(Over)Reliance on Test Agents in AI-Assisted Software Testing

Eduard Paul Enoiu[0000−0003−2416−4205]

Department of Computer Science and Engineering
Mälardalen University, Västerås, Sweden
eduard.paul.enoiu@mdu.se

Abstract. AI-based test agents promise to accelerate software testing by shortening feedback loops in continuous development and improving scalability and maintainability. To realize these benefits, engineers must still be able to assess if agent outputs are useful, valid, and reliable, rather than treating them as credible because they come from a capable system. This paper argues that overreliance on AI in testing is both an agency problem, in which engineers may cede cognitive control over test design decisions, and an assurance problem, in which testing artifacts may be accepted as evidence without sufficient scrutiny. We develop this argument through three theoretical lenses: software testing as cognitive problem-solving, test agents as adaptively autonomous entities, and test design argumentation as a means of making generated tests reviewable. We propose a framework for collecting data on overreliance in test agent workflows and identify specific modes of overdependence. The goal is to support accelerated testing without weakening judgment or the assurance value of testing evidence.

Keywords: test agents · cognitive problem solving · overreliance.

1 Introduction

Modern AI-assisted software development produces rapid code changes and large volumes of test artifacts, increasing the cognitive load on software engineers [14, 6, 18]. This scale and pace make it difficult to maintain oversight, assess quality, and make sound decisions, even as expectations for reliability remain high [25]. The current challenge is to ensure that testing supports human understanding and judgment when AI agents are used during software testing. Prior work introduced the notion of *test agents* and adaptive autonomy in regression test selection [9, 20], arguing that test selection and scheduling decisions can benefit from decentralized control. A recent study on ethical challenges in software test automation [25] identified explainability, logging and monitoring, privacy, technical risks, and human control as central concerns when AI is introduced into test automation. This raises a specific question for software testing: when LLMs explain generated tests, do engineers treat those explanations as hypotheses to inspect or as assurance to accept [1].

Capabilities that make test agents useful also create risks. If a test agent can generate tests, explain why they exist, summarize execution results, delegate subtasks to other agents and regenerate itself after software changes, then the tester role shifts. The human becomes a supervisor for (semi)autonomous work, and overreliance on test agents can indirectly weaken human oversight by undermining the evidence that justifies confidence.

This paper makes the following contributions. It frames overreliance on test agents as a supervisory control and assurance problem and develops this framing through three theoretical lenses: software testing as cognitive problem-solving, test agents as adaptively autonomous test artifacts, and test design argumentation for making generated tests more reviewable. In the end, it proposes a categorization of overreliance modes specific to AI-assisted software testing and outlines implications for test agent workflows and empirical studies.

2 Related Work and Theoretical Lens

Automated test generation techniques have been used in safety-critical development, such as in industrial control software [7]. Several studies [21] have shown that automated test generation is often more efficient than manual testing but remains less effective than manual test design by experienced engineers in detecting naturally occurring faults.

Research on the cognitive foundations of software testing [10, 3, 19] has focused on tester routines and problem-solving [13]. Together, these results suggest that test design relies on goal formulation, selection of test design methods, and goal representation and context-based heuristics that are rarely captured in current automated approaches. This cognitive perspective is central to this work.

A review [23] found that automation bias is evident in testing practice [11], making overreliance a particularly important focus. Biases such as confirmation bias and anchoring seem to be present in practice [11], making overreliance especially important to study in test design and review. This framing builds on classic automation literature showing that automation can improve performance and reduce human involvement in monitoring, making intervention more difficult when problems occur [4, 22].

In safety-critical software development, tests can contribute to claims about system behavior and release readiness through verification evidence and assurance practices such as ISO 26262 [17], EN 50128 [5], and the use of GSN [24].

2.1 From Test Cases to Test Agents

Building on the concept of adaptive autonomy for *test agents* [9, 20], one can view software test agents as capable of adjusting their autonomy levels in response to the testing context, available information, and observed system conditions. Instead of relying on a single centralized mechanism to decide which test cases to select or prioritize, one can rely on a decentralized form of coordination among agents. As illustrated in Figure 1, this allows individual agents

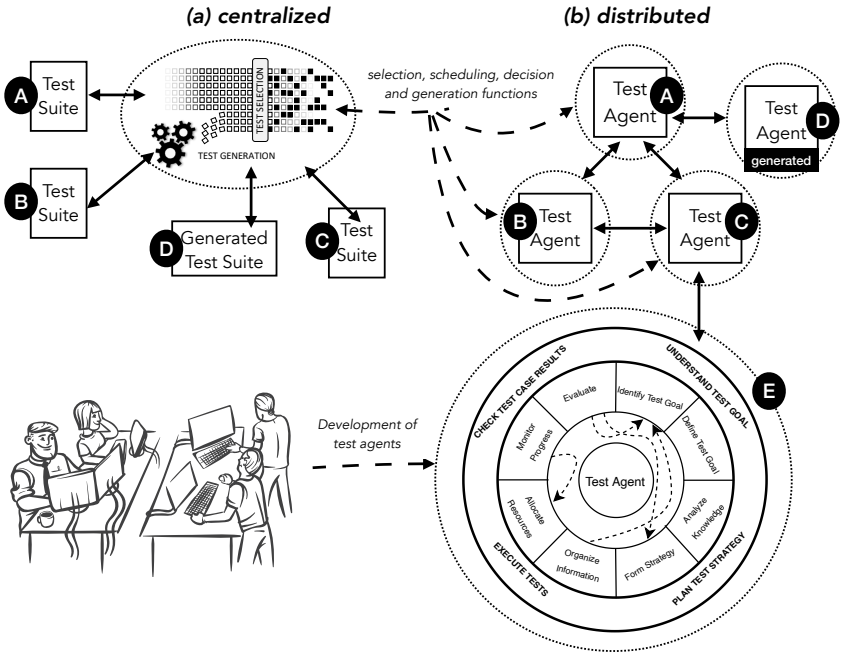

Fig. 1. From centralized regression testing (a) to supervised test agent workflows (b). The internal agent problem-solving model is included to indicate which elements of agent behavior should remain inspectable for reliance.

A, B, C, or D to make local decisions, interact with other agents when needed, and adapt their behavior. Traditional automated test cases are typically static artifacts. They contain input values, expected results, and executable scripts. They are selected, scheduled, and prioritized by a surrounding regression test system. In contrast, the test-agent vision proposes test cases that can reason, adapt, interact, and update their behavior over time. Test agents are intended to decentralize regression testing by allowing tests to know when to execute, how to adapt their purpose, and when to interact with other tests.

To avoid using the term *test agent* as a loose term for any AI tooling, we adopt the following definition in line with prior work on test agents and agent-based software testing [9, 20].

> **Definition (Test Agent).** A test agent is an autonomous or semi-autonomous software agent embedded in a testing workflow that can observe changes in the system under test and its testing environment, maintain explicit testing goals, generate, select or execute test artifacts, interact with other agents or humans and produce evidence relevant to assurance.

This changes what engineers must review. A test case becomes a partially autonomous entity in a testing workflow. Earlier test agent work [9] describes agents with states such as Idle, Interact, Execute, Regenerate, and Out of Order. A test agent may execute its own task, request assistance, respond to another

agent, or regenerate with help from a test engineer when its original purpose no longer holds. Test agent interactions can include non-committal information sharing, one-to-one dialogue, one-to-one delegation, and one-to-many dialogue or delegation, for example, when agents coordinate around coverage, execution time, or fault history. This means that test agents operate across several layers of testing work. They may generate tests, evaluate results, exchange evidence, delegate goals, monitor changes, and change future test execution. In the language of semi-executable artifacts, such workflows combine executable code, prompts, agent workflows, evaluation harnesses, policies, and human judgment. Feldt et al.'s [15] semi-executable stack describes these as artifacts whose behavior depends partly on deterministic execution and partly on human interpretation.

2.2 Cognitive Problem Solving and Argumentation

To understand overreliance in testing, one must first understand what human testers do. Software testing is not only about producing test inputs or executing scripts [2]. It is a *cognitive problem-solving activity* [13]. Testers interpret requirements, infer risks, formulate test goals, select techniques, construct test cases, judge adequacy, evaluate results, and communicate their meaning to others (as shown inside test agent E in Figure 1). AI-assisted testing can change this cognitive structure. It can support or replace parts of the tester's reasoning process. A model may propose the test goal, generate a concrete test, state the claim the test supports, provide a rationale, identify evidence, summarize the execution, and recommend whether a suite is adequate. For example, overreliance could occur when the engineer stops treating these outputs as hypotheses to be examined and starts treating them as evidence to be accepted. In complementary work on AI-assisted test generation, generated tests can be related to an explicit test goal, claim, reason, and evidence [12] through *test argumentation*. These results are relevant to overreliance, as they provide the initial conceptual basis for argumentative and inspectable AI-assisted testing.

3 A Framework for Studying Overreliance on Test Agents

This section turns the theoretical lenses into a conceptual framework for studying overreliance on test agents. It explains why the shift toward test agents makes overreliance both a supervisory-control and assurance problem, and it describes a data-collection setup for analyzing how test agents are implemented, used, evaluated, and revised. Finally, it identifies test-specific modes of overreliance that can serve as a checklist for designing test agent workflows.

3.1 Overreliance as a Supervisory Control and Assurance Problem

Prior work on ethical AI-powered test automation [25] identifies human control and responsibility as central concerns, asking how humans remain in control, how they interact with automation, how inadequate performance is detected, and who

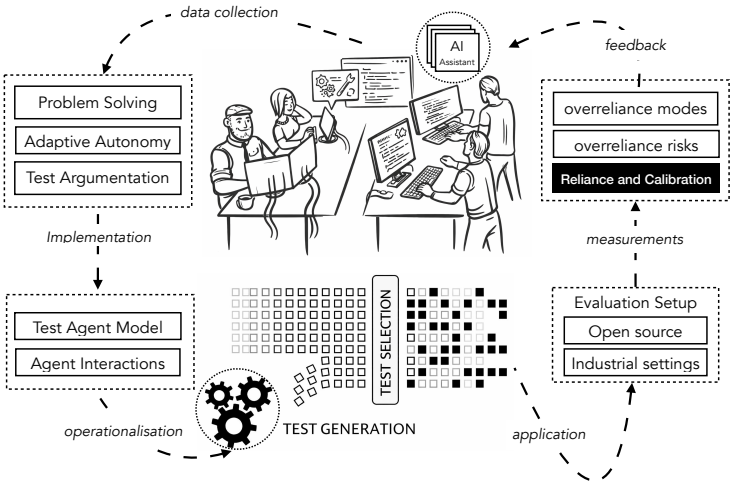

Fig. 2. Conceptual framework for studying overreliance-aware test agents. The framework links theoretical lenses, implementation of adaptive and argumentative test agents, and empirical evaluation in open-source and industrial settings.

is responsible when automation underperforms. The same work explicitly raises risks such as over-trust and flawed decision support in test automation.

Test agents [9] can amplify these risks. A statically generated test can be inspected. A test agent can act over time. It can change state, request help, delegate subtasks, regenerate, and produce artifacts and summaries. The human supervisor must therefore understand why it acted, which evidence it used, and when its results should be challenged.

Testing is also an assurance activity [2]. It supports claims about software behavior, quality, risk, and release readiness. In regulated and safety-critical settings, testing evidence often contributes to larger quality assurance. Prior work on test design argumentation [12] notes that regulated domains require evidence that testing has met integrity and certification objectives, and that argumentation can structure claims, strategies, and supporting evidence, even for individual test-case design. In AI-assisted testing, overreliance could mean accepting agent-produced testing artifacts, arguments, summaries, delegations, or regenerated suites as assurance without sufficient challenge of their arguments.

3.2 Data Collection for Overreliance on Test Agents

Figure 2 summarizes the steps proposed for future data collection. Building on the concept of adaptive autonomy for test agents [9, 20], software test agents can adjust their level of autonomy based on the testing context, available information, and observed system conditions. Cognitive models of test design and adaptive test agents [9, 10, 8, 16] can be operationalized as argumentative test agents. These agents are intended to capture the features of skilled human test-

Table 1. Overreliance modes and how they relate to AI-assisted software testing.

Mode	Risk Description
Test Goal Overreliance	Accepting the test goal selected by the test agent without checking whether it is the right one.
Test Strategy Overreliance	Accepting the chosen technique, e.g., boundary analysis, without asking if another strategy is needed.
Claim Overreliance	Accepting what the test agent says a test demonstrates.
Reason Overreliance	Accepting a plausible reason for why a test agent exists.
Evidence Overreliance	Treating logs, traces, coverage, or summaries as sufficient evidence without checking relevance.
Test Argument Overreliance	Accepting a coherent generated rationale without challenging the connecting warrant.
Test Execution Overreliance	Trusting execution summaries instead of inspecting specific failures, skipped tests, uncertainty or evidence.
Oracle Overreliance	Accepting generated expected results or assertions despite weak support.
Delegation Overreliance	Trusting that another test agent handled a delegated subtask correctly.
Regeneration Overreliance	Accepting regenerated tests as improved, equivalent, or sufficient.
Escalation Overreliance	Assuming the test agent will ask for human help when needed.
Maintenance Overreliance	Treating agent-updated tests as preserving the historical test intent(s).

ing: they can pursue explicit test goals, adjust their level of autonomy to the situation at hand, interact with other agents and human engineers when needed, and make their reasoning visible through explicit arguments. This can be operationalized in adaptive test agents that maintain explicit representations of goals, reasons, claims, assumptions, and supporting evidence [12].

Methodologically, the framework in Figure 2 can be operationalized as an iterative loop: collect data and model test-agent behavior, operationalize the relevant constructs, apply the workflow, measure testing efficiency, effectiveness, and reliance outcomes, and use the results to inform the framework. Empirical studies should combine open-source systems with industrial-scale software and use software versions with known naturally occurring faults.

Reliance measures may include *acceptance of incorrect claims, the frequency of challenged outputs, evidence inspection behavior, human overrides and detection of lost test goals.* The results can be used to iteratively refine the overreliance modes and risks toward an updated model of reliance for test agents.

3.3 Overreliance Modes in AI-Assisted Test Design

In this section, I apply this proposal to collect data on the conceptual view of test design and test agents. Table 1 summarizes the modes where overreliance can occur and the risks it can pose in software testing. It shows that overreliance can occur at different points in the tester's problem solving process, from accepting an agent's test goal or strategy to trusting its claims, evidence, execution, oracles, delegated subtasks, and regeneration. This is intended as an initial checklist for identifying overreliance. This analysis uses a problem-solving lens and a test

agent perspective, but offers only a partial view from the author's perspective. Without case study evidence, claims about the overreliance remain interpretive. Nevertheless, overreliance on software testing takes distinct forms when viewed through the lenses of test agents, problem-solving, and test argumentation.

Illustrative Scenario. Consider an industrial control system after a requirements change. Test Agent A detects the change and regenerates boundary tests, then delegates input interaction checks to Test Agent B. Agent B reports no failures, but its evidence covers only single interaction cases. Test Agent C flags that an important interaction scenario remains untested. The regenerated tests are assumed to preserve prior intent, limited evidence is treated as sufficient, delegated work is trusted too broadly and a summary hides an untested scenario.

4 Conclusions and Limitations

This paper argues that overreliance on test agents should be understood as an agency, supervisory control, and assurance problem in AI-assisted software testing. By viewing software testing as cognitive problem-solving, we show how agentic AI can shift human work from direct test design to the supervision of test agents. This shift creates testing-specific modes of overreliance, including overreliance on goals, strategies, arguments, oracles, delegation, regeneration, and maintenance. The proposed modes should be treated as a working taxonomy. Future work should study these modes in open-source and industrial settings, develop measures of reliance and quality, and evaluate workflow mechanisms.

Acknowledgments. This work was supported by Software Center, MONA LISA and MATISSE (101140216) projects and the AI and Society Fellowship.

References

1. Akbarova, S., Dobslaw, F., Feldt, R.: Understanding on the edge: Llm-generated boundary test explanations. arXiv preprint arXiv:2601.22791 (2026)
2. Ammann, P., Offutt, J.: Introduction to software testing. Cambridge University Press (2016)
3. Aniche, M., Treude, C., Zaidman, A.: How developers engineer test cases: An observational study. IEEE Transactions on Software Engineering **48**(12), 4925–4946 (2021)
4. Bainbridge, L.: Ironies of automation. In: Analysis, design and evaluation of man–machine systems, pp. 129–135. Elsevier (1983)
5. CENELEC: 50128: Railway Application–Communications, Signaling and Processing Systems–Software for Railway Control and Protection Systems. In: Standard Report (2001)
6. Chen, H., Chen, K., Zhang, F., Wang, T., Cheng, L.: AgentTester: An LLM-based tool for unit test generation with automatically generated prompts. In: International Conference on Intelligent Computing. pp. 114–126. Springer (2025)

7. Enoiu, E., Čaušević, A., Ostrand, T., Weyuker, E., Sundmark, D., Pettersson, P.: Automated test generation using model checking: an industrial evaluation. STTT **18**(3) (2016)
8. Enoiu, E., Feldt, R.: Towards human-like automated test generation: Perspectives from cognition and problem solving. In: CHASE. pp. 123–124. IEEE (2021)
9. Enoiu, E., Frasheri, M.: Test agents: The next generation of test cases. In: NEXTA. pp. 305–308. IEEE (2019)
10. Enoiu, E., Tukseferi, G., Feldt, R.: Towards a model of testers' cognitive processes: Software testing as a problem solving approach. In: QRS. pp. 272–279. IEEE (2020)
11. Enoiu, E.P., Cusmaru, A., Malm, J.: Unveiling cognitive biases in software testing: Insights from a survey and controlled experiment. In: 2024 31st Asia-Pacific Software Engineering Conference (APSEC). pp. 422–431. IEEE (2024)
12. Enoiu, E.P., Feldt, R.: Test design and review argumentation in AI-Assisted test generation. In: ITEQS Workshop (May 2026)
13. Enoiu, E.P., Gay, G., Esber, J., Feldt, R.: Understanding problem solving in software testing: An exploration of tester routines and behavior. In: IFIP International Conference on Testing Software and Systems. pp. 143–159. Springer (2023)
14. Fakhoury, S., Naik, A., Sakkas, G., Chakraborty, S., Lahiri, S.K.: Llm-based test-driven interactive code generation: User study and empirical evaluation. IEEE Transactions on Software Engineering **50**(9), 2254–2268 (2024)
15. Feldt, R.: Keynote: Agentic Software Engineering Will Eat the World: AI-Based Systems as the New Operating System of Society. International Workshop on Agentic Engineering, AGENT 2026, co-located with ICSE 2026 (2026), https://conf.researchr.org/details/icse-2026/agent-2026-papers/31/Keynote-Agentic-Software-Engineering-Will-Eat-the-World-AI-Based-Systems-as-the-New, keynote talk, Rio de Janeiro, Brazil, 14 April 2026. Accessed: 2026-05-08
16. Frasheri, M., Çürüklü, B., Ekström, M.: Towards collaborative adaptive autonomous agents. In: ICAART (1). pp. 78–87 (2017)
17. Griessnig, G., Schnellbach, A.: Development of the 2nd edition of the ISO 26262. In: European Conference on Software Process Improvement. pp. 535–546. Springer (2017)
18. Harman, M., Ritchey, J., Harper, I., Sengupta, S., Mao, K., Gulati, A., Foster, C., Robert, H.: Mutation-guided llm-based test generation at meta. In: Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering. pp. 180–191 (2025)
19. Itkonen, J., Mäntylä, M.V., Lassenius, C.: The role of the tester's knowledge in exploratory software testing. TSE **39**(5), 707–724 (2012)
20. Kumaresen, P.P., Frasheri, M., Enoiu, E.P.: Agent-based software testing: A definition and systematic mapping study. In: QRC. pp. 24–31. IEEE (2020)
21. Kurmaku, T., Enoiu, E.P., Kumrija, M.: Human-based test design versus automated test generation: A literature review and meta-analysis. In: ISEC. pp. 1–11 (2022)
22. Parasuraman, R., Sheridan, T.B., Wickens, C.D.: A model for types and levels of human interaction with automation. IEEE Transactions on systems, man, and cybernetics-Part A: Systems and Humans **30**(3), 286–297 (2000)
23. Romeo, G., Conti, D.: Exploring automation bias in human–ai collaboration: a review and implications for explainable ai. Ai & Society **41**(1), 259–278 (2026)
24. Spriggs, J.: GSN-The Goal Structuring Notation: A structured approach to presenting arguments. Springer Science & Business Media (2012)
25. Strandberg, P.E., Enoiu, E.P., Frasheri, M.: Ethical challenges and software test automation. AI and Ethics **5**(6), 6185–6206 (2025)