

Original Article

Runtime-Guided Change-Impact Analysis for Explainable Regression Test Selection in Evolving Software Platforms: A Corrected Benchmark and Defects4J Pilot

***Devinder Tokas**
Independent Researcher, USA.

Abstract:

Context: Runtime-guided regression test selection can connect changed code to previously observed test behavior, but evaluation granularity may determine whether competing methods can be distinguished. Objective: We examine an explainable method, TRACE-RTS, and test whether a class-level real-fault pilot can evaluate its function-level change-impact contribution. Method: TRACE-RTS combines runtime footprints, changed functions, bounded dependency propagation, history, freshness, execution cost, and per-test explanations. A corrected deterministic benchmark uses 40 generated subjects, 20 releases per subject, five cost budgets, sparse faults, and independently degraded selection evidence. A pre-specified Defects4J 3.0.1 pilot evaluates nine Lang faults using test classes, production-class coverage, strict measured-cost ceilings, and 1,000 Random orders per fault-budget pair. Results: At a 30% synthetic budget, TRACE-RTS detected faults in 77.63% of 800 releases, versus 82.13% for Direct-Rank. Direct-Rank had significantly higher fault-revealing-test recall, while TRACE-RTS attained 99.74% of the impact weight reachable by the full suite and reached a first fault after 6.69% of full-suite cost. In Defects4J, TRACE-RTS, Direct-Rank, and Direct-Greedy produced identical results at every budget. Class-impact coverage was zero below 30% and saturated at 100% thereafter. Random was descriptively higher at four of five budgets, but nine-fault bootstrap intervals included zero. Conclusion: The evidence supports a tradeoff, not TRACE-RTS superiority. More importantly, the pilot demonstrates that relevant-tests-only, class-level evidence is too coarse to evaluate a function-level RTS contribution. Method-level, all-tests, multi-project evaluation is required.

Keywords:

Regression Testing, Regression Test Selection, Change-Impact Analysis, Runtime Coverage, Defects4j, Negative Results, Reproducibility, Explainable Software Engineering.

Article History:

Received: 07.12.2024

Revised: 10.01.2025

Accepted: 20.01.2025

Published: 29.01.2025

1. Introduction

Regression test selection (RTS) reduces validation cost by selecting tests likely to exercise behavior affected by a software change. Established research evaluates RTS through inclusiveness, precision, cost, coverage, and fault detection [1-4]. Practical systems must also address evidence freshness, dependency maintenance, flaky outcomes, distributed execution, and the operational cost of explaining why a test was selected [5,6].

Dynamic coverage provides concrete evidence that a test previously exercised an entity, but direct overlap with changed code can miss propagated effects. Static dependencies widen the impact set but may over-select. Historical failures can improve



prioritization without proving relevance to the present change. These signals are complementary, but combining them creates an evaluation challenge: a method-level contribution cannot be assessed reliably with evidence that has already been aggregated to classes.

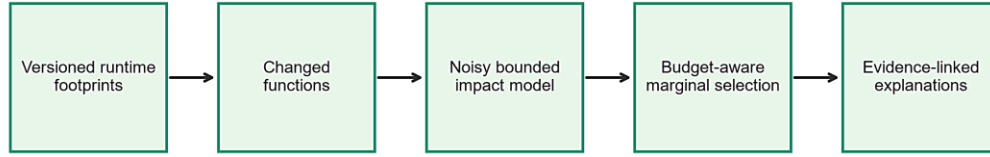


Figure 1. TRACE-RTS Workflow from Changed Functions and Runtime Evidence to Budget-Aware Selection and Explanations

This Short Communication presents TRACE-RTS, an explainable runtime-guided selection method, and reports two deliberately bounded studies. The first is a corrected synthetic benchmark designed to avoid several common sources of optimistic bias. The second is a pre-specified nine-fault Defects4J pilot whose negative result identifies an invalid evaluation granularity. The goal is not to claim superiority. It is to provide a reproducible method, report the observed tradeoffs, and show why coarse evidence can prevent an empirical study from testing its intended construct.

The contribution is threefold:

- An auditable selection method combining runtime and change evidence under a measured-cost budget;
- A corrected benchmark that separates true fault behavior from the noisy evidence available to selection; and
- Empirical evidence that a relevant-tests-only, class-level pilot collapses distinct function-level rts strategies.

2. Trace-RTS

Let F be program functions, T available tests, $C(t)$ the historical function footprint of test t , Δ the changed functions, G a dependency graph, $c(t)$ measured test cost, and B the execution budget. Each changed function receives severity $s(f)$. TRACE-RTS assigns impact weight:

$$w(f) = \max \text{ over changed roots } r \text{ of } s(r) * a[d(r,f)],$$

Where $d(r,f)$ is bounded dependency distance and the reported implementation uses $a[0] = 1.0$, $a[1] = 0.58$, and $a[2] = 0.27$.

For a partial selection S , a candidate test receives marginal impacted weight for functions in $C(t)$ not already represented by S . The executable proof of concept combines this marginal weight with decayed failure history and footprint freshness, divided by the square root of measured execution cost. The highest-scoring test that fits the remaining strict budget is selected iteratively.

For each selected test, the method records its cost, selection rank, remaining budget, changed or propagated functions contributing marginal value, dependency distance, impact weight, history and freshness contributions, and evidence status. The explanation is derived from the same contributions used for selection rather than from a separate post-hoc model.

TRACE-RTS is not formally safe. Historical coverage can omit unobserved behavior, bounded propagation can miss longer effects, and unresolved function identity can invalidate overlap. It is intended as a budget-constrained selection or prioritization method with explicit evidence limitations, not as a universal replacement for safety-preserving RTS.

3. Corrected Synthetic Benchmark

3.1. Design

The deterministic benchmark generates 40 subjects with 220-420 functions, 90-180 tests, 8-16 modules, directed dependencies, runtime footprints, and lognormally distributed costs. Each subject contributes 20 evolving releases evaluated at 10%, 20%, 30%, 40%, and 50% of full-suite cost.

To reduce circularity, the generator maintains separate true and observed dependency graphs and coverage. True evidence determines fault behavior. Selection receives independently degraded evidence with 15% dependency-edge dropout, 5% added edges, 10% footprint-function dropout, and 2% added functions. Faults are sparse: approximately 1.38% of tests reveal a fault on average. History is updated only from executed failing tests, freshness is reset only for executed tests, and nondetection remains missing in first-fault analysis.

We compare All, Random, History, Direct-Rank, Direct-Greedy, TRACE-RTS, and an unattainable Oracle-Greedy upper bound. Direct-Greedy separates marginal set coverage from dependency propagation. Comparisons use subject-level paired bootstrap intervals, Wilcoxon signed-rank tests for continuous measures, exact McNemar tests for binary detection, rank-biserial effects, and Holm correction.

3.2. Results

Table 1. Corrected Synthetic Results at A 30% Measured-Cost Budget

Algorithm	Fault detection	Fault-test recall	Attainable impact coverage	First-fault/full-suite cost
All	100.00%	100.00%	100.00%	37.51%
Random	50.00%	32.46%	92.96%	14.29%
History	71.25%	54.75%	97.91%	9.36%
Direct-Rank	**82.13%**	**70.98%**	98.14%	7.90%
Direct-Greedy	74.63%	55.00%	98.93%	8.61%
TRACE-RTS	77.63%	58.27%	**99.74%**	**6.69%**
Oracle-Greedy	80.00%	60.98%	approximately 100%	6.11%

TRACE-RTS was not universally superior. Direct-Rank's detection point estimate was 4.50 percentage points higher at the 30% budget, although the adjusted detection comparison was not significant. Direct-Rank's 12.71-point fault-test-recall advantage was significant. TRACE-RTS covered 1.60 points more attainable impact weight than Direct-Rank and 0.81 points more than Direct-Greedy; both differences were significant. TRACE-RTS reached a first fault 1.84 points earlier than Direct-Greedy, but its 1.18-point advantage over Direct-Rank was not significant after correction.

The result exposes a meaningful objective tradeoff. Direct ranking retained more fault-revealing tests, whereas marginal propagated coverage represented a broader impacted-function set and found the first failure earlier in some comparisons. TRACE-RTS also required approximately 0.039 seconds of selection time per release at 30%, compared with approximately 0.0005 seconds for Direct-Rank.

4. Defects4J Feasibility Pilot

4.1. Pre-Specified Protocol

The public pilot pinned Defects4J 3.0.1 and active Lang faults 1 and 3-10. Every buggy version reproduced its documented triggering failure; no fault was excluded. Developer test classes were the selection unit, tests.relevant defined the common candidate universe, production-class coverage supplied runtime evidence, and reverse internal class dependencies were extracted with JDK jdeps. Trigger labels were withheld until scoring.

Budgets were 10%-50% of the measured relevant-suite cost. Strict ceilings allowed an empty plan and never forced a test over budget. Random used 1,000 deterministic orders for each fault-budget pair. The pilot compared All-Relevant, Random, Direct-Rank, Direct-Greedy, and TRACE-RTS.

4.2. Results

Table 2. Defects4J Lang Pilot Fault Detection

Nominal budget	Random	TRACE/direct methods	Random realized cost	TRACE/direct realized cost
10%	5.40%	0.00%	4.01%	3.93%
20%	20.90%	22.22%	9.42%	9.75%

30%	25.36%	11.11%	20.93%	21.47%
40%	29.20%	22.22%	24.00%	24.49%
50%	45.87%	22.22%	37.15%	33.61%

TRACE-RTS, Direct-Rank, and Direct-Greedy selected identically at every budget. Their attainable-normalized class-impact coverage was zero below 30% and 100% from 30% onward. The relevant-test universe averaged only 5.56 classes per fault. Test classes were therefore too few and their production coverage too aggregated for propagation or marginal coverage to differentiate the methods.

Random was descriptively higher at four of five budgets, including 45.87% versus 22.22% detection at 50%. However, fault-level bootstrap intervals for TRACE-minus-Random detection included zero at every budget. Nine faults are insufficient for a stable superiority or inferiority conclusion.

Strict budgets also undershot their nominal values because test classes are indivisible. When class-impact coverage tied, the inverse-square-root cost term favored inexpensive classes and could omit a larger triggering class. Thus both granularity and cost normalization contributed to the observed behavior.

5. Implications

The pilot is useful precisely because it did not support the intended effectiveness claim. It completed checkout, failure reproduction, direct cost measurement, runtime-coverage collection, dependency extraction, evidence separation, selection, scoring, resumability, and raw-result retention. Yet the resulting evidence could not test a function-level contribution.

Three requirements follow for RTS evaluation:

- Match evidence granularity to the claimed mechanism. Function-level propagation requires per-test-method footprints and method-level changed entities. Class-level aggregation can erase the variation the method is designed to exploit.
- Use the full declared developer-test universe. A framework-curated relevant subset may be appropriate for other analyses but can remove realistic selection choices and create a small, stepwise search space.
- Report realized cost and indivisibility. Nominal budgets can materially differ from executable plans when selection units are coarse.

The next valid study should collect per-test-method coverage over all discoverable developer tests, identify changed methods, preserve full method descriptors, compare multiple projects and real faults, and reproduce recognized RTS tools where technically compatible. Exclusions must be categorical and retained rather than replaced with simulated baseline outcomes.

6. Threats to Validity

The synthetic generator structurally represents faults through function coverage and bounded downstream manifestations. It does not model configuration, concurrency, timing, environment, or complex state. Impact coverage aligns with the proposed objective and is not independent proof of fault effectiveness.

The Defects4J pilot covers nine faults from one project and uses classes rather than methods. Its purpose is diagnostic, not generalization. The Random comparison averages 1,000 orders within each fault, but only nine independent fault units remain. Hyperparameters were fixed for the reported run but were selected by the author. The generator, method, and baselines share one implementation, so common implementation defects remain possible despite deterministic artifacts and component baselines. Explanation records are faithful to the implemented score, but no human study establishes their usefulness. No production deployment or Microsoft system is evaluated or claimed.

7. Conclusion

TRACE-RTS combines runtime footprints, changed-function evidence, bounded propagation, history, freshness, cost, and faithful per-test explanations. The corrected synthetic benchmark shows a tradeoff: Direct-Rank retained more fault-revealing tests, while TRACE-RTS represented more attainable impact weight and sometimes reached a fault earlier.

The real-fault pilot provides a more important methodological result. Relevant-tests-only, class-level evidence made three distinct deterministic strategies identical and produced stepwise impact coverage. The study therefore supplies no evidence that TRACE-RTS improves real-fault detection. It demonstrates that evaluation granularity can prevent an RTS experiment from testing its claimed mechanism.

Reporting this null result avoids an unsupported effectiveness claim and provides a reproducible basis for a method-level, all-tests, multi-project follow-up.

Author Declarations

- Funding: No external funding was received for this research.
- Competing interests: The author declares no competing interests.
- Data and code availability: The complete executable artifact, generation parameters, raw synthetic observations, aggregate tables, Defects4J protocol, per-fault records, and analysis code accompany the submission. A persistent public repository URL and archival DOI must be inserted before submission.
- Author contribution: Devinder Tokas conceived the method, designed and implemented the studies, analyzed the results, and wrote the manuscript.
- Confidentiality and affiliation: This independent research uses only public and synthetic data. It contains no employer source code, telemetry, schemas, endpoints, application identities, or unpublished operational metrics. The views are the author's own.

References

- [1] G. Rothermel and M. J. Harrold, "Analyzing regression test selection techniques," *IEEE Transactions on Software Engineering* 22(8) (1996) 529-551. <https://doi.org/10.1109/32.536955>.
- [2] G. Rothermel and M. J. Harrold, "A safe, efficient regression test selection technique," *ACM Transactions on Software Engineering and Methodology* 6(2) (1997) 173-210. <https://doi.org/10.1145/248233.248262>.
- [3] G. Rothermel and M. J. Harrold, "Empirical studies of a safe regression test selection technique," *IEEE Transactions on Software Engineering* 24(6) (1998) 401-419. <https://doi.org/10.1109/32.689399>.
- [4] R. Kazmi, D.N.A. Jawawi, R. Mohamad, I. Ghani, "Effective regression test case selection: A systematic literature review," *ACM Computing Surveys* 50(2) (2017), Article 29. <https://doi.org/10.1145/3057269>.
- [5] H. Zhong, L. Zhang, S. Khurshid, "TestSage: Regression test selection for large-scale web service testing," *Proceedings of ICST* (2019) 430-440. <https://doi.org/10.1109/ICST.2019.00052>.
- [6] M. Machalica, A. Samylkin, M. Porth, S. Chandra, "Predictive test selection," *Proceedings of ICSE-SEIP* (2019) 91-100. <https://doi.org/10.1109/ICSE-SEIP.2019.00019>.
- [7] G. Zhang, L. Liu, Z. Chen, J. Wang, "Hybrid regression test selection by integrating file and method dependences," *Proceedings of ASE* (2024) 1557-1569. <https://doi.org/10.1145/3691620.3695525>.
- [8] R. Just, D. Jalali, M.D. Ernst, "Defects4J: A database of existing faults to enable controlled testing studies for Java programs," *Proceedings of ISSTA* (2014) 437-440. <https://doi.org/10.1145/2610384.2628055>.
- [9] S. Yoo, M. Harman, "Regression testing minimization, selection and prioritization: A survey," *Software Testing, Verification and Reliability* 22(2) (2012) 67-120. <https://doi.org/10.1002/stvr.430>.