International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 10 | Oct - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

QUALTAN: A Governed Agentic Architecture for Specification-Driven and Execution-Grounded Autonomous Software Testing

By- **Mohan Gundala**

Abstract

Large Language Models (LLMs) are increasingly used to generate software tests, yet generated tests are probabilistic artifacts whose validity, relevance, and execution behavior must be verified independently. This paper presents the protocol for QUALTAN, a governed agentic architecture that separates LLM-based reasoning from deterministic test execution. QUALTAN combines specification-driven test intent, canonical test artifacts, policy checks before execution, execution-grounded validation, risk-aware test selection, and intent-preserving autonomous repair. We pre-specify six research questions, six hypotheses with falsification criteria, a four-condition controlled experiment (traditional automation, LLM-only generation, LLM generation with execution feedback, and the full architecture), a six-component ablation study, and a statistical analysis plan with predefined primary outcomes. The study uses open-source and synthetic applications with seeded defects and a public benchmark, and reports paired comparisons with corrected significance tests and effect sizes. Because the protocol is fixed before data collection, the claims it tests can be supported, refined, or rejected by the results. The paper therefore contributes a falsifiable experimental design for studying how governance and execution feedback affect the quality of LLM-generated test evidence.

Keywords: software testing, large language models, agentic systems, test generation, mutation testing, test oracles, reproducibility, empirical software engineering

1 Introduction

Large Language Models have become a major research direction in software engineering, with applications spanning requirements analysis, code generation, test generation, debugging, and maintenance. A 2025 research roadmap on LLM-based software testing, based on a semi-systematic literature review, describes rapid growth in this area and identifies unresolved challenges across the testing lifecycle [1].

Evaluation methodology shapes what such research can conclude. Huang et al. introduced ULT, a benchmark of real-world Python functions designed to reduce data contamination and to avoid the structural simplicity of toy programs. Their results show that LLM-generated tests achieve, on average, 41.32% accuracy, 45.10% statement coverage, 30.22% branch coverage, and a 40.21% mutation score, substantially below the performance reported on easier or potentially leaked benchmarks [2]. Headline numbers from simpler benchmarks therefore do not transfer directly to realistic software.

These findings motivate a shift in emphasis. Rather than asking only whether an LLM can generate a test that compiles and runs, the more consequential question is whether an agentic system can produce a chain of evidence that remains traceable to the original specification and is validated by deterministic execution. A passing test is not, by itself, evidence that the product behaves correctly: it may pass because its oracle is weak, because it targets the wrong interface, or because a repair step has adjusted it until it turns green.

This paper makes three contributions. First, it articulates the governed transition from probabilistic test generation to execution-grounded quality evidence as a research problem with explicit boundaries (Section 3). Second, it specifies QUALTAN, an architecture that separates reasoning, governance, and deterministic execution (Section 4). Third, it pre-specifies a complete experimental protocol, including research questions, hypotheses, metrics, ablations, statistical tests, falsification criteria, and threats to validity (Sections 5 to 10), so that the resulting study can be judged on its design before its results are known.

2 Related Work

LLM-based test generation. Early empirical work evaluated whether LLMs can produce useful unit tests directly from source code. Schäfer et al. conducted an empirical evaluation of LLMs for automated unit test generation, reporting on the quality of the tests produced by the model under controlled conditions [3]. CodaMosa combined LLM-generated tests with search-based test generation to overcome coverage plateaus that either technique reached alone [4]. At Meta, TestGen-LLM used LLMs to improve existing test classes and applied automated filters so that only generated tests that build, pass, and add value were retained [5]. These systems show that LLMs can contribute useful tests, but they generally treat execution as a filter on output rather than as a structured stage in a governed pipeline with explicit artifacts and evidence records.

Surveys and roadmaps. Surveys organize the rapidly growing literature on LLMs for software testing and identify open problems in test oracles, evaluation, and reliability [1], [6]. The present protocol builds on these roadmaps by turning several of the open problems into testable hypotheses about architecture.

Evaluation validity. The ULT benchmark demonstrates that contamination and structural simplicity can inflate reported

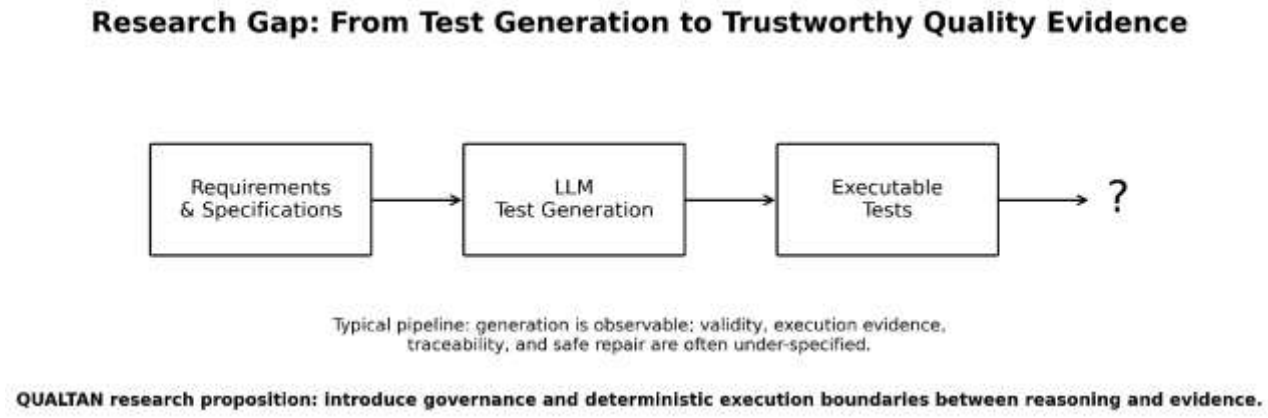
performance [2]. This motivates the protocol's emphasis on realistic applications, seeded defects, controlled contamination checks, and repeated runs.

Adequacy criteria. Mutation analysis is a standard way to assess whether a test suite detects faults, through the proportion of injected faults (mutants) that a suite kills [7]. We use mutation score as an effectiveness measure and treat equivalent mutants explicitly, because misclassifying them distorts the metric.

Statistical practice. Randomized and nondeterministic algorithms in software engineering require statistical methods that account for run-to-run variation and paired designs [8]. We use rank-based paired tests, the Vargha–Delaney A12 effect size [9], and Holm's correction for multiple comparisons [10].

The present work differs from this literature in its unit of analysis. Rather than evaluating only the tests an LLM produces, it evaluates whether a governed pipeline that couples specification, policy checks, deterministic execution, and repair yields tests whose validity, traceability, and maintenance behavior can be audited, and it isolates each mechanism through ablation.

3 Problem Statement and Research Gap



Typical pipelines make test generation observable, but validity, execution evidence, traceability, and safe repair are often under-specified.

Figure 1. The research gap: LLM test generation is only one stage on the path to trustworthy quality evidence.

Current AI-assisted testing systems often place the LLM at the center of test synthesis. This is productive, but it introduces recognizable failure modes:

- Hallucinated product behavior, unavailable interfaces, or incorrect selectors, endpoints, contracts, or test data.
- Syntactically valid tests that are semantically weak.
- Assertions that do not verify the intended behavior.
- Duplicate or low-value scenarios.
- Tests that pass because the oracle is weak rather than because the product is correct.
- Failure diagnoses that conflate product defects with test, data, dependency, or environment faults.
- Self-healing behavior that modifies a test until it becomes green, potentially weakening its original intent.

The research gap is therefore not test generation alone. It is the governed transition from probabilistic reasoning to deterministic, auditable, execution-grounded quality evidence. We formulate this gap as the claim tested by the study:

> **Core claim.** A governed agentic testing architecture provides a more reliable bridge from LLM reasoning to executable software-quality evidence than LLM-only test generation, and the contribution of each governance mechanism can be isolated experimentally.

3.1 Positioning QUALTAN in the AI-Driven Development Lifecycle

The AI-Driven Development Lifecycle (AI-DLC) is an open-source, methodology-first framework that AWS introduced in 2025 for structuring AI-assisted software delivery. It organizes work into three phases, Inception, Construction, and Operations. Its workspace detection step determines whether a project is greenfield or brownfield: brownfield projects proceed to reverse engineering of the existing codebase, while greenfield projects proceed to requirements analysis. Human approval gates separate AI-generated artifacts from the decisions that humans must make, and the lifecycle records its artifacts for audit [11].

AI-DLC specifies where humans decide and how AI-generated requirements, design, and code are organized. It does not, however, specify how the test evidence produced during Construction is validated, traced, or maintained as the system changes. A lifecycle that generates code quickly, and that generates tests with the same model, can therefore accumulate tests whose validity is unknown. QUALTAN is positioned to close this gap within the lifecycle rather than beside it. Table 1 summarizes the correspondence.

Table 1. Where QUALTAN operates within the AI-DLC phases.

AI-DLC phase	What the methodology provides	Testing gap it leaves	QUALTAN contribution	Measured by
Inception	Requirements or reverse-engineering artifacts; approval gates	No defined path from approved requirements to executable, traceable tests	Specification agent and quality intelligence derive Test Intent; requirement links are recorded from the outset	Requirement coverage (RQ1); requirement-to-evidence coverage (RQ6)
Construction	Design and code generation with human review	Generated tests can pass for weak reasons; no record of execution-grounded validity	Policy gate before execution; execution as evidence; evidence records	Executable-test rate (RQ2); defect detection (RQ3)
Operations	Ongoing operation and change of the system	Application changes break tests; repairs can weaken intent; suite volume grows	Risk-aware selection; intent-preserving repair accepted only after re-execution	Repair success and false-positive rate (RQ4); executed-test volume (RQ5)

3.2 Greenfield Projects

In a greenfield project, AI-DLC proceeds directly to requirements analysis because no existing code or tests constrain the system. This is the setting in which requirements, tests, and code are created together, and it carries a specific risk: if tests are derived from the generated implementation, they confirm what the code does rather than what it should do. Such tests can pass consistently and still provide little evidence of correctness.

QUALTAN addresses this by making specification precede generation. Test intent is derived from approved requirement artifacts, not from the code, and the policy gate checks that each test traces to a requirement before execution. Because the canonical artifacts are created at project start, traceability is established from the first iteration rather than reconstructed later. The benefit that matters most in this setting is the one the protocol measures directly: whether requirement-to-evidence coverage and defect detection are higher when tests are generated from governed specifications than when they are generated from unconstrained prompts (H1, H3, H6).

A limit must be stated. Tests derived from the same specification as the code share any errors in that specification. QUALTAN mitigates this through independent oracle review, human approval gates, and seeded-defect evaluation, but it cannot detect a requirement that is itself wrong. The protocol therefore treats specification quality as a stated assumption and reports it as a threat to validity (Section 10).

3.3 Brownfield Projects

In a brownfield project, AI-DLC begins with reverse engineering to build an understanding of the existing system. This phase is valuable, but existing test suites in such projects are often incomplete, outdated, or undocumented, and their assertions may encode current behavior, including defects. The characteristic risk here is the characterization trap: generated tests that faithfully pin whatever the system currently does, so that they pass and lend false confidence.

QUALTAN responds to this risk in three ways. First, reverse-engineered artifacts and the existing test inventory feed the quality-intelligence layer, which uses change information and historical failures to prioritize what to test. Second, risk-aware selection can reduce the execution volume of large existing suites while the protocol checks that defect detection is preserved (H5). Third, and most important, repair must distinguish an intended change from a regression. When application code changes, existing tests fail, and an unconstrained repair agent can make them pass by weakening the assertion. QUALTAN's repair is accepted only if policy validation and re-execution show that the original intent is still satisfied, and the assertion checker rejects repairs that weaken assertions (H4).

The oracle for a brownfield test is taken from the specification where one exists. Where none exists, the oracle review protocol classifies a test that only pins current behavior as a *weak* oracle, so that characterization tests do not count as validated evidence. The benchmark's application-change scenarios are drawn from this setting, so the maintenance hypotheses are tested where they matter most.

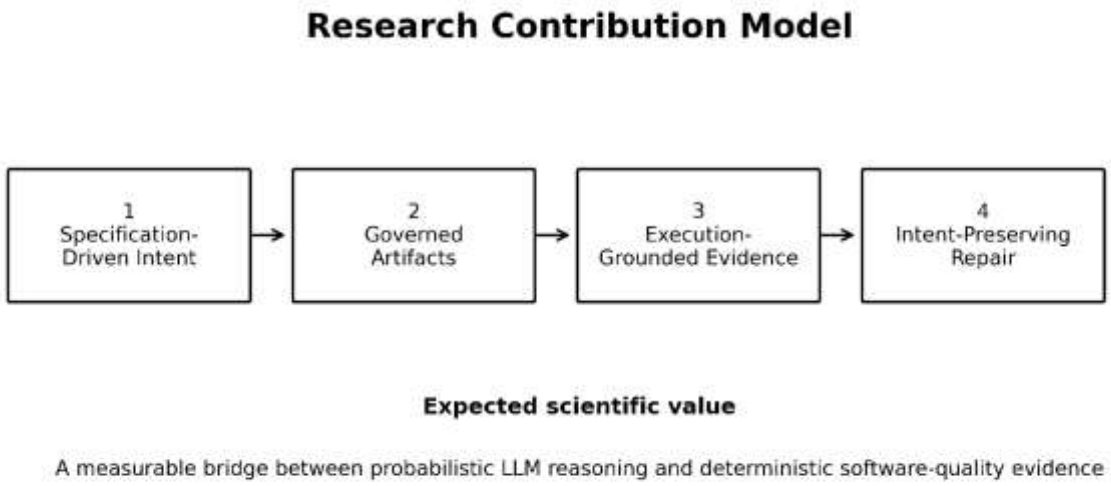
3.4 Why the Positioning Matters

The practical significance of QUALTAN within AI-DLC follows from the two settings. In greenfield projects, the architecture determines whether the test evidence is traceable to requirements from the start. In brownfield projects, it determines whether tests remain trustworthy as the system changes. In both, the architecture governs the step that AI-DLC leaves open: turning generated tests into evidence that a reviewer can audit. Whether this improves outcomes is an empirical question, and the protocol is designed so that the answer could be negative. The importance claimed here is therefore a claim about where the protocol's hypotheses apply, not a result.

4 The QUALTAN Architecture

QUALTAN rests on five architectural principles:

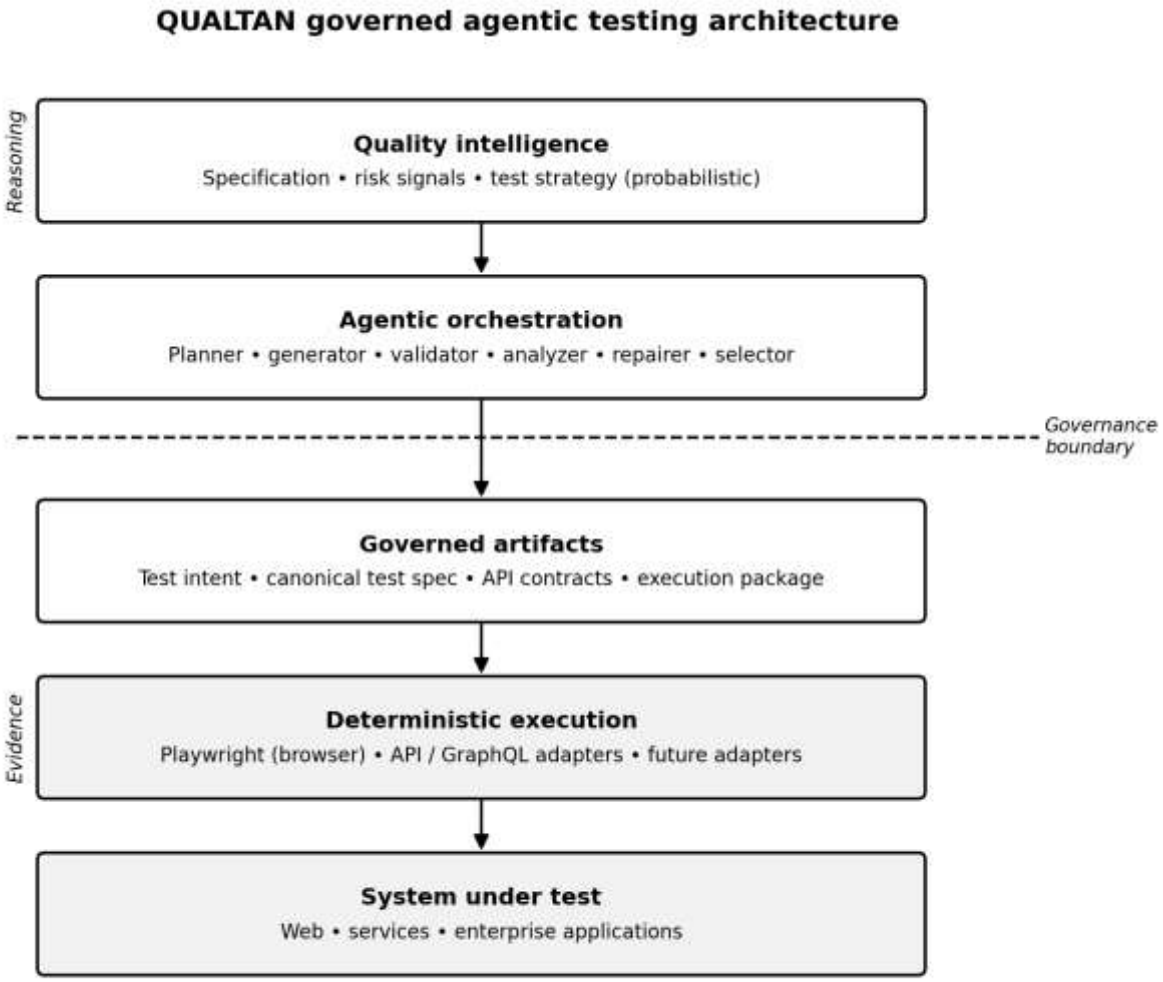
1. **Specification before generation.** Requirements and contracts are converted into structured test intent before any test code is produced.
2. **Generation before execution.** Generated artifacts pass schema, semantic, dependency, and policy checks before they are executed.
3. **Execution as evidence.** A generated test is not treated as validated until it executes against the intended system under test.
4. **Repair preserves intent.** An autonomous repair is accepted only when the original requirements and assertions remain satisfied on re-execution.
5. **Traceability by construction.** The chain from requirement to test intent, canonical artifact, execution record, evidence, and repair history is recorded as the pipeline runs.



The four mechanisms connect probabilistic reasoning to deterministic, auditable evidence.

Figure 2. Research contribution model: four architectural mechanisms connect probabilistic reasoning to deterministic evidence.

The architecture separates intelligence from execution. Agents reason about specifications, risks, scenarios, and failures, while deterministic adapters perform actions against the system under test (Figure 3). The boundary is intended to reduce the authority of unverified model output while retaining the productivity benefits of agentic reasoning.



QUALTAN reference architecture. The governance boundary separates probabilistic reasoning above from deterministic execution below; artifacts crossing it must pass policy checks.

Figure 3. QUALTAN reference architecture. Governed artifacts are the only interface between reasoning and execution.

Quality intelligence layer. This layer interprets requirements, acceptance criteria, business rules, change information, historical failures, and risk signals. Its output is a structured test strategy, not executable code.

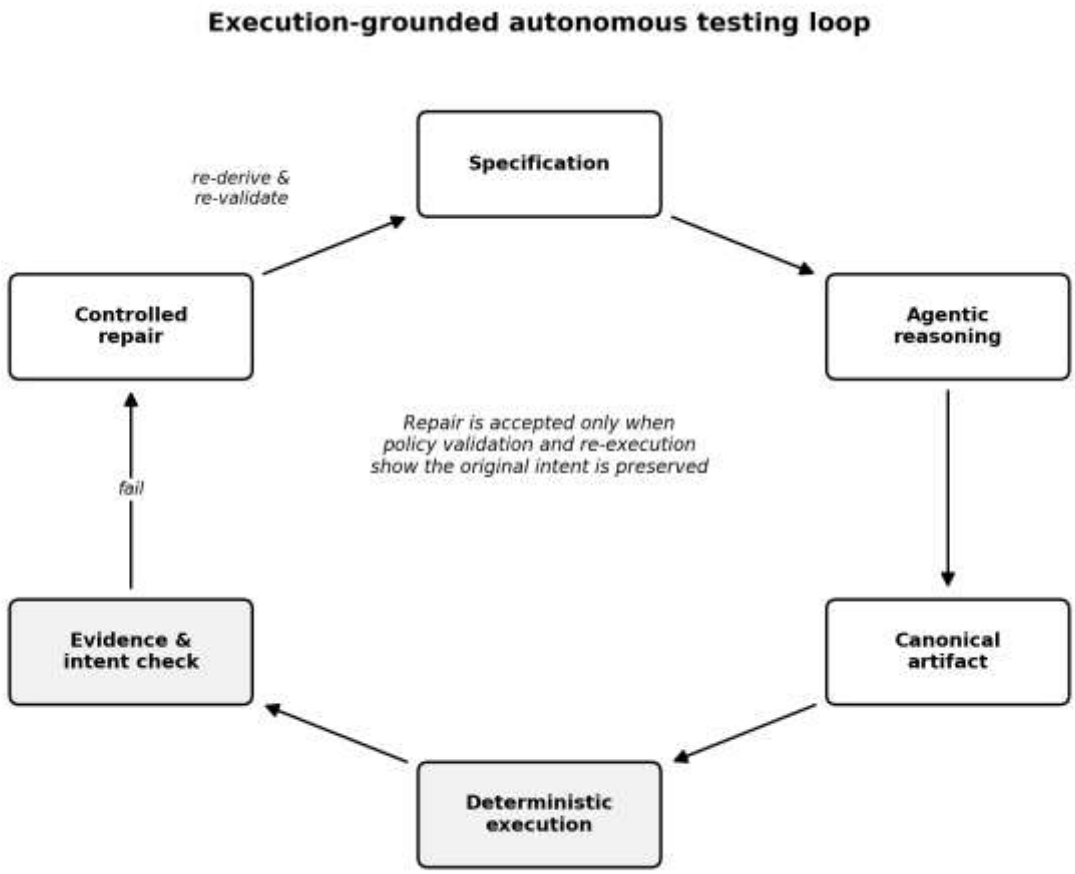
Agentic orchestration layer. The proposed agent set comprises a specification agent, planner, generator, validator, executor coordinator, failure analyzer, repair agent, and test selector. Specialized agents with explicit responsibilities limit the tendency of a single general-purpose agent to take on incompatible roles.

Governed artifact layer. Canonical representations, namely Test Intent, Test Specification, API Contract, Test Data, Execution Package, and Evidence, provide stable interfaces between agents and execution adapters and a persistent basis for traceability. Appendix A gives an example of a canonical test artifact.

Deterministic execution layer. The initial implementation uses Playwright for browser workflows and deterministic API and GraphQL adapters. Additional adapters can be added without changing the research model.

4.1 The Execution-Grounded Testing Loop

The defining behavior of QUALTAN is a closed loop from specification to evidence and, when necessary, controlled repair (Figure 4). A generated script is not treated as successful merely because it compiles or runs. Evidence is accepted only after the policy gate and execution stage have run, and a repair is accepted only after policy validation and re-execution show that the original intent is preserved.



Execution-grounded testing loop. A failed test is routed to controlled repair, which re-enters at the specification and must pass policy validation and re-execution before acceptance.

Figure 4. Execution-grounded testing loop. Passing executions produce evidence records; failing executions are routed to controlled repair, which is accepted only when re-execution preserves the original intent.

5 Research Questions and Hypotheses

Table 2 links each research question to its hypothesis, primary metric, and primary comparison. The hypotheses are stated directionally so that each can be rejected.

Table 2. Research questions, hypotheses, primary metrics, and primary comparisons. Condition labels are defined in Table 3.

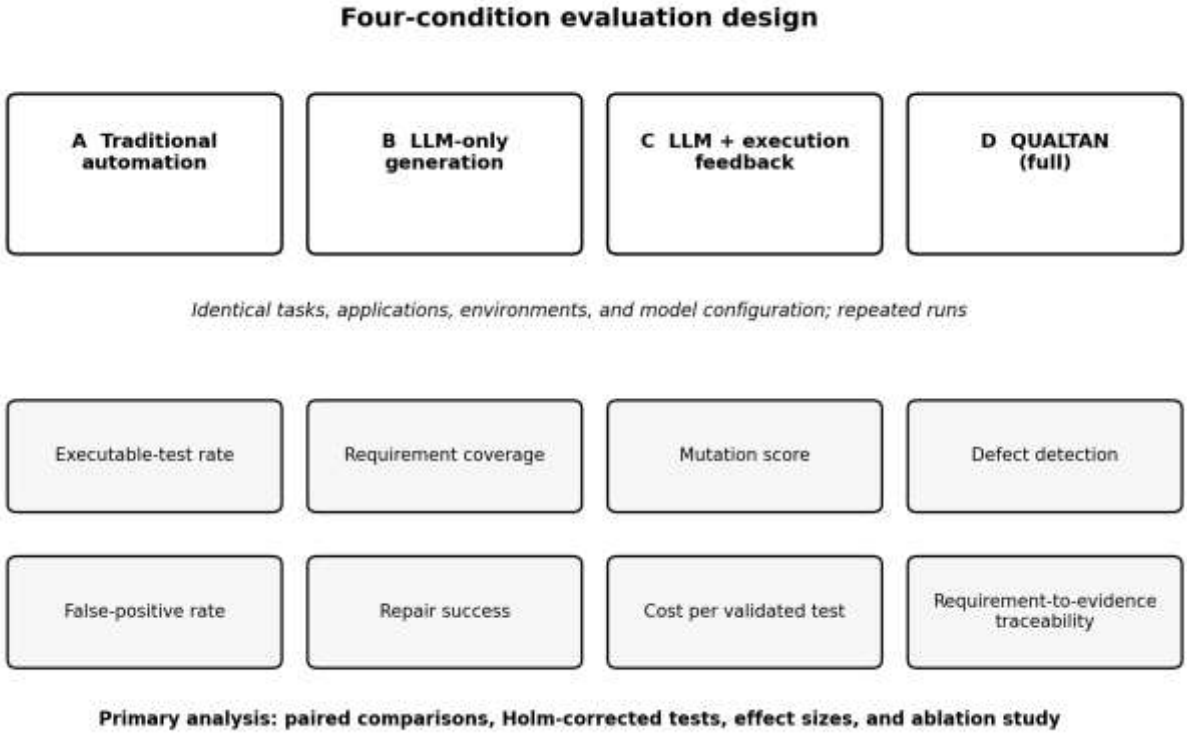
RQ	Research question	Hypothesis	Primary metric	Primary comparison
RQ1	Does specification-driven generation improve the functional relevance of LLM-generated tests compared with unconstrained prompting?	H1: Specification-driven generation increases requirement and scenario coverage.	Requirement coverage	D vs. B
RQ2	Does execution-grounded validation reduce invalid or non-executable generated tests?	H2: Execution-grounded validation increases the executable-test rate.	Executable-test rate	C vs. B (secondary: D vs. B)
RQ3	Does governed agentic orchestration improve defect detection compared with LLM-only generation?	H3: Governed agentic testing improves seeded-defect detection over LLM-only generation.	Defect detection rate	D vs. B
RQ4	Can intent-preserving autonomous repair reduce maintenance effort without weakening assertions or increasing false positives?	H4: Intent-preserving repair reduces maintenance effort while preserving mutation score and not increasing false-positive rate.	Repair success rate; maintenance effort; mutation score	D vs. D without repair

RQ	Research question	Hypothesis	Primary metric	Primary comparison
RQ5	Can risk-aware test selection reduce execution volume while preserving defect detection?	H5: Risk-aware selection reduces executed-test volume with limited loss in defect detection.	Executed-test count; defect detection rate	D vs. D without selection
RQ6	Does canonical artifact governance improve reproducibility and requirement-to-test traceability?	H6: Canonical artifacts increase reproducibility and traceability.	Reproduction rate; requirement-to-evidence coverage	D vs. D without artifacts

6 Experimental Design

6.1 Conditions

The independent variable is the level of architectural governance and execution feedback. All conditions use identical tasks, applications, environments, and a single pinned model configuration (Figure 5).



Four-condition evaluation design and primary metrics. All conditions share tasks, environments, and model configuration; metrics are analyzed with paired comparisons.

Figure 5. Four-condition evaluation design and proposed metrics.

Table 3. Experimental conditions.

Condition	Description	Purpose	Expected role
A: Traditional automation	Human-authored tests taken from each benchmark application's existing suite, or written to the same specification if none exists	Reference for conventional engineering practice	Reference
B: LLM-only generation	The model receives the application source and a request for tests; no structured specification, no execution feedback	Measures unconstrained generation	Primary weak baseline
C: LLM + execution feedback	As B, plus execution of each generated test and iterative correction from execution errors	Measures the value of execution grounding alone	Intermediate baseline

Condition	Description	Purpose	Expected role
D: QUALTAN	Specification, governance, execution, repair, and selection as specified in Section 4	Evaluates the complete architecture	Proposed system

Condition A uses existing human-authored tests from open-source projects, so the study involves no new human participants. For fairness, all LLM conditions (B, C, and D) receive the same per-task token budget ceiling, so that any advantage of D cannot be attributed to additional model calls alone.

6.2 Benchmark Composition

The benchmark combines four sources:

- Real-world open-source applications with executable test suites, selected for licensing compatibility and buildability.
- Synthetic enterprise workflows, used where controlled ground truth is required.
- REST and GraphQL services with published schemas.
- Browser workflows running in stable, containerized test environments.

Tasks are labeled by project context. Greenfield tasks begin from requirements with no existing tests or code. Brownfield tasks begin from an existing codebase and its test suite, and include application-change scenarios. Both contexts are represented in the benchmark so that the hypotheses can be assessed in each.

Each application carries seeded defects spanning functional, validation, boundary, authorization, and integration failures, and application-change scenarios that exercise maintenance and repair. Contamination is addressed by reporting, for each task, whether its source and tests appear in public repositories predating the model's training cutoff, and by including synthetic and recently created applications in the analysis. The number of tasks is determined by a power analysis performed before data collection, targeting 80% power at $\alpha = 0.05$ for a medium paired effect ($d_z = 0.5$).

6.3 Run Protocol

Each task is executed under each condition for at least five independent runs, because model nondeterminism is expected to be material. We pin the model identifier and version and record the inference configuration (temperature, maximum output tokens, and any seed parameters supported by the provider). We record prompts, tool configurations, intermediate artifacts, and execution traces for every run. Repair experiments reuse the same application-change scenarios across conditions. Test execution occurs in clean environments reset between runs.

Oracle review protocol. Two reviewers independently classify the assertions of each validated test against the acceptance criteria of its requirement as *adequate*, *weak* (checks an observable but not the specified behavior), or *incorrect*. Disagreements are resolved by discussion with a third reviewer. Inter-rater agreement is reported as Cohen's kappa. Reviewers are blind to the experimental condition that produced each test.

7 Evaluation Metrics

Table 4 defines each metric operationally. A *validated test* is one that executes without infrastructure errors against the unmodified system under test and whose assertions are checked against the specified expected behavior by the oracle review protocol described in Section 6.3.

Table 4. Evaluation metrics and operational definitions.

Dimension	Metric	Operational definition
Generation quality	Executable-test rate	Generated tests that execute to completion (pass or fail, without infrastructure error) before any repair, divided by generated tests
Coverage	Requirement coverage	Benchmark requirements linked to at least one validated test whose assertions reference that requirement's acceptance criteria, divided by benchmark requirements
Coverage	Branch/statement coverage	Branch and statement coverage of the application under the executed suite, measured with the standard tool for the language
Effectiveness	Mutation score	Killed mutants divided by non-equivalent mutants. Equivalent mutants are identified by automated detection plus manual review of a random sample, and the review agreement is reported
Effectiveness	Defect detection rate	Seeded defects for which at least one test in the suite fails on the defective version and passes on the corrected version, divided by seeded defects
Reliability	False-positive rate	Test failures reported as product defects whose root cause is not a product defect (per ground-truth labels), divided by all failures reported as product defects
Maintenance	Repair success rate	Repaired tests that pass on the updated application and whose original assertions are retained, equivalent, or strengthened, as verified by an assertion checker, divided by attempted repairs
Maintenance	Maintenance effort	Agent iterations, edited assertion lines, and human intervention events per accepted repair

Dimension	Metric	Operational definition
Economics	Cost per validated test	Model, infrastructure, and execution cost in US dollars divided by validated tests
Traceability	Requirement-to-evidence coverage	Benchmark requirements with a complete recorded chain (requirement, test intent, canonical artifact, execution record, evidence record), verified by automated link checks
Reproducibility	Reproduction rate	Independent re-executions of a pipeline from its recorded artifacts that reproduce the same verdicts, divided by re-executions attempted

8 Ablation Study

An agentic architecture can appear effective merely because it contains more components. QUALTAN is therefore evaluated with each mechanism removed in turn:

- Full QUALTAN (D).
- D without specification-driven intent.
- D without governance policy checks.
- D without execution feedback.
- D without risk-based selection.
- D without autonomous repair.
- D without canonical artifacts.

The objective is to estimate the marginal contribution of each mechanism rather than to attribute every observed improvement to the complete system. Each ablation uses the same tasks, runs, and budget ceilings as the main experiment.

9 Statistical Analysis Plan

Unit of analysis. For each task and condition, we average the outcome over runs, yielding one paired observation per task. Binary per-test outcomes are additionally analyzed with a mixed-effects logistic regression that includes a random intercept for task and a fixed effect for condition, as a sensitivity analysis.

Primary tests. Paired comparisons use the Wilcoxon signed-rank test, which does not assume normality. Effect sizes are reported as the Vargha–Delaney A12 statistic [9], interpreted as small (≈0.56), medium (≈0.64), or large (≈0.71) by conventional thresholds. Confidence intervals are obtained by bootstrap resampling over tasks with 10,000 resamples, which respects the task-level clustering of the data.

Multiple comparisons. The primary family comprises the six hypotheses. Within each hypothesis, p-values are adjusted with Holm's procedure [10].

Nondeterminism. For each task and condition, the coefficient of variation of the outcome across runs is reported. We also report the intraclass correlation of outcomes within tasks, so readers can judge how much of the observed variation reflects differences between tasks and how much reflects run-to-run stochasticity.

Falsification criteria. Each hypothesis is supported only if all of the following hold: the Holm-adjusted p-value is below 0.05; the effect is in the predicted direction; A12 is at least 0.64 (medium or larger); and the 95% bootstrap confidence interval for the paired difference excludes zero.

For H5, which is a non-inferiority claim, support requires that the selected suite executes at most 70% of the full-suite tests and that the lower bound of the 95% confidence interval for the retained defect-detection ratio is at least 0.95. These thresholds are fixed in this protocol before data collection. Any change to them in the course of the study will be reported as a deviation with justification.

Aggregate and per-application results. Results are reported in aggregate and per application, so that a small number of large applications does not dominate the conclusions. Results are additionally reported separately for greenfield and brownfield tasks. This stratification is pre-specified and descriptive: the hypotheses are tested on the pooled primary analysis, and the stratified results show whether each effect holds in both project contexts.

Primary outcomes. The primary outcomes, fixed before experimentation, are executable-test rate, mutation score, seeded-defect detection rate, repair success rate, and requirement-to-evidence coverage.

10 Threats to Validity

Internal validity. Improvements may arise from factors other than the intended mechanisms, such as differences in token budget, prompt length, or the number of tool calls. We mitigate this with budget ceilings applied equally to all LLM conditions and with the ablation study, which holds all but one mechanism constant. Differences in environment setup between conditions can also confound results, which we address by resetting environments between runs.

Construct validity. Metrics may not capture what they are intended to measure. A high mutation score can reflect mutants that are easy to kill rather than thorough oracles, and a passing test can have a weak oracle. We mitigate this with an oracle review protocol, equivalent-mutant review, and the separate assertion checker used for repair success. Requirement coverage depends on how requirements are linked to tests, and the linking rules are fixed in advance.

External validity. Results from selected applications may not generalize to all software domains, languages, or testing

practices. The mix of open-source, synthetic, REST, GraphQL, and browser tasks is intended to broaden coverage, but generalization beyond the benchmark remains an open question.

Conclusion validity. Small numbers of tasks or high run-to-run variance can produce unreliable conclusions. The power analysis, minimum run count, paired design, bootstrap intervals, and multiple-comparison correction are intended to address this.

Model dependence. Different LLMs and versions may produce different results. We pin the model version and report results for one primary configuration. Any second model is treated as a replication, not as a pooled condition.

Benchmark contamination. Public benchmark material may overlap with model training data. The contamination reporting described in Section 6.2 and the inclusion of synthetic and recent applications are intended to bound this risk.

Environment dependence. Execution outcomes can vary with environment, data, and dependencies. Containerized environments with pinned dependencies reduce this variation.

Agent nondeterminism. Repeated runs may follow different reasoning paths. The five-run minimum and the variance decomposition in Section 9 quantify this effect.

Repair bias. A repair mechanism may appear effective if it is permitted to weaken assertions. The assertion checker and the requirement that repaired tests pass on the updated application address this directly, and repair success is reported separately from mutation score.

11 Expected Contribution and Interpretation of Outcomes

The intended contribution is a measurable architectural pattern for converting probabilistic LLM reasoning into governed software-quality evidence, together with an experimental protocol that can be reused and extended. The protocol does not claim that QUALTAN will outperform all existing approaches. It defines an experiment capable of supporting, refining, or falsifying the proposed claims.

If the hypotheses are supported, the resulting contribution is a framework for execution-grounded agentic testing with explicit governance boundaries, together with evidence for which mechanisms drive the improvement. If some hypotheses are rejected, the ablation and benchmark results remain informative: they would show which mechanisms do not materially affect AI-assisted testing quality, which is useful for practitioners deciding where to invest engineering effort. Reporting negative results under the same protocol is an explicit goal of the study.

12 Reproducibility and Data Availability

Reproducibility is treated as a first-class requirement. The IEEE author guidance encourages detailed methodology and the sharing of data and code so that independent researchers can verify and build on findings [12]. Accordingly, the study will:

- Version-lock model identifiers and inference configuration.
- Publish benchmark task definitions, seeded-defect specifications, and environment setup.
- Publish canonical artifact schemas and the policy rules used by the governance layer.
- Publish evaluation scripts, the oracle review protocol, and the statistical analysis code.
- Publish raw and aggregated experiment results where licensing permits.
- Archive datasets and code with a persistent identifier such as a DOI at the time of publication.
- Record prompts, tool configurations, and execution traces needed for replication.

Data and code availability statements will be finalized at submission, and the archived repository identifier will be inserted at that time.

13 Ethical Considerations

The study involves no new human participants and no personal data. Condition A uses existing test suites from open-source projects under their licenses, and all benchmark applications will be selected for licensing compatibility. Synthetic enterprise workflows contain no real customer data. Costs of LLM inference are reported to support transparent comparison and to avoid overstating the practicality of any configuration.

14 Conclusion

This paper presents a protocol for a controlled study of governed, execution-grounded agentic testing. Its central proposition is that LLM reasoning becomes more useful for software quality when it is constrained by specifications, canonical artifacts, deterministic execution, evidence collection, and intent-preserving repair. The protocol pre-specifies the research questions, hypotheses, conditions, metrics, statistical tests, and falsification criteria under which that proposition will be judged. The results of the study will determine whether the proposition holds.

Declarations

Funding: [To be completed by the author.]

Competing interests: [To be completed by the author.]

Data and code availability: Described in Section 12. The archive identifier will be provided at submission.

Author contributions: Mohan Gundala conceived the architecture, designed the protocol, and wrote the manuscript

Appendix A: Example Canonical Test Artifact

The following example illustrates a canonical test artifact for a login requirement. The artifact is the only form in which test intent passes from the reasoning layer to the execution layer.

```
{
  "test_id": "AUTH-001",
  "requirement_id": "REQ-001",
  "risk": "HIGH",
  "type": "web",
  "preconditions": [],
  "steps": [
    {"action": "navigate", "target": "/login"},
    {"action": "fill", "target": "username", "value":
"${USERNAME}"},
    {"action": "fill", "target": "password", "value":
"${PASSWORD}"},
    {"action": "click", "target": "login"}
  ],
  "assertions": [
    {"type": "url", "expected": "/dashboard"}
  ]
}
```

References

[1] C. Augusto, A. Bertolino, G. De Angelis, F. Lonetti, and J. Morán, “Large Language Models for Software Testing: A Research Roadmap,” arXiv:2509.25043, 2025.

[2] 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,” arXiv:2508.00408, 2025.

[3] 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*, 2024.

[4] C. Lemieux, J. P. Inala, S. K. Lahiri, and S. Sen, “CodaMosa: Escaping Coverage Plateaus in Test Generation with Pre-trained Large Language Models,” in *Proc. 45th IEEE/ACM International Conference on Software Engineering (ICSE)*, 2023.

[5] N. Alshahwan, J. Chheda, A. Finegenova, B. Gokkaya, M. Harman, I. Harper, A. Marginean, S. Sengupta, and E. Wang, “Automated Unit Test Improvement using Large Language Models at Meta,” in *Proc. 32nd ACM Symposium on the Foundations of Software Engineering (FSE) Industry Track*, 2024.

[6] 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.

[7] Y. Jia and M. Harman, “An Analysis and Survey of the Development of Mutation Testing,” *IEEE Transactions on Software Engineering*, vol. 37, no. 5, 2011.

[8] A. Arcuri and L. Briand, “A Practical Guide for Using Statistical Tests to Assess Randomized Algorithms in Software Engineering,” in *Proc. 33rd International Conference on Software Engineering (ICSE)*, 2011.

[9] A. Vargha and H. D. Delaney, “A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong,” *Journal of Educational and Behavioral Statistics*, vol. 25, no. 2, 2000.

[10] S. Holm, “A Simple Sequentially Rejective Multiple Test Procedure,” *Scandinavian Journal of Statistics*, vol. 6, no. 2, 1979.

[11] Amazon Web Services, “Building with AI-DLC using Amazon Q Developer,” AWS DevOps & Developer Productivity Blog. [Online]. Available: https://aws.amazon.com/blogs/devops/building-with-ai-dlc-using-amazon-q-developer/. [Accessed: September 2026].

[12] IEEE Author Center Journals, “Research Reproducibility.” IEEE. Accessed September 2026.