Whose Oracle Is It?
The Authority Problem in Deployed Agentic Program Repair

Bardia Mohammadi
bmohammadi@mpi-sws.org
Max Planck Institute for Software Systems
Saarbrücken, Germany

Laurent Bindschaedler
bindsch@mpi-sws.org
Max Planck Institute for Software Systems
Saarbrücken, Germany

Abstract

Tests play two roles in automated program repair (APR): they are the evidence that accepts a patch, and they are repository artifacts that may themselves need repair. Benchmarks resolve the tension by making evaluator-owned tests immutable; deployment cannot, because tests legitimately evolve with the code they check. We argue that the resulting question, who may authorize a change to the evidence that accepts a patch, is a problem of allocating authority, distinct from whether a patch is correct, and that no diff separates an authorized change of requirements from an agent weakening its own acceptance test. We propose an *admissible repair witness*, evidence that exists before the agent runs and that the agent cannot rewrite, and a *two-zone* policy that protects it while the rest of the test surface evolves under review. In 230 merged bug-fix pull requests from nine repositories, 160 of 172 test-changing repairs (93.0%) edit a test artifact that already exists, so locking the test surface is not an option. Of the 172, 87 delete test lines, but only 53 delete an assertion, a test case, an expected output, or a whole test file: escalating on deleted lines alone sends half of all test-changing repairs to a human, while reading what was deleted sends at best under a third.

CCS Concepts

• **Software and its engineering → Software testing and debugging**; *Software maintenance tools.*

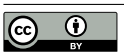
Keywords

automated program repair, coding agents, test oracles, patch validation, test evolution, provenance

ACM Reference Format:
Bardia Mohammadi and Laurent Bindschaedler. 2026. Whose Oracle Is It? The Authority Problem in Deployed Agentic Program Repair. In *Proceedings of the 7th International Workshop on Automated Program Repair (APR '26), October 12–16, 2026, Munich, Germany.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3843776.3844673

1 Introduction

An agent is asked to fix a bug. Its patch breaks an existing test, so the agent edits the test; continuous integration turns green, and the pull request reaches review with a passing build. Nothing in that diff says whether the test was wrong or the patch was.



APR '26, Munich, Germany

ACM ISBN 979-8-4007-2988-1/2026/10
https://doi.org/10.1145/3843776.3844673

Generate-and-validate repair has always leaned on incomplete oracles: passing the available tests makes a patch plausible, not correct [1, 13]. Agentic APR sharpens the problem, because the actor that writes production code can also rewrite the artifacts used to accept it. In controlled evaluations this appears as reward hacking: agents delete tests, exploit harnesses, or otherwise score well without doing the intended task [9, 19, 22].

Benchmark owners can hide, restore, or randomize tests. Deployment cannot. Tests, fixtures, snapshots, and runner configuration are ordinary repository artifacts that evolve with the code they check [4, 12], and some of them are repair targets. Locking the whole test surface would forbid work that APR is expected to do.

We therefore ask: *how should deployed agentic APR permit legitimate test repair while preventing the repair actor from weakening the evidence used to accept its own patch?* Our position is that this is a question about *authority* (who may authorize a change to acceptance evidence), and that it is separate from whether a patch is correct. Answering it requires neither inferring intent nor proving correctness.

We contribute an identification argument for why a diff cannot settle the question (§2), a placement against related work (§3), two constructs and a review gate that make the position operational (§4), and a study of 230 merged bug-fix pull requests measuring what such a policy would cost (§5).

2 The Authority Problem

Why a Diff Cannot Answer. Let P be a production patch and ΔT a test change that replaces an assertion of behavior b_0 with b_1. Two histories produce the same pair. Under *authorized migration*, the requirement really changed to b_1: the old test is stale, and code and test must co-evolve. Under *specification erosion*, the requirement is still b_0: P violates it, and ΔT weakens the oracle until P looks acceptable.

Running more tests can show that the code is consistent with b_1; it cannot show that anyone with standing chose b_1, and a model-written rationale is a claim by the party under review, not independent evidence. Flaky-test suppression makes this concrete: developers do skip flaky tests, accepting the loss in test effectiveness [10]. The identical skip applied to an intermittent failure that P introduced instead erases evidence against P. The diff and the green build do not tell the two apart.

Why Authority Rather Than Validation. A finite suite observes only part of the intended behavior, so stronger validation narrows uncertainty but never authorizes a change of intent. *Separation of privilege* asks instead for independently controlled conditions on a sensitive decision [16]. Applied here, proposing a patch and ratifying a relaxation of the evidence that accepts it are different

privileges: one acceptance condition stays anchored outside the actor, and somebody else approves any relaxation of it. We assume an actor optimizing for a passing signal rather than one deliberately subverting, so the gate enforces procedural integrity, not adversarial robustness.

3 Literature Synthesis and Research Gap

Patch Correctness. APR has long separated plausible patches from correct ones: weak suites admit overfitting, and automated correctness assessment remains imperfect [11, 13, 20]. Recent agentic systems infer intent, build contextual rubrics, or abstain and validate [2, 14, 15], all asking whether a patch conforms to available or inferred intent. We ask a prior question: whether the evidence granting that confidence is still authorized after the candidate has changed it.

Benchmark Integrity. ImpossibleBench measures shortcut-taking, BenchJack audits exploitable harnesses, and CapCode uses randomized re-evaluation to catch test gaming [9, 19, 22]; PatchDiff exposes behavioral divergence among test-passing SWE-bench patches [21]. These defenses isolate or randomize a fixed, evaluator-owned oracle. Production cannot draw the same boundary, because some oracle artifacts are repair targets.

Test Evolution and Cogeneration. Suites evolve by addition, deletion, and repair; ReAssert and TEBench target stale or missing tests [4, 12, 17]. That work establishes mutability as legitimate, but it usually receives the target behavior from a developer change or from benchmark ground truth. Closest to deployment, Cheng et al. have an APR agent emit a fix and a bug-reproduction test in one patch, and build patch selectors that account for the test change [3]. Cogeneration is precisely the configuration our question is about: the generated test is useful evidence, but because it is derived from the candidate's own author, it cannot by itself be the evidence that ratifies that candidate.

Control, Provenance, and the Gap. Abstain and Validate separates proposal from ratification, but decides conformance to inferred intent rather than who may alter the oracle [2]; AI Control supplies a general untrusted-proposer and trusted-ratifier structure [7]; in-toto attests artifact provenance and authorized steps [18]. We specialize these to behavioral acceptance evidence. No prior work combines all three requirements: permitting legitimate test evolution, separating candidate authorship from ratification of oracle-weakening changes, and binding acceptance evidence to provenance and to a time before the candidate existed. We call that gap the *authority problem*, and test governance the policy layer that answers it.

4 An Operational Baseline

4.1 Admissible Repair Witnesses

A *repair witness* $W = (o, p, t, v, a)$ is a checkable observation or predicate o with provenance p, capture time t, applicable repository or specification version v, and an authority a answerable for the constrained behavior. A failing CI case, a crash reproducer, a protocol-conformance check, and a maintainer-approved acceptance example all qualify. W is *admissible* when it is fixed before the candidate is generated; not derived solely from that candidate or its author; recorded outside the actor's write boundary along with its version and environment; and issued by someone entitled to approve the behavior.

Provenance and authority are different properties. A CI service can establish when a failure occurred without being entitled to redefine expected behavior, and differential testing shows the split cleanly: Mokav produces a checkable, version-bound behavioral difference but says nothing about which behavior is intended [5]. Authority is also scoped: a maintainer may ratify a component contract but not a repository-wide one, so the record must state the basis and behavioral scope of a rather than treat human authorship, organizational membership, or model independence as authority by default.

Availability is therefore a real precondition: an issue, a generated test, or a model rationale stays a candidate source until its timing, precision, version, and authority are established. Independence does not require a human: a pre-existing crash report qualifies, whereas a second model gains no authority by agreeing. Without pre-candidate evidence the agent may still generate P, diagnostic tests, and a proposed witness (as in cogeneration), but autonomous acceptance waits for an authorized binding. Specification changes stay possible: an authority records $A : S_0 \to S_1$ and rebinds W to S_1.

4.2 Two Zones and What a Witness Depends On

The policy splits the test surface. In the *witness zone*, at least one admissible witness and its provenance record are immutable to the repair actor while the candidate is generated, and the gate checks that the witness was satisfied and its execution path not bypassed. In the *governed zone*, other tests, fixtures, snapshots, and runner configuration may change: additions and mechanical repairs take ordinary review, whereas relaxing an assertion, deleting coverage, changing expected output, or editing the harness needs someone else to approve the semantic effect.

Zones are integrity domains, not file lists. What a witness means can depend on fixtures, root configuration, test runners, CI definitions, dependency pins, and environment state. Its closure $D(W)$ holds these acceptance-path dependencies but excludes the production code the witness is meant to observe; including the repair target would make repair impossible. Operationally, $x \in D(W)$ when, holding the witness input and the repair target fixed, changing x can change whether W runs or how its observation becomes a verdict. That is narrower than repository reachability, and any uncertain dependency escalates rather than being silently admitted.

A Worked Instance. In `pytest` issue #13083, a project member filed a bug-labeled report 54 hours before PR #13086, linked a pinned pre-repair revision, and stated the predicate: scanning a missing directory returns an empty list. Minimally, W is a hash-preserved snapshot of that record and $D(W) = \{$approved runner, pinned image$\}$; conservatively, $D(W)$ also holds the CI bootstrap, dependency locks, and runner configuration. The pull request changes four artifacts, none in either closure: the production target is excluded by definition, and its test stays mutable. The repair therefore proceeds without escalation under both mappings. One trace shows

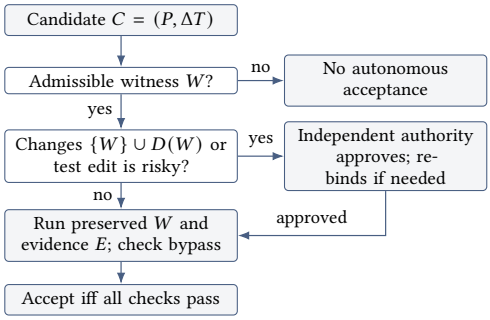


Figure 1: Governed acceptance. Risk signals route authority; they do not decide whether a test change is legitimate.

the corridor between locking everything and reviewing everything is not empty; §5 measures how wide it is.

Path ownership [6] can enforce a declared boundary at review time, but cannot establish that evidence predates the candidate, discover the closure, or authorize a changed requirement. A large $D(W)$ grows the locked side while an aggressive risk predicate sends the rest to review, so what survives is the set of candidates with an admissible W that leave $\{W\} \cup D(W)$ unchanged and raise no sign of weakened tests. If that set is empty the policy is merely universal review, so witness coverage, closure size, edit–closure overlap, and escalation rate decide feasibility.

4.3 The Gate

Figure 1 turns the model into a review decision. Write-boundary enforcement is primary for W and whatever part of $D(W)$ can be enclosed; acceptance-time detection is a backstop for dependencies that must stay writable. A candidate is *risky* when it may modify or bypass $\{W\} \cup D(W)$, relax behavior, or reduce test strength; uncertainty also routes it to review. The predicate is procedural and does not declare an edit illegitimate: for a migration, an independent authority records $A : S_0 \to S_1$ and rebinds W. Without an admissible W or the required A, generation may continue but autonomous acceptance cannot. The gate logs its branch and the acceptance packet $(P, \Delta T, W, A, E)$, with E the evidence run at acceptance, for audit. Passing establishes compliance with the authority policy; independent tests still address correctness, since witness governance alone cannot stop a patch that special-cases the witness.

5 What the Policy Would Cost

Two quantities bound the cost. How often does repair edit a test artifact that already exists? If fixes mostly add tests, immutability is cheap. And how often would an escalation trigger fire? We measure both on merged, maintainer-labeled bug-fix pull requests rather than on SWE-bench, whose construction conditions on test-modifying pull requests [8].

5.1 Method and Results

The protocol fixes ten public repositories and their maintainer-applied bug labels, enumerates merged 2025 pull requests whose pull request or GitHub-linked closing issue carries a configured label, then samples at most 30 per repository with a fixed seed.

Table 1: Descriptive, design-weighted pull-request-level counts and proportions.

Quantity	Result
Eligible labeled pull requests	4,184
Sampled pull requests (contributing repositories)	230 (9)
Pull requests with detected test changes	172
touch a pre-existing test artifact	160 (93.0%)
add only new test files	12 (7.0%)
delete lines in a test artifact	87 (50.6%)
delete an assertion, case, expectation, or file	53 (30.8%)
Linked issue (all sampled pull requests)	188/230 (81.7%)

Nine repositories contributed 4,184 eligible and 230 sampled pull requests; one stratum was empty. Path rules identify test artifacts, file status separates additions from edits, and line counts identify deletions. A second pass re-reads the cached diffs and labels every deleted line in a test artifact as an assertion, a test-case declaration, an expected-output line, incidental (imports, comments, blanks, and text re-added elsewhere in the same file), or unclassified.

Of 230 sampled pull requests, 172 changed a test artifact. Within this design, 160 (93.0%) touched a test artifact that already existed and only 12 (7.0%) added test files exclusively (Table 1), so blanket immutability is unavailable; repository-level rates ranged from 57.1% to 100% (unweighted mean 95.2%). Of the 172, 87 (50.6%) deleted lines in a test artifact, but only 53 (30.8%) deleted an assertion, a test case, an expected output, or a whole test file. Reading *what* a deletion removes rather than *that* something was deleted thus retires at best 39.1% of the escalations a line-count trigger raises. Nine of the 87 delete nothing but imports, comments, and moved text; 25 cannot be decided from syntax alone and escalate under a conservative default, placing refined escalation between 30.8% and 45.3% of test-changing repairs. Fourteen also add a skip or xfail marker, a cheap and separate signal.

Escalation load varies by repository: the oracle-bearing share of deletion-bearing repairs runs from 35.7% to 80.0% across the eight strata that have any, and all 12 add-only pull requests come from the Rust stratum, whose file-oriented compiler tests explain its 57.1% outlier. Triggers therefore need per-repository calibration. Across all 230 repairs, 188 (81.7%) link to a labeled bug issue.

5.2 Interpretation and Limitations

The pooled proportions are design-weighted by the per-repository cap, not population estimates. Repository and label choices restrict external validity, path rules can misclassify tests, and descriptive Wilson intervals (88.2%–96.0% for 93.0%, 24.4%–38.1% for 30.8%) ignore repository clustering; every add-only observation comes from the Rust stratum, making 7.0% especially design-dependent.

Touching an existing file may only append a regression case, and an oracle-bearing deletion may still be refactoring, snapshot replacement, or authorized migration. So 93.0% supports required write access and 30.8% bounds the review demand of a semantic trigger; neither measures erosion. The deleted-line labels are syntactic and reproducible, validated against all nine incidental cases and ten randomly drawn oracle-bearing ones, but migration versus relaxation still needs human adjudication. A linked issue may likewise be late,

vague, version-ambiguous, or unauthorized; because the collector retained neither its timestamp nor its content, 81.7% is only an upper bound on issue-linked witness availability. Collector, labeler, and data are at https://github.com/mpi-dsg/test-governance.

6 Agenda and Counterarguments

A direct evaluation should cross TEBench's legitimate test evolution with ImpossibleBench's adversarial specification conflicts, comparing total immutability, unrestricted mutability, and the two-zone gate: a useful gate preserves more legitimate evolution than immutability and admits fewer specification-violating patches than unrestricted mutability. **RQ1**: how often does admissible evidence precede repair, coded by temporal boundary, behavioral precision, version binding, and authority scope, not by issue linkage, which our own 81.7% shows to be a permissive proxy. **RQ2**: how do closure size and repository layout constrain what can be accepted without human sign-off. **RQ3**: how should semantic triggers be calibrated to strength loss, impact, and reversibility; our labels cut escalation volume by 10.3%–39.1%, and adjudicating the same 87 pull requests by hand as addition, mechanical repair, migration, or relaxation would show how much precision remains. **RQ4**: which evidence packet supports accurate authorization at low review cost. Studies should publish their adjudication protocol, double-code each dimension, and keep disputed authority distinct from absent evidence.

Three counterarguments delimit the position. A stronger validator or a second model reduces erroneous edits but cannot confer authority over intended behavior; the two are complementary. Path locks enforce a chosen boundary but cannot define $D(W)$ or authorize changed behavior. Reviewing every test edit avoids the question only by removing autonomous acceptance, and our numbers price that choice: every test-changing repair, against 30.8%–45.3% for a semantic trigger. A false escalation delays one repair, whereas accepted erosion weakens every later validation, which argues for conservative, per-repository defaults.

7 Conclusion

Repair actors may propose test changes; they should not ratify their own acceptance evidence. Admissible witnesses and governed mutability give that principle an operational vocabulary and a baseline to study. Practicality turns on two measurable things: whether independent evidence exists before repair, and whether protecting it leaves a useful corridor between locking everything and reviewing everything. On real bug fixes that corridor exists but needs calibration: 93.0% of test-changing repairs edit tests that already exist, and reading what a deletion removes rather than counting deleted lines cuts escalation from half of those repairs to between a third and a half.

References

[1] Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. 2015. The Oracle Problem in Software Testing: A Survey. *IEEE Transactions on Software Engineering* 41, 5 (2015), 507–525. doi:10.1109/TSE.2014.2372785

[2] José Cambronero, Michele Tufano, Sherry Shi, Renyao Wei, Grant Uy, Runxiang Cheng, Chin-Jung Liu, Shiying Pan, Satish Chandra, and Pat Rondon. 2026. Abstain and Validate: A Dual-LLM Policy for Reducing Noise in Agentic Program Repair. In *Proceedings of the IEEE/ACM 48th International Conference on Software Engineering: Software Engineering in Practice*. 118–129. doi:10.1145/3786583.3786858

[3] Runxiang Cheng, Michele Tufano, José Cambronero, Renyao Wei, Sherry Shi, Grant Uy, Pat Rondon, and Franjo Ivančić. 2026. Dynamic Cogeneration of Bug Reproduction Test in Agentic Program Repair. In *Companion Proceedings of the 34th ACM International Conference on the Foundations of Software Engineering*. ACM, 644–654. doi:10.1145/3803437.3805237

[4] Brett Daniel, Vilas Jagannath, Danny Dig, and Darko Marinov. 2009. ReAssert: Suggesting Repairs for Broken Unit Tests. In *Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering*. 433–444. doi:10.1109/ASE.2009.17

[5] Khashayar Etemadi, Bardia Mohammadi, Zhendong Su, and Martin Monperrus. 2025. Mokav: Execution-Driven Differential Testing with LLMs. *Journal of Systems and Software* 230 (2025), 112571. doi:10.1016/j.jss.2025.112571

[6] GitHub. 2026. About Code Owners. https://docs.github.com/articles/about-code-owners. Accessed 2026-08-08.

[7] Ryan Greenblatt, Buck Shlegeris, Kshitij Sachan, and Fabien Roger. 2024. AI Control: Improving Safety Despite Intentional Subversion. In *Proceedings of the 41st International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235)*. 16295–16336.

[8] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-world GitHub Issues?. In *The Twelfth International Conference on Learning Representations*. https://proceedings.iclr.cc/paper_files/paper/2024/hash/edac78c3e300629acfe6cbe9ca88fb84-Abstract-Conference.html

[9] Thanawat Lodkaew, Johannes Ackermann, Soichiro Nishimori, Nontawat Charoenphakdee, Masashi Sugiyama, and Takashi Ishida. 2026. Do Coding Agents Deceive Us? Detecting and Preventing Cheating via Capped Evaluation with Randomized Tests. In *ICML Workshop on Statistical Frameworks for Uncertainty in Agentic Systems*. doi:10.48550/arXiv.2606.07379

[10] Qingzhou Luo, Farah Hariri, Lamyaa Eloussi, and Darko Marinov. 2014. An Empirical Analysis of Flaky Tests. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 643–653. doi:10.1145/2635868.2635920

[11] Martin Monperrus. 2014. A Critical Review of "Automatic Patch Generation Learned from Human-Written Patches": Essay on the Problem Statement and the Evaluation of Automatic Software Repair. In *Proceedings of the 36th International Conference on Software Engineering*. 234–242. doi:10.1145/2568225.2568324

[12] Leandro Sales Pinto, Saurabh Sinha, and Alessandro Orso. 2012. Understanding Myths and Realities of Test-Suite Evolution. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. doi:10.1145/2393596.2393634

[13] Zichao Qi, Fan Long, Sara Achour, and Martin C. Rinard. 2015. An Analysis of Patch Plausibility and Correctness for Generate-and-Validate Patch Generation Systems. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*. 24–36. doi:10.1145/2771783.2771791

[14] Mohit Raghavendra, Anisha Gunjal, Bing Liu, and Yunzhong He. 2026. Agentic Rubrics as Contextual Verifiers for SWE Agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 15265–15290. doi:10.18653/v1/2026.acl-long.697

[15] Haifeng Ruan, Yuntong Zhang, and Abhik Roychoudhury. 2025. SpecRover: Code Intent Extraction via LLMs. In *Proceedings of the 47th International Conference on Software Engineering*. 963–974. doi:10.1109/ICSE55347.2025.00080

[16] Jerome H. Saltzer and Michael D. Schroeder. 1975. The Protection of Information in Computer Systems. *Proc. IEEE* 63, 9 (1975), 1278–1308. doi:10.1109/PROC.1975.9939

[17] Ye Shang, Quanjun Zhang, Haichuan Hu, Chunrong Fang, Liang Xiao, and Zhenyu Chen. 2026. Breaking, Stale, or Missing? Benchmarking Coding Agents on Project-Level Test Evolution. arXiv preprint arXiv:2605.06125. doi:10.48550/arXiv.2605.06125

[18] Santiago Torres-Arias, Hammad Afzali, Trishank Karthik Kuppusamy, Reza Curtmola, and Justin Cappos. 2019. in-toto: Providing Farm-to-Table Guarantees for Bits and Bytes. In *28th USENIX Security Symposium*. 1393–1410.

[19] Hao Wang, Hanchen Li, Qiuyang Mang, Alvin Cheung, Koushik Sen, and Dawn Song. 2026. Do Androids Dream of Breaking the Game? Systematically Auditing AI Agent Benchmarks with BenchJack. arXiv preprint arXiv:2605.12673. doi:10.48550/arXiv.2605.12673

[20] Shangwen Wang, Ming Wen, Bo Lin, Hongjun Wu, Yihao Qin, Deqing Zou, Xiaoguang Mao, and Hai Jin. 2020. Automated Patch Correctness Assessment: How Far Are We?. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. 968–980. doi:10.1145/3324884.3416590

[21] You Wang, Michael Pradel, and Zhongxin Liu. 2026. Are "Solved Issues" in SWE-bench Really Solved Correctly? An Empirical Study. In *Proceedings of the IEEE/ACM 48th International Conference on Software Engineering*.

[22] Ziqian Zhong, Aditi Raghunathan, and Nicholas Carlini. 2026. ImpossibleBench: Measuring LLMs' Propensity of Exploiting Test Cases. In *The Fourteenth International Conference on Learning Representations*. https://proceedings.iclr.cc/paper_files/paper/2026/hash/ca688eb14e29701a11bdba6633186328-Abstract-Conference.html