Resource Allocation and Optimization
Volume 1 · Number 1 · 2026

Cheap Reference Executions Can Hide Expensive Mistakes: A Three-Test Selection Audit

Simon Hayes[1] · Liam Hall[2] · Evan Anderson[3]

[1] Spain · [2] Portugal · [3] Austria

Original publication: October 31, 2026 | Article 659

Abstract

Choosing short tests can save work while removing useful evidence. On 26 QuixBugs tasks, selecting the three cases with the fewest corrected-reference execution lines reduced recorded pair-audit work from a uniform-policy expectation of 429,531 line events to 326,749. Mean bug detection fell from 92.7% to 76.9%. The geometric mean task-level work ratio was 0.311, illustrating how a typical proportional saving can differ from the total-work saving. These finite-benchmark results expose a workload-detection tradeoff rather than an unconditional testing optimization.

Exploratory research draft. Fixed public program pairs and recorded execution events; no wall-clock speedup, newly generated repair, or fine-tuning result.

Cheap Reference Executions Can Hide Expensive Mistakes: A Three-Test Selection Audit

> Exploratory research draft. Fixed public program pairs and recorded execution events; no wall-clock speedup, newly generated repair, or fine-tuning result.

Abstract

Choosing short tests can save work while removing useful evidence. On 26 QuixBugs tasks, selecting the three cases with the fewest corrected-reference execution lines reduced recorded pair-audit work from a uniform-policy expectation of 429,531 line events to 326,749. Mean bug detection fell from 92.7% to 76.9%. The geometric mean task-level work ratio was 0.311, illustrating how a typical proportional saving can differ from the total-work saving. These finite-benchmark results expose a workload-detection tradeoff rather than an unconditional testing optimization.

1. A test budget needs an evidence objective

Limited evaluation resources create pressure to retain a small set of fast tests. A simple heuristic is to prefer cases whose reference implementation executes few source lines. Such cases may be convenient for constructing or checking many candidate comparisons. Yet a short reference execution need not expose the behavior that distinguishes a buggy candidate. The heuristic uses one kind of information to allocate effort toward an outcome that it does not directly measure.

We ask how bounded testing workload and bug detection change when three cases are selected by reference execution count rather than uniformly retained. The comparison holds the number of tests constant within every program. Its purpose is to inspect the tradeoff between a measurable work proxy and the evidence retained by the tests. The public QuixBugs pairs Lin et al., 2017 provide fixed candidates and expected outputs, while regression-test prioritization research Rothermel et al., 2001 motivates treating test order and selection as consequential decisions rather than incidental implementation details.

The question is relevant when tests help construct training or evaluation preferences for coding systems. Tao and Kamata, 2026 study fine-tuning coding assistants from pairwise preferences about what to change. We do not train an assistant or use its preference data. Instead, we examine a possible upstream choice: which available cases are retained when a fixed source pair is audited? If a cost heuristic removes discriminating tests, the resulting relation can become less informative even though its construction requires fewer recorded execution events.

The analytical proposition is a tradeoff, not a presumption of free efficiency. Reference-cost selection may lower pair-audit work, but it can also lower the probability of detecting the captured buggy implementation. The two outcomes must therefore be reported together. A fixed-count heuristic is not automatically a fixed-work policy, and reduced measured work is not automatically improved decision quality. The experiment uses finite recorded executions to make each component of that statement explicit.

2. The eligible source pairs and trace record

The starting benchmark execution record contains thirty-one JSON-backed Python tasks. We restrict this study to the twenty-six whose repository-designated corrected implementation passed every captured case under the declared compatibility oracle. The retained tasks contain 198 cases. That eligibility rule is inherited from the prior execution audit, rather than introduced to improve the cost heuristic's outcome. Tasks failing the reference gate remain outside this comparison, and their exclusion limits the claim to the compatible finite cohort.

The underlying calls use reviewed pure functions, copied JSON arguments, explicit standard-library helpers, and a limit of one hundred thousand executed source lines per candidate call. Generator outputs are materialized, tuple and list containers normalized for comparison, and numeric values checked with the recorded tolerance. The broader oracle problem Barr et al., 2015 explains why those conventions belong in the evidence record. A reference-pass gate does not establish complete semantic correctness beyond the supplied cases; it defines a reproducible comparison cohort.

Executed lines are recorded through the Python tracing interface Python Software Foundation, 2026. Only events in the reviewed benchmark function source contribute to the count. Operations inside compiled helpers are not assigned a corresponding number of source-line events. A single traced line can also do very different amounts of computation depending on its expression and input. Cyclomatic complexity McCabe, 1976 describes another structural property, but it is neither our selection variable nor an inferred replacement for these measured execution counts.

The resulting workload unit is pair-audit source-line events: the sum of reference and buggy-candidate events for selected cases. It is not elapsed time, processor instructions, memory consumption, or a monetary cost. Measurement cautions illustrated by Mytkowicz et al., 2009 and rigorous benchmarking discussion Kalibera and Jones, 2013 motivate keeping those quantities separate. No timing experiment is performed here, and no numerical line-count reduction is converted into an advertised runtime speedup.

3. Two policies with the same case count

The cost-guided policy orders each program's cases by corrected-reference execution-line count, breaking ties by the captured source-case index. It keeps the first three. The ranking does not inspect buggy execution status, buggy work, or whether a particular test detects the bug. This information restriction matters: a policy that chose cases after examining detection would be an oracle-guided selection exercise rather than a usable reference-cost heuristic. The selected case identifiers and their recorded statuses are saved.

The comparator retains three cases uniformly without replacement from the same program's available cases. We compute its exact finite-record detection probability from the number of passing and failing buggy cases. Its expected additive pair work is three times the program's average observed per-case pair work. No Monte Carlo sample of test subsets is needed for either expectation. The deterministic cost-guided selection is compared with those exact uniform expectations for each task.

Both policies therefore retain seventy-eight cases across the twenty-six tasks, counting the three selected or expected-retained cases per task. They do not receive the same realized line-event budget. This is an equal-count policy comparison whose workload is an outcome. A separate optimization under a common event budget would answer another question and could allocate different numbers of cases to different programs. We do not silently reinterpret the current design as that budget-constrained optimum.

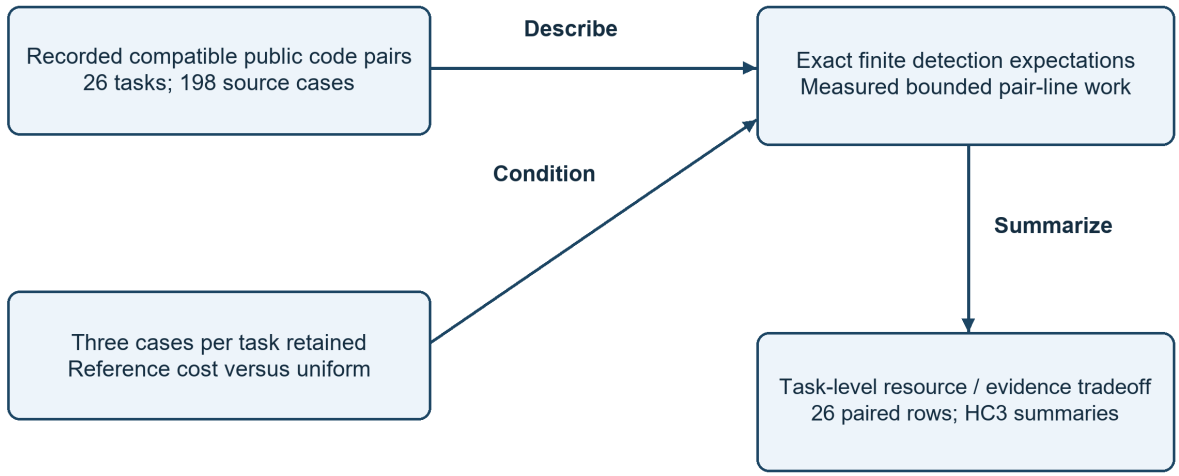


Figure 1. Reference-only test ranking, fixed three-case retention, and workload-detection outcomes.

Figure 1 shows the available information and the two downstream summaries. Reference trace counts determine cost-guided retention; buggy outcomes enter only evaluation. The same captured source pairs and oracles underlie both policies. The diagram includes no assistant-generation or fine-tuning stage. Direct preference optimization Rafailov et al., 2023 and human-feedback instruction learning Ouyang et al., 2022 provide downstream contexts in which evidence quality can matter, but the present mechanical detection outcomes are not certified human preferences or measured training benefits.

4. Exact expectations and task-level summaries

For every program, the first summary is the log of cost-guided pair work divided by uniform expected pair work. Averaging this log quantity gives an equal-task proportional comparison; exponentiating the average gives the geometric mean work ratio. The second summary is cost-guided detection, which is zero or one, minus the exact uniform three-case detection probability. Both are fitted with intercept-only OLS across twenty-six task rows, using HC3 standard errors and normal-reference intervals.

Robust covariance reasoning White, 1980 motivates reporting the variance estimator, while the dependence discussion in Cameron and Miller, 2015 cautions against treating all test cases as independent tasks. The regression N is twenty-six, not 198 or seventy-eight. These intervals are conditional summaries for hypothetical task heterogeneity around the captured cohort. They do not represent Monte Carlo error, since the uniform expectations are calculated exactly, and they do not establish the frequency of the effect in an independently sampled population of future software problems.

We also report total recorded work because it answers a different aggregation question. Equal-task log averaging gives each program one proportional contribution, while summing line events lets programs with large absolute traces dominate. Neither summary should be relabeled as the other. Scoring and estimation principles Gneiting and Raftery, 2007 favor declaring that target before interpreting a performance number. Cost-sensitive learning Elkan, 2001 would additionally require consequences for missed bugs and resource use before deciding whether the observed tradeoff is acceptable.

No noninferiority margin is specified for detection. We therefore cannot declare the heuristic harmless simply because it saves recorded work or because a particular confidence interval might include zero in another sample. The model-selection concerns described by Varma and Simon, 2006 also apply to searching among many ranking features. Here the one declared ranking uses reference lines only. The

analysis does not choose the most favorable heuristic from an unreported set of cost predictors or retention budgets.

5. Work decreases, but so does detection

The uniform three-case policy has equal-task mean detection probability 0.9265123. The cost-guided policy detects the buggy implementation in twenty of twenty-six tasks, giving 0.7692308. Their mean difference is negative 0.1572815, equivalent to about 15.7 percentage points. The standard error is 0.0640839 and the 95% interval extends from negative 0.2828837 to negative 0.0316793. This is a measurable loss of retained detection evidence under the declared comparison.

The mean log work ratio is negative 1.1673025, with standard error 0.3520949 and interval from negative 1.8573958 to negative 0.4772091. Exponentiating the point estimate gives geometric mean ratio 0.3112053. That ratio describes the task-level proportional summary. The total cost-guided work is 326,749 line events, compared with a uniform expectation of 429,530.7101 events. Fractional expected work is legitimate because it averages over possible uniform subsets, while the deterministic retained set has an integer event count.

Table 1. Task-level workload and detection contrasts for reference-cost selection

Model	Term	Estimate (SE)	95% CI	N
M1	Mean outcome/contrast	-1.1673 (0.3521)	[-1.8574, -0.4772]	26
M1: fit	R-squared = 0.0000	-	-	26
M2	Mean outcome/contrast	-0.1573 (0.0641)	[-0.2829, -0.0317]	26
M2: fit	R-squared = 0.0000	-	-	26

Model labels: M1 = Reference-cost-selected versus uniform log pair-work ratio; M2 = Reference-cost-selected minus uniform detection probability.

Table 1 reports two intercept-only OLS models, HC3 standard errors, normal-reference intervals, N26, no controls, and no fixed effects. M1 estimates the mean log pair-work ratio; M2 estimates the mean detection difference. Zero R-squared is a property of the mean-only specification. Work counts source-line trace events for both candidates, including bounded operational failures, and does not measure wall-clock speed. Detection uses the same captured expected outputs and eligibility cohort for both policies.

The task direction counts add detail. Cost guidance has lower recorded pair work than the uniform expectation in twenty-three tasks and higher work in two; one is equal. Detection is better in two tasks, worse in six, and equal in eighteen. Thus a reference-cost ranking does not guarantee a lower buggy-execution cost, and its detection loss is concentrated rather than uniform across the cohort. The saved task rows identify those cases without filtering them from the aggregate result.

The geometric work ratio and total-work comparison differ markedly because they weight the cohort differently. A large proportional reduction in a small task contributes to the log average but may save few events in the total. A bounded long-running buggy case can dominate total work even if most programs are cheap. We do not remove such operational failures as inconvenient outliers. Their presence is part of the declared bounded audit, and the limit is retained in the interpretation of every workload figure.

6. Why cheap reference cases are not sufficient evidence

The finding is consistent with a selection mismatch. Reference execution length measures how the designated corrected implementation traverses a case; it does not directly measure how strongly that case distinguishes the captured bug. A short path can omit relevant boundary behavior, while a cheap reference case can still cause an expensive buggy execution. The analysis does not identify which structural feature explains each loss. An influence examination following Cook, 1977 could inspect dominant tasks, but no unreported diagnostic is used to delete them or redefine the selected cases.

The comparison also omits the cost of obtaining reference profiles. In the source audit, all relevant cases had already been executed. A new workflow would need those measurements or estimates before applying the heuristic. That setup expense could exceed the saving from retaining three cases later, particularly for a one-off task. The study therefore reports conditional selection-stage work, not a net end-to-end efficiency gain. Production workflows with repeated candidate audits might amortize profiling differently, which would require its own measurement.

Coding benchmarks vary in how such evidence is organized. MBPP Austin et al., 2021, Codex-associated code evaluation Chen et al., 2021, and EvalPlus Liu et al., 2023 supply different task and test constructions. LiveCodeBench Jain et al., 2024 introduces another evaluation setting. Their cases are not included in the current line-event totals. The observed tradeoff should motivate testing a selection policy in its intended benchmark, rather than assuming that a favorable or unfavorable result transports automatically across different problem interfaces and test suites.

Repository agents make the distinction still broader. SWE-agent Yang et al., 2024 and Agentless Xia et al., 2024 operate in issue-resolution settings that differ from these short pure functions. CodeRL Le et al., 2022, DeepSeek-Coder Guo et al., 2024, and InCoder Fried et al., 2022 provide other coding-model contexts. No model from those works is run here. Their relevance is to define possible downstream users of execution evidence, while avoiding an unsupported claim that our simple retention heuristic explains their reported performance or should replace their evaluation procedures.

7. Auditing a resource tradeoff

Structured evaluation principles Roberts et al., 2017 favor keeping source overlap and cohort restrictions visible. This study reuses the same bounded program execution evidence as the uniform-retention analysis, but changes the policy from varying a random case count to selecting a fixed count by reference cost. It is not an independent new benchmark replication. The contextual interpretation urged by Wasserstein and Lazar, 2016 also favors the paired workload and detection magnitudes over a single label of efficient or inefficient.

The retained records connect selected case indices to source-case counts, both implementations, event totals, execution statuses, uniform expectations, and task contrasts. Such construction details belong in dataset documentation Gebru et al., 2021. The tracing and production-failure topic in Yan, 2026 supplies a useful comparison of evidential context: a cheap execution summary is not automatically a diagnosis of the defect that matters. No microservice tracing system is evaluated here. Undocumented selection criteria can still become technical debt Sculley et al., 2015 because they alter which failures an evaluation signal is able to reveal.

For this finite cohort, reference-cost selection reduced recorded audit work while sacrificing bug detection. The result does not reject all cost-aware testing, and it does not endorse uniform sampling as universally optimal. It shows why a resource heuristic should be evaluated against the evidence objective

it is meant to preserve. Fewer execution events are one outcome; retaining useful distinctions between candidate programs is another. Both are necessary to assess this three-test policy honestly.

References

1. Le, Hung; Wang, Yue; Gotmare, Akhilesh Deepak; Savarese, Silvio; Hoi, Steven C. H.. (2022). CodeRL: Mastering Code Generation through Pretrained Models and Deep Reinforcement Learning. arXiv:2207.01780. https://arxiv.org/abs/2207.01780.
2. Ouyang, Long; Wu, Jeff; Jiang, Xu; Almeida, Diogo; Wainwright, Carroll L.; Mishkin, Pamela; Zhang, Chong; Agarwal, Sandhini; Slama, Katarina; Ray, Alex; Schulman, John; Hilton, Jacob; Kelton, Fraser; Miller, Luke; Simens, Maddie; Askell, Amanda; Welinder, Peter; Christiano, Paul; Leike, Jan; Lowe, Ryan. (2022). Training language models to follow instructions with human feedback. arXiv:2203.02155. https://arxiv.org/abs/2203.02155.
3. Guo, Daya; Zhu, Qihao; Yang, Dejian; Xie, Zhenda; Dong, Kai; Zhang, Wentao; Chen, Guanting; Bi, Xiao; Wu, Y.; Li, Y. K.; Luo, Fuli; Xiong, Yingfei; Liang, Wenfeng. (2024). DeepSeek-Coder: When the Large Language Model Meets Programming -- The Rise of Code Intelligence. arXiv:2401.14196. https://arxiv.org/abs/2401.14196.
4. Cook, R. Dennis. (1977). Detection of Influential Observation in Linear Regression. Technometrics. https://doi.org/10.1080/00401706.1977.10489493.
5. Varma, Sudhir and Richard Simon. “Bias in error estimation when using cross-validation for model selection.” *BMC Bioinformatics*, vol. 7, no. 1, 2006. https://doi.org/10.1186/1471-2105-7-91.
6. Jain, Naman; Han, King; Gu, Alex; Li, Wen-Ding; Yan, Fanjia; Zhang, Tianjun; Wang, Sida; Solar-Lezama, Armando; Sen, Koushik; Stoica, Ion. (2024). LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code. arXiv:2403.07974. https://arxiv.org/abs/2403.07974.
7. Mytkowicz, Todd; Diwan, Amer; Hauswirth, Matthias; Sweeney, Peter F.. (2009). Producing wrong data without doing anything obviously wrong!. Proceedings of the 14th international conference on Architectural support for programming languages and operating systems. https://doi.org/10.1145/1508244.1508275.
8. Cameron, A. Colin, and Douglas L. Miller. “A Practitioner’s Guide to Cluster-Robust Inference.” *Journal of Human Resources*, vol. 50, no. 2, 2015, pp. 317-372. https://doi.org/10.3368/jhr.50.2.317.
9. Barr, Earl T.; Harman, Mark; McMinn, Phil; Shahbaz, Muzammil; Yoo, Shin. (2015). The Oracle Problem in Software Testing: A Survey. IEEE Transactions on Software Engineering. https://doi.org/10.1109/tse.2014.2372785.
10. Sculley, D., et al. "Hidden Technical Debt in Machine Learning Systems." *Advances in Neural Information Processing Systems*, vol. 28, 2015. https://papers.nips.cc/paper/2015/hash/86df7dcfd896fcaf2674f757a2463eba-Abstract.html
11. Gneiting, Tilmann; Raftery, Adrian E. (2007). Strictly Proper Scoring Rules, Prediction, and Estimation. Journal of the American Statistical Association. https://doi.org/10.1198/016214506000001437.
12. White, Halbert. “A Heteroskedasticity-Consistent Covariance Matrix Estimator and a Direct Test for Heteroskedasticity.” *Econometrica*, vol. 48, no. 4, 1980, pp. 817-838. https://doi.org/10.2307/1912934.
13. Gebru, Timnit, et al. "Datasheets for datasets" *Communications of the ACM*, 2021. https://doi.org/10.1145/3458723.
14. Rafailov, Rafael; Sharma, Archit; Mitchell, Eric; Ermon, Stefano; Manning, Christopher D.; Finn, Chelsea. (2023). Direct Preference Optimization: Your Language Model is Secretly a Reward Model. arXiv:2305.18290. https://arxiv.org/abs/2305.18290.
15. Fried, Daniel; Aghajanyan, Armen; Lin, Jessy; Wang, Sida; Wallace, Eric; Shi, Freda; Zhong, Ruiqi; Yih, Wen-tau; Zettlemoyer, Luke; Lewis, Mike. (2022). InCoder: A Generative Model for Code Infilling and Synthesis. arXiv:2204.05999. https://arxiv.org/abs/2204.05999.
16. Tao, Hongmo; Kamata, Hiroyuki. (2026). Teaching Coding Assistants What to Change: Fine-Tuning from Pairwise Preferences. Proceedings of the Second Workshop for REsearch on Agent Language Models (REALM 2026), 1363-1374. https://doi.org/10.18653/2026.realm-1.95.
17. Yan, Biyun. (2026). From Observability to Causality: Detecting Production Failures in Microservices with Distributed Tracing. 2026 9th International Conference on Computer Information Science and Application Technology (CISAT), 1522-1525. https://doi.org/10.1109/CISAT71030.2026.11682113.
18. Austin, Jacob; Odena, Augustus; Nye, Maxwell; Bosma, Maarten; Michalewski, Henryk; Dohan, David; Jiang, Ellen; Cai, Carrie; Terry, Michael; Le, Quoc; Sutton, Charles. (2021). Program Synthesis with Large Language Models. arXiv:2108.07732. https://arxiv.org/abs/2108.07732.
19. Kalibera, Tomas; Jones, Richard. (2013). Rigorous benchmarking in reasonable time. Proceedings of the 2013 international symposium on memory management. https://doi.org/10.1145/2464157.2464160.

20. Wasserstein, Ronald L.; Lazar, Nicole A.. (2016). The ASA Statement on p -Values: Context, Process, and Purpose. The American Statistician. https://doi.org/10.1080/00031305.2016.1154108.

21. Roberts, David R., et al. “Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure.” *Ecography*, vol. 40, no. 8, 2017, pp. 913-929. https://doi.org/10.1111/ecog.02881.

22. Liu, Jiawei; Xia, Chunqiu Steven; Wang, Yuyao; Zhang, Lingming. (2023). Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation. arXiv:2305.01210. https://arxiv.org/abs/2305.01210.

23. Chen, Mark; Tworek, Jerry; Jun, Heewoo; Yuan, Qiming; Pinto, Henrique Ponde de Oliveira; Kaplan, Jared; Edwards, Harri; Burda, Yuri; Joseph, Nicholas; Brockman, Greg; Ray, Alex; Puri, Raul; Krueger, Gretchen; Petrov, Michael; Khlaaf, Heidy; Sastry, Girish; Mishkin, Pamela; Chan, Brooke; Gray, Scott; Ryder, Nick; Pavlov, Mikhail; Power, Alethea; Kaiser, Lukasz; Bavarian, Mohammad; Winter, Clemens; Tillet, Philippe; Such, Felipe Petroski; Cummings, Dave; Plappert, Matthias; Chantzis, Fotios; Barnes, Elizabeth; Herbert-Voss, Ariel; Guss, William Hebgen; Nichol, Alex; Paino, Alex; Tezak, Nikolas; Tang, Jie; Babuschkin, Igor; Balaji, Suchir; Jain, Shantanu; Saunders, William; Hesse, Christopher; Carr, Andrew N.; Leike, Jan; Achiam, Josh; Misra, Vedant; Morikawa, Evan; Radford, Alec; Knight, Matthew; Brundage, Miles; Murati, Mira; Mayer, Katie; Welinder, Peter; McGrew, Bob; Amodei, Dario; McCandlish, Sam; Sutskever, Ilya; Zaremba, Wojciech. (2021). Evaluating Large Language Models Trained on Code. arXiv:2107.03374. https://arxiv.org/abs/2107.03374.

24. Lin, Derrick; Koppel, James; Chen, Angela; Solar-Lezama, Armando. (2017). QuixBugs: a multi-lingual program repair benchmark set based on the quixey challenge. Proceedings Companion of the 2017 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity. https://doi.org/10.1145/3135932.3135941.

25. Xia, Chunqiu Steven; Deng, Yinlin; Dunn, Soren; Zhang, Lingming. (2024). Agentless: Demystifying LLM-based Software Engineering Agents. arXiv:2407.01489. https://arxiv.org/abs/2407.01489.

26. Yang, John; Jimenez, Carlos E.; Wettig, Alexander; Lieret, Kilian; Yao, Shunyu; Narasimhan, Karthik; Press, Ofir. (2024). SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. arXiv:2405.15793. https://arxiv.org/abs/2405.15793.

27. Elkan, Charles. “The Foundations of Cost-Sensitive Learning.” *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, 2001, pp. 973-978. https://cseweb.ucsd.edu/~elkan/rescale.pdf.

28. McCabe, T.J.. (1976). A Complexity Measure. IEEE Transactions on Software Engineering. https://doi.org/10.1109/tse.1976.233837.

29. Python Software Foundation. (2026). Python 3.12 documentation: sys.settrace. Documentation snapshot captured 7 November 2026. https://docs.python.org/3.12/library/sys.html#sys.settrace.

30. Rothermel, G.; Untch, R.H.; Chengyun Chu; Harrold, M.J.. (2001). Prioritizing test cases for regression testing. IEEE Transactions on Software Engineering. https://doi.org/10.1109/32.962562.