PriMera Scientific Engineering
Volume 8 Issue 6 June 2026
ISSN: 2834-2550

PriMera Scientific Publications

The Rise of Agentic Testing: Toward Autonomous and Self-Improving Software Quality Assurance

Type: Editorial Note
Received: February 18, 2026
Published: May 31, 2026

Citation:
Mohammad Baqar. "The Rise of Agentic Testing: Toward Autonomous and Self-Improving Software Quality Assurance". PriMera Scientific Engineering 8.6 (2026): 80-82.

Mohammad Baqar*

Cisco Systems Inc, USA

***Corresponding Author:** Mohammad Baqar, Cisco Systems Inc, USA.

As software systems grow increasingly complex and release cycles accelerate under continuous integration and deployment models, traditional approaches to software testing are struggling to keep pace. Manual test authoring remains labor-intensive and slow, while even AI-assisted test generation tools often operate in static, one-shot modes that fail to account for execution outcomes. In this evolving landscape, the paper "*The Rise of Agentic Testing: Multi-Agent Systems for Robust Software Quality Assurance*" presents a compelling shift in paradigm. Rather than viewing test generation as a standalone task, the authors propose an autonomous, multi-agent architecture that treats testing as an iterative, feedback-driven process. This concept "termed agentic testing" signals an important step toward self-correcting and adaptive quality assurance systems.

The central critique underlying the paper is that current large language model (LLM)-based test generation systems lack execution awareness. While modern generative models can produce syntactically correct test cases from code or specifications, they often fail when those tests are executed in real environments. Missing dependencies, incorrect assumptions about program state, or misunderstood APIs frequently render generated tests invalid. More importantly, most systems lack a mechanism for refinement. They generate tests once, and if those tests fail, human engineers must intervene. The authors identify this absence of closed-loop feedback as the key limitation of static AI-driven testing methodologies.

To address this gap, the proposed Agentic Testing Architecture (ATA) introduces a cooperative, multi-agent framework composed of three distinct roles: a Test Generation Agent, an Execution and Analysis Agent, and a Review and Optimization Agent. Together, these agents form a cyclical pipeline in which tests are generated, executed, analyzed, and iteratively refined until performance thresholds such as improved code coverage and reduced failure rates are achieved. This architecture mirrors the workflow of experienced human testers, who rarely produce perfect tests on the first attempt but instead refine them based on observed outcomes. By embedding this iterative reasoning directly into an automated system, the framework transforms testing from a static artifact creation process into a dynamic learning loop.

The Test Generation Agent leverages large language models to produce initial test cases from source code, requirements, or annotations. These tests are then passed to the Execution and Analysis Agent, which runs them in a sandboxed environment and gathers metrics such as runtime errors, assertion failures, and coverage statistics. The Review and Optimization Agent interprets this execution feed-

back and revises or regenerates improved tests accordingly. This loop continues until convergence criteria are satisfied. The key innovation lies not in any single agent, but in the orchestration of agents that collaborate through structured feedback. It is this iterative refinement mechanism that differentiates agentic testing from conventional AI-assisted test generation.

The empirical evaluation presented in the paper strengthens the argument for this approach. Across multiple open-source and enterprise-scale applications, the authors report substantial gains in testing performance. The agentic framework achieved approximately a 30 percent increase in mean code coverage compared to static LLM-based generation, while simultaneously reducing invalid test cases by roughly 60 percent. These improvements are particularly significant because high coverage and low failure rates are often competing objectives in automated test generation. The results suggest that iterative refinement enables the system to both expand coverage and improve correctness. Additionally, the reduction in manual debugging and intervention highlights a practical benefit: agentic testing does not merely enhance metrics, but may meaningfully reduce engineering effort.

Beyond the reported results, the conceptual contribution of the work lies in its reframing of testing as an autonomous, self-improving ecosystem. The broader field of AI-assisted software engineering has seen rapid advancements in code synthesis, documentation generation, and static analysis. However, many of these systems remain transactional in nature - they produce outputs without adapting to downstream consequences. The proposed architecture aligns more closely with reinforcement learning principles, where actions are evaluated and adjusted based on feedback signals. By embedding execution outcomes directly into the refinement loop, the framework moves testing closer to a learning-driven process rather than a generative one.

This shift is particularly timely given the increasing complexity of modern software systems. Microservices architectures, distributed deployments, and multi-language stacks introduce layers of interaction that static test generators struggle to model accurately. An agentic approach, capable of diagnosing its own failures and adjusting accordingly, offers a more resilient pathway forward. The modular structure of the architecture further enhances its appeal. Because generation, execution, and review are separated into distinct agents, improvements in one component - such as advances in LLM performance or better coverage analysis tools can be integrated without redesigning the entire system.

Nevertheless, the proposal raises important considerations for future exploration. One challenge lies in managing the inherent non-determinism of large language models. Iterative refinement processes that rely on stochastic outputs may introduce variability across runs, potentially complicating reproducibility and debugging. Scalability also warrants careful examination. While sandboxed execution and iterative feedback loops are manageable for moderate-sized codebases, the computational overhead could grow significantly for large-scale enterprise systems. Efficient orchestration strategies and context management mechanisms will be critical to sustaining performance gains at scale.

Another open question concerns generalizability. The evaluation demonstrates promising results across selected applications, but broader validation across diverse programming languages, testing frameworks, and deployment contexts would further strengthen confidence in the approach. Additionally, as multi-agent AI systems become more integrated into development pipelines, governance and observability mechanisms will be necessary to ensure transparency and maintain developer trust.

Despite these challenges, the paper's contribution is both conceptually innovative and practically grounded. By introducing a structured, feedback-driven architecture, the authors articulate a vision in which automated testing systems do more than generate artifacts - they reason, adapt, and improve. In doing so, agentic testing represents a meaningful evolution in the field of software quality assurance.

In conclusion, "*The Rise of Agentic Testing*" presents a compelling case for rethinking automated testing through the lens of multi-agent collaboration and iterative refinement. The framework moves beyond static generation and toward autonomous quality assurance systems that learn from execution outcomes. As AI tools continue to reshape software engineering workflows, agentic testing offers a blueprint for more intelligent, adaptive, and resilient QA ecosystems. If further validated and scaled, this approach may help

usher in an era where software testing becomes not just automated, but genuinely self-improving - bringing us closer to fully autonomous development pipelines.