K. Smolander, "Expectations vs Reality - A Secondary Study on AI Adoption in Software Testing," arXiv:2504.04921, 2025.

[3] K. Karhu, J. Kasurinen, and K. Smolander, "Barriers and Enablers of AI Adoption in Software Testing," ICSEA secondary study, 2025.

[4] NIST, "Artificial Intelligence Risk Management Framework (AI RMF 1.0)," NIST AI 100-1, 2023.

[5] ISO, "ISO/IEC 42001:2023 - Artificial intelligence management system," official overview, 2023.

[6] EU AI Act Service Desk, "Article 12: Record-keeping," 2025.

[7] Regulation (EU) 2024/1689, Artificial Intelligence Act, official framework overview and record-keeping requirements, 2024.

[8] EU AI Act Service Desk / explanatory materials, "Article 14: Human oversight," 2025.

[9] A. Rao et al., "Challenges to the Monitoring of Deployed AI Systems," NIST AI 800-4, 2026.

[10] M. Schäfer, S. Nadi, A. Eghbali, and F. Tip, "An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation," IEEE Transactions on Software Engineering, 2023.

[11] A. M. Dakhel et al., "Effective Test Generation Using Pre-trained Large Language Models and Mutation Testing," arXiv:2308.16557, 2023.

[12] D. Huang, J. M. Zhang, M. Harman, Q. Zhang, M. Du, and S.-K. Ng, "Benchmarking LLMs for Unit Test Generation from Real-World Functions: ULT," arXiv:2508.00408, 2025.

[13] S. Haroon, M. T. Khan, and M. A. Gulzar, "Evaluating LLM-Based Test Generation Under Software Evolution," arXiv:2603.23443, 2026.

[14] E. Alor, S. Khatoonabadi, and E. Shihab, "Evaluating the Use of LLMs for Documentation to Code Traceability," arXiv:2506.16440, 2025.

[15] M. D. Santana, C. Magalhaes, and R. de Souza Santos, "Software Testing with Large Language Models: An Interview Study with Practitioners," arXiv:2510.17164, 2025.

[16] A. Vogelsang, A. Korn, G. Broccia, A. Ferrari, J. Fischbach, and C. Arora, "On the Impact of Requirements Smells in Prompts: The Case of Automated Traceability," arXiv:2501.04810, 2025.

[17] S. Baltes et al., "Evaluation Guidelines for Empirical Studies in Software Engineering involving LLMs," arXiv:2508.15503, 2025.

[18] Z. Zhou, "Risk-based test framework for LLM features in regulated software," arXiv:2601.17292, 2026.