



International Journal of Artificial Intelligence and Machine Learning

Publisher's Home Page: <https://www.svedbergopen.com/>



Research Paper

Open Access

History-Based Test Case Prioritization in Regression Testing: A Systematic Review and Analysis

Bharti Bhattad¹, Sachin Patel²

¹Associate Professor, Acropolis Institute of Technology and Research

²Professor, Sage University

Corresponding Author : bhartibhattad118@gmail.com

Abstract

Regression testing is one of the most resource-intensive activities in the software maintenance lifecycle, and organizations rarely have the time or budget to re-execute an entire test suite after every change. Test case prioritization (TCP) addresses this constraint by reordering test cases so that the tests most likely to reveal faults are executed earliest, allowing defects to surface sooner and giving developers more time to react. Among the many families of prioritization strategies that have been proposed, history-based approaches are distinctive in that they exploit the recorded outcomes of previous test executions — pass/fail results, execution time, and prior fault-detection behaviour — rather than relying solely on static code coverage or subjective requirement weighting. This paper surveys the conceptual foundations of test case prioritization, presents a classification of the major families of prioritization techniques, and provides a focused review of history-based methods and the empirical evidence supporting them. The paper also discusses the Average Percentage of Faults Detected (APFD) metric that is commonly used to evaluate prioritization effectiveness, identifies open research gaps such as the scarcity of source-code-independent techniques and the difficulty of validating methods across heterogeneous industrial test suites, and outlines directions for future work, including hybrid history-and-coverage strategies and the application of machine learning to execution-history data.

1. Introduction

Software testing consumes a substantial share of the effort and budget of any development project, and its purpose is to expose defects before they reach production. As a system evolves through successive releases, previously written test cases are re-run against the modified code to confirm that new changes have not broken existing behaviour — a practice known as regression testing. Because test suites tend to grow monotonically over the life of a project while the time available to run them does not, teams are frequently forced to test under time pressure, and it is common for only a fraction of the available test cases to be executed before a release deadline.

When exhaustive execution of a test suite is not feasible, three broad strategies are available: reduce the suite by removing redundant tests (test suite minimization), select only the tests relevant to a particular change (regression test selection), or reorder the existing tests so that the most valuable ones run first (test case prioritization). Prioritization is attractive because, unlike minimization or selection, it does not discard any test cases; it simply changes the order of execution so that if testing is interrupted, the tests that were most likely to reveal a defect have already been run. This property makes prioritization particularly well suited to continuous integration pipelines, where builds are triggered frequently and testing time is tightly bounded.

A wide variety of criteria have been proposed to decide which tests should run first, ranging from structural code coverage to customer-assigned requirement priority. This paper focuses in particular on history-based test case prioritization, a family of techniques that ranks test cases using data recorded from earlier executions of the same test suite — for example, whether a test previously failed, how long ago it last detected a fault, or how its execution time and outcome have varied across releases. Because these techniques rely on execution logs rather than on access to source code, they are attractive in settings such as outsourced or black-box testing, where the tester may not have visibility into the implementation.

The remainder of this paper is organized as follows. Section 2 provides background on regression testing and motivates the need for prioritization. Section 3 defines the test case prioritization problem and its objectives. Section 4 classifies the major families of prioritization techniques that have been studied in the literature. Section 5 reviews history-based prioritization in depth. Section 6 discusses the evaluation metrics used to compare techniques, principally the Average Percentage of Faults Detected. Section 7 discusses open challenges and research gaps, and Section 8 concludes the paper with directions for future work.

2. Background

2.1 Software Testing and the Cost of Regression

Testing is widely regarded as a necessary phase of the software development lifecycle because it is one of the few activities that can provide direct evidence about the correctness and reliability of a program. However, testing is also expensive: it requires designing test cases, executing them, and analyzing their results, and this cost recurs every time the software changes. Regression testing in particular has been estimated to consume a large share of the total cost of software maintenance, because a modification to even a small part of a system can, in principle, affect the behaviour of any other part that depends on it, directly or indirectly.

Rothermel and Harrold characterized regression testing as the selective re-testing of a modified program to provide confidence that no new faults have been introduced into previously working functionality and that the program still satisfies its specified requirements. As component-based and service-oriented architectures have become more common, this task has grown more difficult, because a change to a third-party component may affect the behaviour of a system whose internals are not fully visible to the tester responsible for regression testing.

2.2 Approaches to Coping with Growing Test Suites

Three complementary strategies are typically used to make regression testing tractable as test suites grow: test suite minimization, which removes test cases that have become redundant; regression test selection, which identifies only the subset of tests relevant to a given change; and test case prioritization, which reorders the full suite without discarding any tests. Minimization and selection both carry the risk of discarding a test case that would later have revealed a fault, since redundancy and relevance are difficult to determine with certainty. Prioritization avoids this risk because every test case remains in the suite; it is simply executed later rather than not at all. For this reason, several empirical studies have identified prioritization as one of the more dependable techniques for improving the efficiency of regression testing without sacrificing fault-detection capability.

3. Test Case Prioritization: Concepts and Objectives

3.1 Problem Definition

Formally, test case prioritization can be defined as the problem of finding an ordering of the test cases in a suite T that maximizes some award function across all orderings of T . In practice, the award function is usually related to the rate at which faults are detected: an ordering is preferred if it tends to expose defects earlier rather than later in the execution sequence, even when the total number of faults ultimately detected is the same regardless of order.

3.2 Objectives of Prioritization

Several objectives have been identified in the literature for why a tester might want to prioritize test cases. These include increasing the rate of fault detection so that defects are found earlier in the regression cycle; increasing the rate at which high-risk or high-severity faults specifically are detected; increasing confidence that recently changed code has been exercised early; increasing the rate at which code coverage targets are met; and providing earlier feedback about the overall reliability of the system under test. Because an interrupted or time-boxed test run only completes a prefix of the ordered suite, an ordering that front-loads valuable tests directly improves the outcome of that interrupted run.

3.3 Classification of Test Cases across Versions

An early and influential contribution to this area came from Leung and White, who proposed a classification scheme distinguishing test cases according to their relationship between two successive program versions. Reusable test cases exercise only the parts of the program that are unchanged and therefore need not be re-executed immediately, though they may be retained for future regression cycles. Retestable test cases exercise

parts of the program that were modified and must be re-run. Obsolete test cases no longer correspond to correct behaviour, either because the specification changed, because the code they were designed to exercise no longer exists, or because they no longer contribute new structural coverage. Two further categories, new-structural and new-specification test cases, describe tests that must be newly created to cover modified code constructs or modified specifications, respectively. This classification underlies much of the later work on both test selection and prioritization, since it distinguishes tests whose historical behaviour remains informative from those whose relevance has changed.

4. Classification of Prioritization Techniques

The literature on test case prioritization can be organized according to the type of information a technique relies on to rank test cases. This section summarizes the principal families before Section 5 examines history-based techniques in more detail.

4.1 Coverage-Based Techniques

Coverage-based techniques prioritize test cases according to the amount of code they exercise, measured at the granularity of statements, branches, basic blocks, functions, or conditions. Rothermel et al. proposed and empirically evaluated several coverage-based prioritization techniques, including total coverage prioritization, which orders tests by the total amount of code they cover, and additional coverage prioritization, which greedily selects the test that covers the most code not yet covered by tests already scheduled. Total techniques are simple to compute but can leave some code uncovered for a long time if many tests cover largely overlapping regions, while additional techniques are more responsive to redundancy but can become blind to code already covered early on. Because these two conventional strategies have complementary weaknesses, several later techniques have attempted to combine or refine them.

4.2 Requirement- and Risk-Based Techniques

Requirement-based techniques prioritize test cases according to properties of the requirements they verify rather than properties of the code. Srikanth et al. proposed mapping test cases to the requirements they test and ranking them using factors such as customer-assigned priority, requirement complexity, and requirement volatility. Krishnamoorthi and Sahaaya Arul Mary extended this idea with additional factors intended to make the ranking more comprehensive. A related family of risk-based techniques, exemplified by the RiteDAP approach of Stallbaum et al., prioritizes test cases according to the estimated risk associated with the requirements or model elements they exercise, typically derived from annotated UML activity diagrams. A recognized limitation of both families is that the underlying weights — customer priority, complexity, or risk — are often assigned subjectively and may not correlate reliably with actual fault-proneness.

4.3 Cost-Aware Techniques

Cost-cognizant prioritization techniques incorporate the cost of executing or maintaining a test case into the ranking, on the premise that a technique should not only maximize fault detection but should do so per unit of testing effort. Elbaum, Malishevsky, and Rothermel extended the APFD metric itself into a cost-cognizant variant (APFDc) to account for the fact that test cases and faults can vary substantially in cost and severity, and demonstrated that rankings which are optimal under the traditional metric are not necessarily optimal once cost is taken into account.

4.4 History-Based Techniques

History-based techniques use data recorded from previous executions of the test suite — pass/fail outcomes, execution duration, and the number of releases since a test last detected a fault — to guide prioritization. Because this information can be collected purely from execution logs, history-based techniques do not require access to source code, which makes them attractive in outsourced or black-box testing contexts. This family of techniques is discussed in detail in Section 5.

4.5 Distribution-Based and Clustering Techniques

Leon and Podgurski introduced distribution-based prioritization, which does not rely on coverage at all but instead clusters test cases according to the dissimilarity of their execution profiles in a multi-dimensional space. Clusters of highly similar profiles tend to indicate redundant test cases, isolated clusters may correspond to unusual execution conditions that are disproportionately likely to expose faults, and low-density regions of the profile space may correspond to rarely exercised behaviours. Leon and Podgurski also proposed a hybrid strategy that begins with coverage-based prioritization and switches to distribution-based prioritization once coverage has saturated, since they observed that repeated coverage maximization loses effectiveness once most of the code has already been covered.

4.6 Model-Based Techniques

Model-based prioritization techniques operate on an abstract model of the system, such as a UML activity diagram or a dependency graph, rather than on source code directly. Korel et al. proposed a selective prioritization approach that classifies test cases into high- and low-priority sets based on their relevance to model changes, and later developed more sophisticated heuristics based on dependence analysis of the model. Because these techniques reason about a model rather than the implementation, they can be applied earlier in development and can be more robust to superficial code-level changes that do not correspond to behavioural changes.

4.7 Human-in-the-Loop and Machine-Learning Techniques

A further family of techniques incorporates human judgment or machine learning directly into the ranking process. Tonella et al. combined case-based reasoning with a pairwise ranking algorithm (RankBoost) that learns a ranking function from pairwise priority judgments supplied by a human tester, seeded with an initial index derived from statement coverage and cyclomatic complexity. Yoo et al. extended this idea with the Interleaved Clusters Prioritization (ICP) technique, which combines clustering with the Analytic Hierarchy Process, a pairwise comparison method from operations research, so that a human tester need only make comparisons between clusters rather than between every pair of individual test cases. Interestingly, Yoo et al. found that ICP could outperform purely coverage-based prioritization even when the simulated human tester made incorrect pairwise judgments more than half the time, an effect they attributed to the benefit of clustering itself rather than to the accuracy of the human input.

5. History-Based Test Case Prioritization: Detailed Review

Because the present survey is particularly concerned with history-based methods, this section reviews the major contributions in more depth and highlights the assumptions and limitations of each.

5.1 Foundational Work

Kim and Porter are generally credited with an early and influential formulation of history-based prioritization, in which test cases are ranked using statistics computed from their execution history, such as how recently a test was last run and whether it detected a fault at that time. A central premise of their approach is that a test case which has not been exercised for several releases, or which detected a fault previously, is disproportionately likely to be valuable in the current regression cycle, and should therefore be scheduled earlier than a test case that has recently passed without incident.

5.2 Metric-Based and Fault-Proneness Approaches

Elbaum et al. examined prioritization techniques that compute a fault-proneness index from a small set of measurable software and test attributes, rather than from raw coverage counts, and used this index to rank test cases. This line of work sits between purely history-based and purely coverage-based prioritization, since the attributes used to compute the index can include both structural properties of the code and historical properties of the test case, such as its past fault-detection record.

5.3 Version-Aware and Cost-Sensitive Extensions

Later work extended history-based prioritization along two directions. The first direction incorporates awareness of which software version a historical result belongs to, on the premise that a test's fault-detection record from a distant past release is less predictive of its value in the current release than a more recent record. The second direction incorporates cost directly into the historical ranking, producing an ordering intended to maximize fault detection per unit of execution time rather than fault detection alone. Qu et al. proposed a related technique that groups previously executed test cases by the type of fault they revealed and uses this grouping, together with run-time information, to dynamically prioritize test cases in black-box regression testing, where the source code is not available to the tester.

5.4 REPiR and Information-Retrieval-Based Prioritization

A distinct strand of history-informed prioritization reframes the problem as one of information retrieval. Under this framing, differences between the artifacts (for example, the textual content of test cases or associated documentation) belonging to two successive program versions are treated as a query, and test cases are ranked by how strongly they relate to that query, using standard information-retrieval ranking techniques. This approach illustrates that "history" in test case prioritization need not be limited to pass/fail statistics; it can also include the recorded textual or structural artifacts associated with a test case across versions.

5.5 Comparative Findings

Across the studies summarized above, a recurring finding is that no single prioritization criterion dominates in every setting. History-based techniques tend to perform particularly well when a test suite is regression-tested repeatedly over many releases and when a meaningful fraction of faults recur in similar areas of the code, since

this is precisely the scenario in which past fault-detection behaviour is most predictive of future behaviour. They tend to be less advantageous immediately after a major redesign, when the execution history accumulated under the previous architecture may no longer be representative. This observation motivates hybrid strategies that combine coverage-based and history-based signals, using coverage information when historical data is sparse or stale and shifting weight toward historical data as it accumulates.

6. Evaluation Metrics

6.1 Average Percentage of Faults Detected (APFD)

The metric most widely used to compare prioritization techniques is the Average Percentage of Faults Detected, or APFD, introduced by Rothermel et al. APFD summarizes, as a single weighted value between 0 and 100, how quickly a given ordering of test cases detects the faults present in a program: an ordering that detects most faults very early in execution receives a value close to 100, whereas an ordering that defers most fault detection until late in execution receives a value close to 0. APFD is computed from the position, within the ordered test suite, of the first test case to expose each known fault, averaged and weighted by the total number of test cases and faults involved.

6.2 Cost-Cognizant and Alternative Metrics

Because APFD assumes that every test case has the same execution cost and every fault has the same severity, several extensions have been proposed. The cost-cognizant APFD_c metric of Elbaum et al. weights the contribution of each fault by its severity and each test case by its execution time, producing a measure of the rate of fault-severity detection per unit of testing effort rather than per test case executed. Other studies have proposed measuring prioritization effectiveness directly in terms of testing time saved, or in terms of the point at which a target level of code coverage is reached, particularly for techniques whose primary objective is not fault detection but coverage acceleration.

7. Research Gaps and Open Challenges

Despite the substantial body of work summarized above, several gaps remain. First, source-code-independent techniques, including history-based and requirement-based approaches, remain comparatively less studied and less consistently benchmarked than coverage-based techniques, even though they are precisely the techniques most applicable in outsourced or black-box testing contexts where source code is unavailable. Second, most empirical evaluations of prioritization techniques have used a relatively small set of open-source or academic benchmark programs, such as the Siemens suite or the Software-artifact Infrastructure Repository programs, and it remains unclear how well the reported results generalize to large, industrial test suites with different characteristics.

Third, there is a persistent gap between the effectiveness of heuristic prioritization techniques and the theoretical optimum achievable if the location of all faults were known in advance; existing techniques have not fully closed this gap, and it remains an open question whether static analysis of fault-proneness, sensitivity analysis, or learning-based approaches can narrow it further. Fourth, most history-based techniques implicitly assume that the test suite itself and its mapping onto the software under test remain relatively stable across releases; how best to prioritize newly added test cases, which by definition have no execution history, remains comparatively underexplored. Finally, as continuous integration pipelines have become the dominant deployment model in industry, there is growing interest in prioritization techniques that operate incrementally and can be updated cheaply after every commit, rather than techniques that assume a single, static prioritization computed at the start of a regression cycle.

8. Conclusion and Future Work

Test case prioritization is a well-established and practically important technique for coping with the growth of regression test suites under limited testing budgets. This paper has reviewed the conceptual foundations of the problem, summarized the principal families of prioritization techniques — coverage-based, requirement- and risk-based, cost-aware, history-based, distribution-based, model-based, and human-in-the-loop approaches — and examined history-based prioritization in particular depth, since it offers the practical advantage of not requiring access to source code.

The evidence reviewed suggests that history-based techniques are especially valuable in settings with long-lived, repeatedly executed test suites, but that their effectiveness can degrade when historical data is sparse,

stale, or collected under a substantially different version of the software. Future work in this area could usefully focus on hybrid techniques that combine historical execution data with lightweight structural or requirement-based signals, on machine-learning models trained to predict fault-detection likelihood directly from execution logs, and on empirical studies conducted on larger and more diverse industrial test suites so that the generalizability of existing findings can be established with greater confidence. As continuous integration and continuous delivery practices continue to shorten release cycles, the ability to prioritize test cases cheaply, incrementally, and without dependence on source code is likely to become an increasingly important research direction.

References

- [1] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Transactions on Software Engineering*, vol. 27, no. 10, pp. 929–948, 2001.
- [2] G. Rothermel and M. J. Harrold, "Analyzing regression test selection techniques," *IEEE Transactions on Software Engineering*, vol. 22, no. 8, pp. 529–551, 1996.
- [3] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Test case prioritization: A family of empirical studies," *IEEE Transactions on Software Engineering*, vol. 28, no. 2, pp. 159–182, 2002.
- [4] S. Elbaum, A. Malishevsky, and G. Rothermel, "Incorporating varying test costs and fault severities into test case prioritization," in *Proceedings of the 23rd International Conference on Software Engineering (ICSE)*, 2001, pp. 329–338.
- [5] H. K. N. Leung and L. White, "Insights into regression testing," in *Proceedings of the Conference on Software Maintenance*, 1989, pp. 60–69.
- [6] H. K. N. Leung and L. White, "A study of integration testing and software regression at the integration level," in *Proceedings of the Conference on Software Maintenance*, 1990, pp. 290–301.
- [7] J.-M. Kim and A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environments," in *Proceedings of the 24th International Conference on Software Engineering (ICSE)*, 2002, pp. 119–129.
- [8] A. Srikanth, L. Williams, and J. Osborne, "System test case prioritization of new and regression test cases," in *Proceedings of the International Symposium on Empirical Software Engineering*, 2005, pp. 64–73.
- [9] R. Krishnamoorthi and S. A. Sahaaya Arul Mary, "Factor oriented requirement coverage based system test case prioritization of new and regression test cases," *Information and Software Technology*, vol. 51, no. 4, pp. 799–808, 2009.
- [10] H. Stallbaum, A. Metzger, and K. Pohl, "An automated technique for risk-based test case generation and prioritization," in *Proceedings of the 3rd International Workshop on Automation of Software Test (AST)*, 2008, pp. 67–70.
- [11] M. Leon and A. Podgurski, "A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases," in *Proceedings of the 14th International Symposium on Software Reliability Engineering (ISSRE)*, 2003, pp. 442–453.
- [12] B. Korel, L. H. Tahat, and M. Harman, "Test prioritization using system models," in *Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM)*, 2005, pp. 559–568.
- [13] P. Tonella, P. Avesani, and A. Susi, "Using the case-based ranking methodology for test case prioritization," in *Proceedings of the 22nd IEEE International Conference on Software Maintenance (ICSM)*, 2006, pp. 123–133.
- [14] S. Yoo, M. Harman, P. Tonella, and A. Susi, "Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge," in *Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2009, pp. 201–212.
- [15] B. Qu, C. Nie, B. Xu, and X. Zhang, "Test case prioritization for black box testing," in *Proceedings of the 31st Annual International Computer Software and Applications Conference (COMPSAC)*, 2007, pp. 465–474.
- [16] Z. Li, M. Harman, and R. M. Hierons, "Search algorithms for regression test case prioritization," *IEEE Transactions on Software Engineering*, vol. 33, no. 4, pp. 225–237, 2007.
- [17] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: A survey," *Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 67–120, 2012.
- [18] H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empirical Software Engineering*, vol. 10, no. 4, pp. 405–435, 2005.
- [19] C. Catal and D. Mishra, "Test case prioritization: A systematic mapping study," *Software Quality Journal*, vol. 21, no. 3, pp. 445–478, 2013.

- [20] M. Khatibsyarbini, M. A. Isa, D. N. A. Jawawi, and R. Tumeng, "Test case prioritization approaches in regression testing: A systematic literature review," *Information and Software Technology*, vol. 93, pp. 74–93, 2018.
- [21] E. Engström, P. Runeson, and M. Skoglund, "A systematic review on regression test selection techniques," *Information and Software Technology*, vol. 52, no. 1, pp. 14–30, 2010.
- [22] L. Zhang, S. Hou, C. Guo, T. Xie, and H. Mei, "Time-aware test-case prioritization using integer linear programming," in *Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2009, pp. 213–224.
- [23] S. Elbaum, G. Rothermel, S. Kanduri, and A. G. Malishevsky, "Selecting a cost-effective test case prioritization technique," *Software Quality Journal*, vol. 12, no. 3, pp. 185–210, 2004.