Towards Just-In-Time Mutation Testing for Simulink Cyber-Physical System Models

Emiray Durmaz[*,†], Murat Kavak[†], Onur Kilincceker[†] and Serge Demeyer[†]

University of Antwerpen, Middelheimlaan 1, 2020 Antwerpen, Belgium
Flanders Make VZW, Oude Diestersebaan 133, 3920 Lommel, Belgium

Abstract

Mutation testing is a highly effective fault-based testing method, but it requires extensive time or significant computational power to execute for complicated systems. This necessity becomes even more apparent when applying mutation testing to cyber-physical design models (such as Simulink and Stateflow) that rely on a dedicated simulation environment, even though the number of mutants is less than in code-level mutation testing. In this paper, we propose a coverage-based approach to reduce mutation testing execution time, with a direction toward just-in-time mutation testing. Our experimental evaluation is conducted on 2 open-source Stateflow models and 1 industrial Simulink model from 2 different safety-critical domains (namely automotive and industrial controller). Results show that the coverage-based approach, compared with random sampling, provides better and more stable results for both Stateflow models; however, it yields similar results for the industrial Simulink model.

Keywords

mutation testing, Simulink, Stateflow, just-in-time testing, time-budgeted testing, cyber-physical systems, search-based software engineering

1. Introduction

Mutation testing is a fault-based software testing approach that employs small synthetic faults (mutant generation) in the system and checks whether existing tests can detect (kill) deviations from the original behavior (mutant execution) [1, 2, 3]. However, mutation testing places excessive demands on both computational and human resources because of the vast number of mutants that must be compiled and run against one or more test cases [4]. Several techniques address this scalability issue to reduce mutation testing costs [5, 6, 7]. However, many domains (such as safety-critical domains) where mutation testing provides advantages but poses scalability problems have not adopted or used these techniques.

Safety-critical domains, including automotive and aerospace, predominantly depend on model-driven engineering to address the complexity and ubiquity of safety-critical software. This dependence becomes even more pronounced in cyber-physical systems, where software must coordinate with physical components and processes while adhering to stringent safety, timing, and reliability requirements [8]. Applying mutation testing at the design (model) level for cyber-physical systems, since the system is modeled at a higher level of abstraction to manage complexity, can also reduce mutation testing costs by generating fewer mutants. However, model-driven engineering tools (such as Simulink[1], Stateflow[2], Ansys Scade[3]) rely on a design simulation environment, and each test case execution takes longer than code-level testing.

BENEVOL2026: Belgium-Netherlands Software Evolution Workshop 26–27 November, 2026, Madrid, Spain

[*] You can use this document as the template for preparing your publication. We recommend using the latest version of the ceurart style.

[*] Corresponding author.

[†] These authors contributed equally.

0009-0004-0697-3046 (E. Durmaz); 0009-0003-4760-8638 (M. Kavak); 0000-0001-5996-4398 (O. Kilincceker); 0000-0002-4463-2945 (S. Demeyer)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

[1] https://nl.mathworks.com/products/simulink.html, Last Accessed: 19.08.2026

[2] https://nl.mathworks.com/products/stateflow.html, Last Accessed: 19.08.2026

[3] https://ansys.synopsys.com/products/embedded-software/ansys-scade-suite, Last Accessed: 19.08.2026

The problem is particularly relevant for Simulink and Stateflow, where test execution is based on repeated model simulations. Previous work on MUT4SLX[4] [9, 10] illustrates how costly this process can become. In its evaluation on the Helicopter Control System, a complete mutation analysis took 8.14 hours, although mutant generation itself was fast [9]. This shows that the execution phase can become a major part of the overall mutation testing cost. In this work, we investigate ways to reduce this execution time. Just-in-time mutation testing refers to obtaining mutation testing results within a predefined time budget, rather than waiting for the complete analysis to finish. We therefore evaluate random mutant selection and coverage-based filtering under different time budgets. The main goal is to reduce the amount of mutation testing that needs to be executed while keeping the resulting mutation score as close as possible to the score obtained from the complete analysis. Reducing this cost is also relevant in software maintenance and evolution. When a Simulink model changes, regression tests may need to be executed again to validate the modified model. In settings such as continuous integration or release validation, repeatedly performing a complete mutation analysis may therefore become impractical when the available validation time is limited.

To address those problems, we present our research questions (RQs) as follows;

- RQ1 (Method): How can a just-in-time mutation testing approach be materialized without running each mutant–test pair under a hard time budget?
- RQ2 (Results): What could be the results for random mutant selection (as a baseline) and coverage-based filtering towards just-in-time mutation testing under a hard time budget?
- RQ3 (Challenges): What are the expected benefits and open challenges on various model types (Simulink, Stateflow and their combinations)?

Based on all the previous discussion, we present the following contributions to the literature.

- We present a coverage-based filtering approach towards just-in-time mutation testing without running each mutant–test pair for Simulink and Stateflow Cyber-Physical System Models.
- We introduce an empirical evaluation of the usage of the coverage-based filtering approach in comparison with the baseline random selection.
- We conduct experiments on 2 open-source Stateflow models (HybridFSM and Cruise Control) (including Simulink submodels) and 1 industrial Simulink model (PID Controller) from 2 different safety-critical domains (namely automotive and industrial controller).

The rest of this paper is organized as follows: we start with the background and related work in Section 2 and we introduce our coverage-based approach in Section 3 and the evaluation setup and the subject models in Section 4. We present our empirical evaluation and results in Section 5, and a discussion of expected benefits and open challenges in Section 6 and potential threats to validity and their mitigation methods in Section 7. We conclude the paper and share the future work in Section 8.

2. Background and Related Work

This section provides details on an illustrative example of the problem this paper addresses and related work on mutation testing of Simulink/Stateflow models and mutation-testing cost-reduction techniques.

To illustrate the execution-cost problem, we consider a hypothetical Railway — level-crossing gate control Simulink model with 100 test cases, where one execution of the full test suite takes about 30 minutes. If mutation testing generates 1,000 mutants, evaluating each mutant against the full test suite would require roughly 500 hours of execution time, corresponding to about three weeks of continuous execution.

In practice, such a long validation window may not be available. After a model change, a continuous integration pipeline or a pre-release validation process may allow only one hour for mutation testing. In the example above, one hour represents only 0.2% of the time required for the complete analysis.

[4] https://github.com/haliliceylan/MUT4SLX, Last Accessed: 19.08.2026

We therefore treat mutation testing as a time-budgeted process, where the execution budget is fixed in advance, and the goal is to keep the resulting mutation score as close as possible to that of the complete analysis.

2.1. Mutation Testing for Simulink Cyber-Physical System Models

Mutation testing applied to Simulink models dates back to 2005, when Zhan and Clark proposed three categories of mutation operators (addition, multiplication, and assignment) that modify the input signal by/with a specified value [11]. Brillout et al. used mutation testing to generate bounded, model-checking-based tests for Simulink models [12].

A taxonomy of Simulink model mutation classes introduced four categories to characterize simple faults that designers might introduce (competent developer hypothesis) [13]. Subsequently, a more comprehensive set of eight categories was proposed to evaluate Simulink model clone-detection techniques [14].

The first prototype tool, SIMULTATE, performs automated mutation testing for Simulink models [15]. It uses a Python interface to orchestrate the workflow, enabling users to add further mutation operators (beyond the initially proposed 4) and to incorporate fault injection (covering 5 defined fault models). FIM is then presented as a proof-of-concept tool that inherits the mutation operators of SIMULATE [16].

MUT4SLX is a state-of-the-art mutation testing solution that supports both automatic mutant generation and execution [17, 9, 10]. It offers an extensive collection of parameter-based mutation operators—currently 28 for Simulink and 18 for Stateflow—derived from realistic faults extracted from an industrial bug database. Furthermore, it is a distinctive tool that provides both requirement traceability and compatibility with CI/CD pipelines. BERTiMuS uses large language models (based on CodeBERT, a pre-trained model for programming languages) for a mutant generation approach for Simulink models [18]. Valle et al. also presented a mutant generation approach for Simulink-Stateflow using LLMs with different prompting strategies [19]. Because of its extensive set of mutation operators and support for Simulink, Stateflow, and their combination, we included MUT4SLX as the only option in our experiments. Also, since our subject models include models with both Signal Builder and Simulink Test, MUT4SLX also supports those test environments.

2.2. Mutation Testing Cost Reduction Techniques

Reducing the computational and practical costs of mutation testing has long been a research topic. Researchers have introduced and used numerous methods to lower mutation testing costs [5].

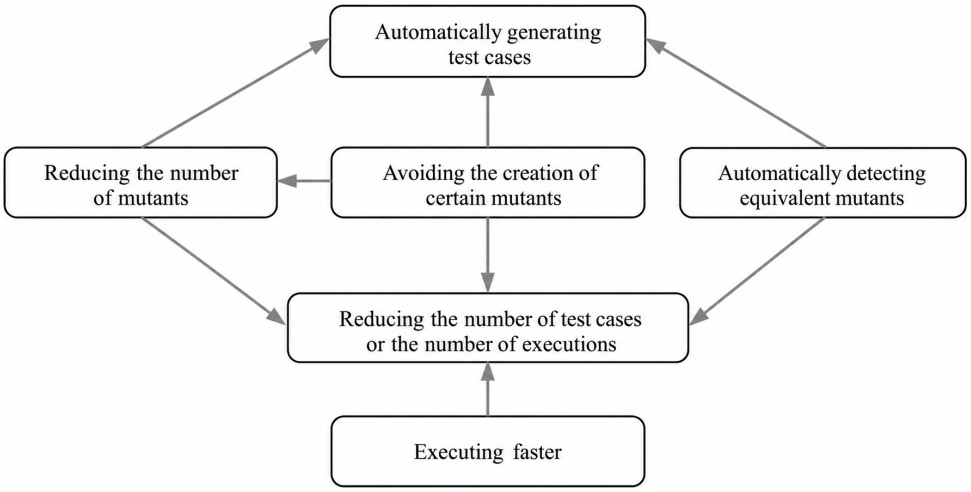


Figure 1: Relationships among mutation testing improvement goals [5].

Offutt and Untch classified cost-reduction techniques into three categories: do fewer, do smarter, and do faster [20]. 'Do fewer' approaches aim to run fewer mutants while avoiding unacceptable

information loss [21, 22, 23]. 'Do smarter' approaches try to spread the computational cost across multiple machines (such as cloud-based solutions [24])or amortize it over multiple executions [25, 26]. 'Do faster' approaches concentrate on generating and executing each mutant as efficiently as possible [27, 28]. Pizzoleto et al. conducted a systematic literature review on techniques and metrics to reduce the cost of mutation testing [5]. Their analysis resulted in the six primary goals shown in Figure 1. Our approach focuses on reducing the number of mutants and test cases using coverage-based evidence. In addition to reducing cost, we primarily aim to narrow the overall test execution time to a hard time budget and keep the mutation score as close as possible to the full baseline score. This idea stems from fault-injection research, specifically the FARM model [29], in which a controller uses optimization techniques to minimize execution time and maximize effectiveness [30]. Reschke et al. explored another direction for reducing mutation-testing costs through what they term *just-in-time and language-agnostic mutation testing* [31]. Their approach applies mutations dynamically during test execution using Truffle's instrumentation API. Their proof of concept targets a small JavaScript example and uses *just-in-time* to refer to the run-time application of mutations, whereas our use of the term concerns budget-driven mutation testing for Simulink/Stateflow models.

3. Proposed Just-in-Time Mutation Testing Approach (RQ1)

Figure 2 shows the overall workflow of the proposed approach.

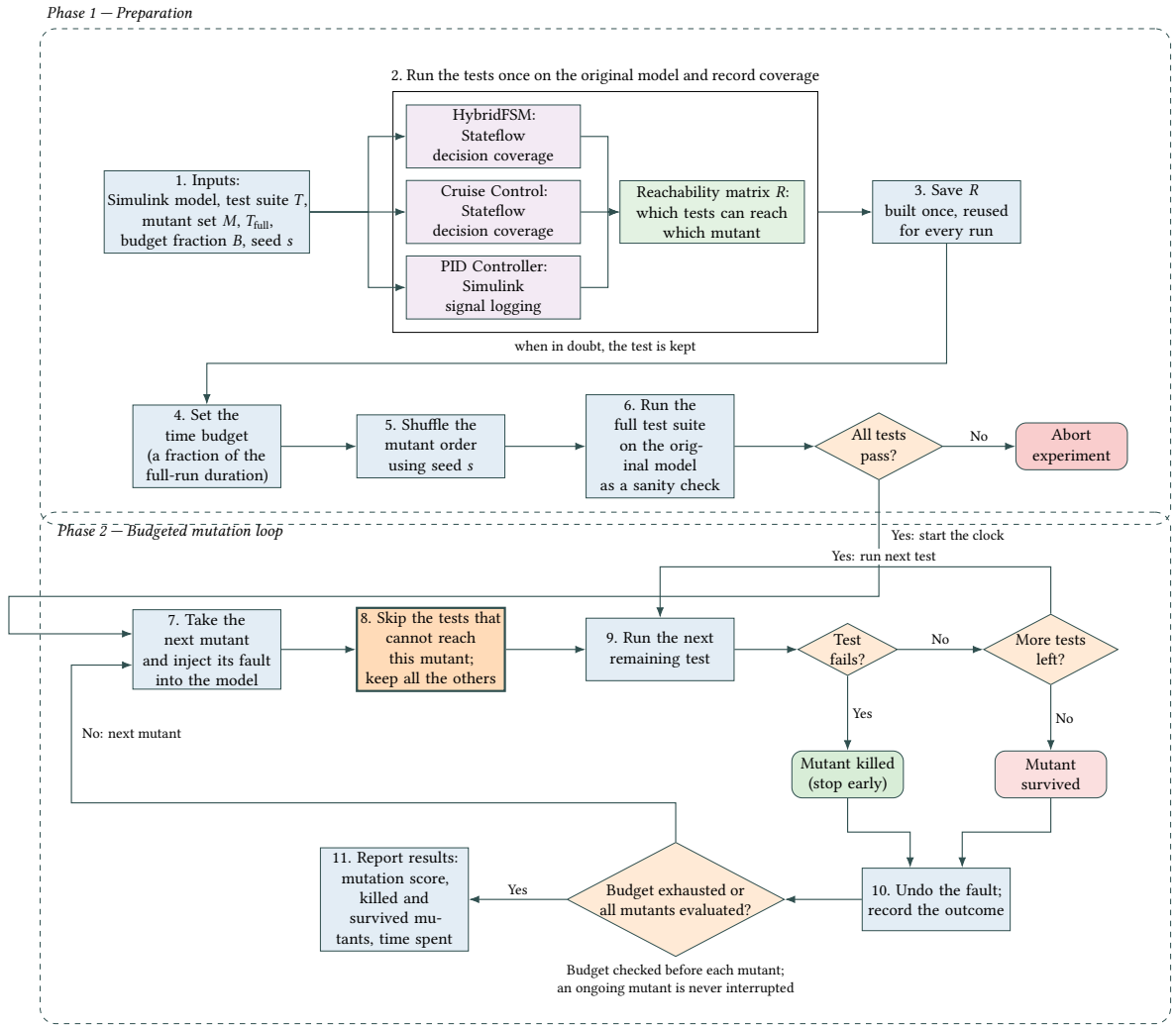


Figure 2: Workflow of the coverage-based budgeted mutation testing approach.

The approach consists of two main phases. In the first phase, a reachability matrix R is created for the test–mutant pairs using the model, the test suite, and the mutant set. Each $R(t, m)$ entry corresponds to a test–mutant pair. The matrix is built from coverage information collected by running the test suite once on the original model before mutation testing begins. The type of information used depends on the model structure. Stateflow decision coverage is used for HybridFSM and Cruise Control. For the PID Controller, Simulink signal logging is used instead, recording which branches of its Switch blocks each test selects. If it can be safely shown that a test cannot reach a specific mutant, the corresponding test–mutant pair is excluded; otherwise, including uncertain cases, the pair is kept for execution. The resulting R matrix is generated once and reused across different random seeds and time-budget settings.

In the second phase, the time budget is set as a selected percentage of the full mutation baseline time, T_{full}, and the mutants are ordered according to the selected random seed. Before mutation testing starts, the full test suite is rerun on the original model as a sanity check to identify test failures, including failures related to test flakiness. The mutants are then applied one by one in the generated order. For each mutant, the corresponding column of R is used to exclude tests that are known to be unreachable, while the remaining tests are executed in their original order. If any selected test fails, the mutant is classified as killed, and the remaining tests for that mutant are skipped. If all selected tests pass, the mutant is classified as survived. After each mutant execution, the mutation is reverted and the time budget is checked. Since the budget is checked only between mutant executions, a mutant that has already started is allowed to complete. This can lead to small differences between the nominal time budget and the observed execution time. The process terminates when the time budget is reached or when all mutants have been evaluated, after which the experiment results are reported.

4. Evaluation Setup and Subject Models

Experiments were conducted on a MacBook Pro equipped with an Apple M5 processor with 10 CPU cores (4 performance and 6 efficiency cores) and 32 GB of RAM. The machine was running macOS 26.2 and MATLAB/Simulink R2026a.

Table 1: Structural characteristics of the subject models, their test suites, and the mutant populations generated by MUT4SLX.

	HybridFSM	PID Controller	Cruise Control
Model structure			
Simulink blocks	16	246	18
Subsystems	— (flat)	9	1
Stateflow charts	1	—	1
Stateflow states	5	—	7
Stateflow transitions	22	—	23
Chart data variables	12	—	11
Test suite and mutants			
Test cases	25	21	15
Mutants	183	314	67

The first subject, *HybridFSM*[5], is a torque-allocation state machine from an open-source plug-in hybrid electric vehicle (PHEV) controller project, implemented as a flat Stateflow chart. The second subject, *PID Controller*, is an industrial controller implemented as a Simulink model. The third subject, *Cruise Control*[6], is a speed-management controller implemented as a hierarchical Stateflow chart. These subjects allow the proposed approach to be evaluated across different model characteristics. These characteristics include the model type (Stateflow-heavy or Simulink-heavy), the testing strategy used

[5] https://github.com/meltinglab/hybrid-controller, Last Accessed: 25.08.2026
[6] https://github.com/mathworks/requirements-based-testing-example, Last Accessed: 28.08.2026

by the model (Simulink Test or Signal Builder with embedded assertion blocks), and their design for use cases from different domains. Since the selection focuses on diversity, the subjects should not be considered as a representative sample of Simulink models.

5. Evaluation Results (RQ2)

Tables 2–4 show the results of each method at each time budget across the five seeds (mean that the executions have been repeated 5 times). Actual time and mutation score are reported as mean ± standard deviation, while mutants evaluated, mutants killed, and tests used are reported as median [IQR] across the five seeds, where IQR stands for interquartile range.

Table 2: HybridFSM results for the Full Baseline, Random Mutant, and Coverage-Based methods.

Method	Budget	Time Budget (s)	Actual Time (s)	Mutation Score (%)	Mutants Evaluated	Mutants Killed	Tests Used
Full Baseline	100%	2909.79	2909.79	76.50	183	140	25
Random Mutant	2%	58.20	65.43 ± 5.53	81.33 ± 18.50	5 [1]	4 [2]	25 [0]
Random Mutant	4%	116.39	123.54 ± 7.04	80.16 ± 14.18	11 [1]	9 [2]	25 [0]
Random Mutant	6%	174.59	182.35 ± 6.88	76.78 ± 11.00	17 [3]	12 [1]	25 [0]
Random Mutant	8%	232.78	239.59 ± 5.74	78.71 ± 7.40	23 [1]	19 [4]	25 [0]
Random Mutant	10%	290.98	302.11 ± 9.04	77.21 ± 8.75	29 [1]	24 [5]	25 [0]
Random Mutant	20%	581.96	594.58 ± 5.00	75.38 ± 6.33	55 [1]	41 [6]	25 [0]
Random Mutant	30%	872.94	877.82 ± 3.43	76.42 ± 2.76	80 [8]	62 [4]	25 [0]
Random Mutant	40%	1163.91	1172.57 ± 7.53	75.29 ± 2.11	105 [13]	79 [7]	25 [0]
Random Mutant	50%	1454.89	1463.40 ± 6.42	74.41 ± 2.38	122 [20]	87 [12]	25 [0]
Coverage-Based	2%	58.20	61.59 ± 4.11	76.79 ± 9.34	22 [6]	16 [7]	23 [1]
Coverage-Based	4%	116.39	118.59 ± 0.77	76.45 ± 5.25	43 [5]	31 [5]	24 [0]
Coverage-Based	6%	174.59	177.95 ± 3.30	75.45 ± 4.40	60 [6]	44 [3]	24 [0]
Coverage-Based	8%	232.78	236.76 ± 3.62	75.73 ± 1.91	81 [6]	61 [5]	24 [0]
Coverage-Based	10%	290.98	294.41 ± 2.77	75.67 ± 1.97	87 [10]	67 [7]	24 [0]
Coverage-Based	20%	581.96	564.77 ± 31.50	75.75 ± 0.94	181 [27]	138 [23]	24 [0]
Coverage-Based	30%	872.94	613.26 ± 76.01	76.50 ± 0.00	183 [0]	140 [0]	24 [0]
Coverage-Based	40%	1163.91	623.41 ± 77.70	76.50 ± 0.00	183 [0]	140 [0]	24 [0]
Coverage-Based	50%	1454.89	634.92 ± 85.40	76.50 ± 0.00	183 [0]	140 [0]	24 [0]

Table 3: Cruise Control results for the Full Baseline, Random Mutant, and Coverage-Based methods.

Method	Budget	Time Budget (s)	Actual Time (s)	Mutation Score (%)	Mutants Evaluated	Mutants Killed	Tests Used
Full Baseline	100%	2440.78	2440.78	61.19	67	41	15
Random Mutant	2%	48.82	93.49 ± 10.14	36.00 ± 35.07	2 [1]	1 [1]	15 [0]
Random Mutant	4%	97.63	157.90 ± 30.52	46.00 ± 31.13	3 [1]	2 [1]	15 [0]
Random Mutant	6%	146.45	172.56 ± 11.85	43.33 ± 27.89	3 [1]	2 [1]	15 [0]
Random Mutant	8%	195.26	229.00 ± 32.32	46.29 ± 32.56	4 [1]	3 [2]	15 [0]
Random Mutant	10%	244.08	282.75 ± 23.96	43.33 ± 26.25	5 [1]	3 [1]	15 [0]
Random Mutant	20%	488.16	527.51 ± 21.20	45.17 ± 18.31	10 [3]	6 [2]	15 [0]
Random Mutant	30%	732.23	763.72 ± 19.73	52.05 ± 23.46	18 [7]	12 [4]	15 [0]
Random Mutant	40%	976.31	1030.12 ± 40.47	54.66 ± 16.48	25 [8]	15 [7]	15 [0]
Random Mutant	50%	1220.39	1252.12 ± 20.06	54.46 ± 11.43	29 [10]	18 [8]	15 [0]
Coverage-Based	2%	48.82	79.52 ± 19.53	32.67 ± 34.19	2 [2]	1 [1]	14 [2]
Coverage-Based	4%	97.63	135.59 ± 24.16	44.33 ± 32.18	4 [1]	2 [1]	14 [0]
Coverage-Based	6%	146.45	156.20 ± 3.88	52.50 ± 31.12	4 [4]	3 [3]	14 [0]
Coverage-Based	8%	195.26	220.70 ± 23.34	50.00 ± 28.87	6 [3]	4 [3]	14 [0]
Coverage-Based	10%	244.08	260.37 ± 22.26	45.33 ± 26.31	10 [6]	6 [3]	14 [0]
Coverage-Based	20%	488.16	510.25 ± 12.45	53.80 ± 17.04	22 [13]	15 [8]	14 [0]
Coverage-Based	30%	732.23	764.09 ± 22.78	55.01 ± 7.49	31 [11]	19 [9]	14 [1]
Coverage-Based	40%	976.31	1011.64 ± 32.98	56.52 ± 6.15	39 [9]	22 [9]	15 [0]
Coverage-Based	50%	1220.39	1227.53 ± 5.05	57.69 ± 5.51	47 [12]	28 [11]	15 [0]

As shown in Table 2, the HybridFSM full baseline evaluates all 183 mutants with the complete 25-test suite, kills 140 of them, and achieves a mutation score of 76.50% in 2909.79 seconds. With Random Mutant, the evaluated and killed counts increase with the budget. However, even at the 50% budget, the method evaluates a median of 122 mutants, kills a median of 87, and remains below the baseline with

a mutation score of 74.41%. The coverage-based approach uses the same budgets more effectively by excluding unreachable test–mutant pairs. At the 20% budget, it evaluates a median of 181 mutants and kills 138, achieving a mutation score of 75.75%. At the 30% budget, it evaluates all 183 mutants, kills 140, and achieves the full baseline mutation score of 76.50% in an average of 613.3 seconds, compared with 2909.79 seconds for the full analysis. The same result is maintained at the 40% and 50% budget levels.

For Cruise Control (Table 3), the full baseline mutation score of 61.19% is used as the reference result. At the 50% budget, random mutant evaluates a median of 29 mutants, kills a median of 18, and achieves a mutation score of 54.46% ± 11.43.

Table 4: PID Controller results for the Full Baseline, Random Mutant, and Coverage-Based methods.

Method	Budget	Time Budget (s)	Actual Time (s)	Mutation Score (%)	Mutants Evaluated	Mutants Killed	Tests Used
Full Baseline	100%	292.36	292.36	42.04	314	132	21
Random Mutant	2%	5.85	7.01 ± 0.51	23.33 ± 22.36	6 [2]	2 [2]	21 [0]
Random Mutant	4%	11.69	12.62 ± 0.62	30.34 ± 13.92	11 [2]	3 [1]	21 [0]
Random Mutant	6%	17.54	18.81 ± 0.40	35.16 ± 5.61	17 [1]	6 [1]	21 [0]
Random Mutant	8%	23.39	23.88 ± 0.49	39.13 ± 6.36	23 [0]	9 [3]	21 [0]
Random Mutant	10%	29.24	30.34 ± 0.62	40.88 ± 5.06	29 [6]	13 [4]	21 [0]
Random Mutant	20%	58.47	59.34 ± 0.39	38.20 ± 4.54	57 [5]	23 [5]	21 [0]
Random Mutant	30%	87.71	88.36 ± 0.35	40.20 ± 4.77	84 [8]	32 [6]	21 [0]
Random Mutant	40%	116.94	118.11 ± 0.48	41.59 ± 5.18	117 [8]	48 [4]	21 [0]
Random Mutant	50%	146.18	147.45 ± 0.57	42.44 ± 4.49	145 [16]	57 [7]	21 [0]
Coverage-Based	2%	5.85	6.93 ± 0.68	23.33 ± 22.36	6 [2]	2 [2]	21 [0]
Coverage-Based	4%	11.69	12.46 ± 0.66	30.82 ± 13.06	11 [5]	3 [2]	21 [0]
Coverage-Based	6%	17.54	17.85 ± 0.50	36.92 ± 7.71	18 [6]	7 [3]	21 [0]
Coverage-Based	8%	23.39	23.94 ± 0.43	38.30 ± 5.97	25 [2]	9 [3]	21 [0]
Coverage-Based	10%	29.24	30.24 ± 0.50	39.92 ± 4.15	31 [2]	13 [4]	21 [0]
Coverage-Based	20%	58.47	59.28 ± 0.43	39.20 ± 5.74	65 [4]	27 [9]	21 [0]
Coverage-Based	30%	87.71	88.56 ± 0.55	39.41 ± 4.22	94 [4]	38 [6]	21 [0]
Coverage-Based	40%	116.94	118.05 ± 0.34	42.38 ± 5.04	131 [4]	54 [5]	21 [0]
Coverage-Based	50%	146.18	146.66 ± 0.34	42.39 ± 4.31	155 [14]	64 [12]	21 [0]

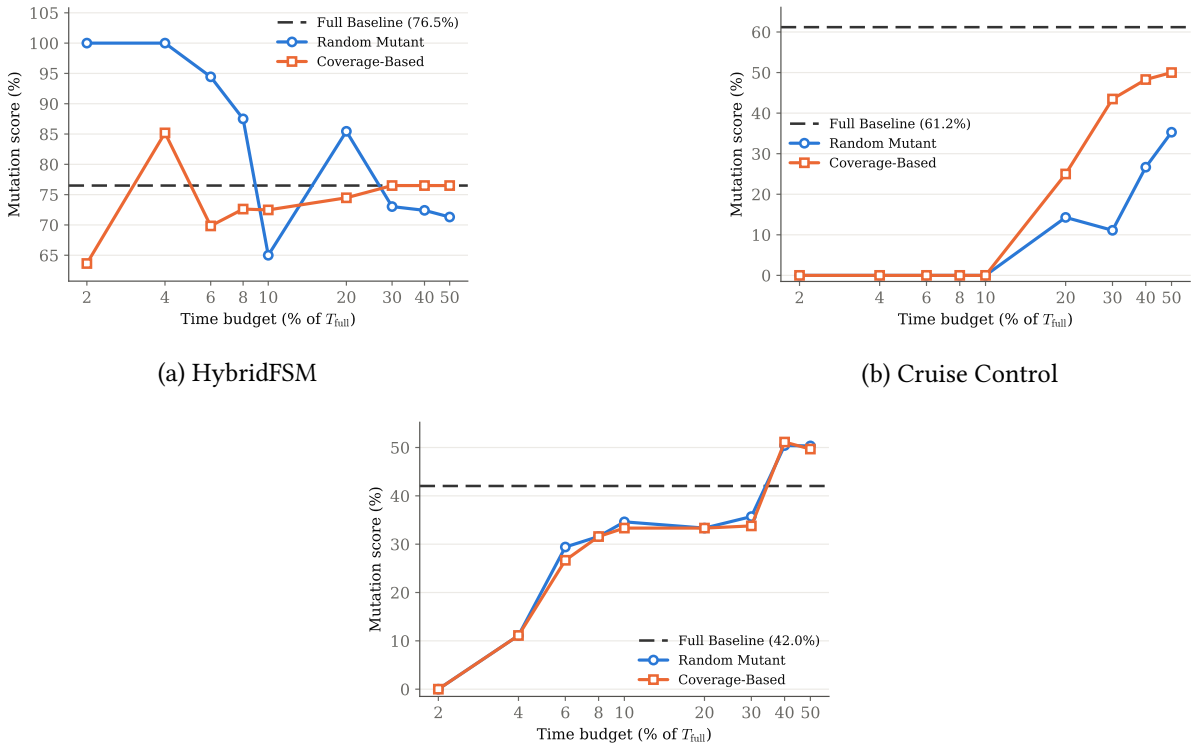
Across all budgets from 6% to 50%, the coverage-based approach evaluates and kills a higher median number of mutants than the random mutant approach. At the 50% budget, it evaluates a median of 47 mutants, kills a median of 28, and achieves a mutation score of 57.69% ± 5.51. At budgets of 30% and above, its mutation scores also have lower standard deviations than those of random mutant.

For the PID Controller (Table 4), the full baseline evaluates all 314 mutants, kills 132, and achieves a mutation score of 42.04%. The two budgeted methods produce similar mutation scores across the examined budgets. At the 50% budget, random mutant evaluates a median of 145 mutants, kills a median of 57, and achieves a mutation score of 42.44%. At the same budget, the coverage-based approach evaluates a median of 155 mutants, kills a median of 64, and achieves a mutation score of 42.39%. Across all budgets from 20% to 50%, the coverage-based approach evaluates and kills a higher median number of mutants than random mutant, although the differences in mutation score remain small.

Figure 3 shows a comparison between these two approaches based on the runs with mutation scores that deviate the most from the baseline across all time budgets. The coverage-based curve for HybridFSM (a) approaches the full baseline early and then remains stable, staying exactly on the baseline from the 30% budget onward. On the other hand, the random mutant curve varies across budgets, going above the baseline at small budgets and below it at large budgets. For Cruise Control (b), both curves start at 0% because few mutants were evaluated and none were killed in the selected worst-case runs. However, after the 10% budget, the coverage-based curve rises toward the baseline more steadily and with fewer changes than the random mutant curve. For the PID Controller (c), both methods produce nearly identical mutation scores across the examined budgets.

6. Discussion on Expected Benefits and Open Challenges (RQ3)

The results indicate that coverage-based approach can reduce unnecessary test–mutant executions; however, its effectiveness depends strongly on the model characteristics and the extent to which the



(a) HybridFSM

(b) Cruise Control

(c) PID Controller

Figure 3: Worst-case seed mutation scores under increasing time budgets.

tests exercise different parts of the model. Table 5 summarizes the test–mutant pairs excluded or retained for each subject. The experiments also highlighted several technical challenges associated with coverage collection, the construction of the reachability matrix, and the enforcement of time budgets. These findings, together with the limitations of the study, are discussed below.

Table 5: Test–mutant pair filtering per subject, independent of budgets and seeds.

	HybridFSM	Cruise Control	PID Controller
Tests × mutants	25 × 183	15 × 67	21 × 314
Total pairs	4575	1005	6594
Excluded pairs	2720	340	488
Excluded (%)	59.5	33.8	7.4
Retained pairs	1855	665	6106

The proposed approach enables mutation-testing results to be obtained within a predefined time limit without waiting for the complete analysis to finish. This makes long-running mutation testing more manageable when execution time is limited and supports its integration into continuous integration, release testing, and other time-constrained testing activities. This benefit was most clearly observed for HybridFSM, where the coverage-based approach evaluated all 183 mutants and matched the full baseline mutation score at the 30% budget level. In HybridFSM, the analysis excluded 59.5% of all test–mutant pairs while in Cruise Control it was limited to 33.8%. This difference is related to the structure of the two state machines. In HybridFSM the five states are independent, so each test visits its own state and stays away from the others, which makes it easy to filter a test that cannot reach a mutant. In Cruise Control, the states are nested inside each other, so to reach an inner state every test must first pass through the same parent ones; the mutants located there are reached by almost every test, and fewer test–mutant pairs can be filtered. The PID Controller has almost no places that a test can avoid. The model is a calculation that runs from input to output at every simulation step, so every test exercises nearly every block. As a result, it is rarely possible to exclude a test that cannot reach a mutant, and only 7.4% of the

test–mutant pairs could be excluded, all of them through the branches of its Switch blocks. In addition to the time savings from filtering, reusing the reachability matrix across different time budgets and repeated mutation-testing runs distributes its construction cost over multiple executions.

A major open challenge arises when the time available for mutation testing is shorter than the time required to execute the original test suite once. Constructing the reachability matrix requires the complete test suite to be executed with coverage instrumentation on the unmutated model. For industrial models, this execution may itself take several hours, while mutation-testing results may be required within a much shorter time. In such cases, constructing the reachability matrix alone would exceed the available time, and the coverage-based approach could not produce mutation-testing results within the specified time limit on its first use. In such cases, random mutant sampling is worth considering as an alternative to the coverage-based approach.

7. Threats to Validity

As with all empirical research, we identify those factors that may jeopardize the validity of our results and the actions we took to reduce or alleviate the risk. Consistent with the guidelines for case study research (see [32, 33]), we organize them into four categories.

Construct validity. We evaluate how closely the mutation score obtained within a time budget approximates the full baseline score. However, budgeted scores are calculated only over the evaluated mutants. A score close to the baseline does not necessarily mean that the same mutants were killed or the same parts of the model were exercised. Additional coverage metrics, such as decision, condition, and MC/DC coverage, could complement this evaluation. However, coverage recording affected execution times and was disabled during the timed runs. Our results also depend on the MUT4SLX operator set [9, 10] and the analyses used to construct the reachability matrix. Different mutation operators or reachability analyses may change the mutant population and the test–mutant pairs eligible for filtering.

Internal validity. Internal validity concerns whether the differences observed between random mutant and coverage-based under the same time budgets result from the methods themselves or from experimental factors affecting simulation time. During the study, coverage instrumentation left enabled in a model configuration substantially increased simulation times and caused progressive slowdown during long MATLAB sessions. This reduced the number of mutants evaluated within a given budget and could bias the comparison between the methods. We therefore disabled coverage instrumentation for all experiments reported in this study, ran each method in a fresh MATLAB session, and verified per-test execution times from the recorded traces. For each subject, T_{full} and all budgeted runs were measured on the same machine under the same configuration. The coverage-based approach also relies on the correct mapping of tests to mutants in the reachability matrix. To reduce the risk of incorrect filtering, uncertain relations were retained, and the matrix entries were verified against the intended test and mutant identifiers.

External validity. Our findings are limited to the three subjects studied: an industrial fixed-point Simulink PID controller and two Stateflow state machines. The filtering opportunities observed in these subjects should not be generalized to all Simulink or Stateflow models, since they depend on both the model structure and the test suite. In particular, the limited filtering observed for the PID Controller should not be interpreted as a characteristic of all pure Simulink models. In models with more conditional execution paths may allow more test–mutant pairs to be exclude. The PID subject also uses a legacy Signal Builder harness, which MathWorks no longer recommends and advises replacing with Signal Editor [34]. Additional studies across different models and test infrastructures are needed to evaluate the broader applicability of our results.

Reliability. Execution times are also environment-dependent. We therefore ran all experiments on the same machine and software configuration and defined each budget as a percentage of the subject-specific T_{full} measured under the same conditions. To ensure a fair comparison, we repeated each budgeted configuration five times using fixed seeds 1–5. Each seed produced a different mutant ordering, which we reused across both methods and all budget levels. We retain the execution traces and analysis scripts

needed to recompute the reported metrics from the recorded executions, while fixed seeds ensure reproducible mutant orderings.

8. Conclusion and Future Work

In this paper, we investigated mutation testing of Simulink models under predefined time budgets. We proposed a coverage-based filtering approach that uses reachability information collected from the unmutated model to exclude test–mutant executions that cannot contribute to killing a mutant. We used the full baseline as a reference and compared the coverage-based approach with random mutant selection on three subjects: HybridFSM, PID Controller, and Cruise Control.

Our results show that the benefit of the coverage-based approach depends on the reachability relationships between the tests and mutated model elements. The method provided its clearest benefit for HybridFSM, where it evaluated all 183 mutants and achieved the full baseline mutation score at the 30% budget level. It also performed better than the random mutant approach for Cruise Control, whereas its benefit was limited for the PID Controller because fewer test–mutant executions could be filtered.

For future work, we plan to conduct experiments on a larger and more diverse set of Simulink and Stateflow models. We plan to collect coverage information by running the tests on the unmutated model to identify which model elements they execute. We will use this information to select subsystems containing elements executed by at least one test. Mutants will then be randomly sampled from these subsystems rather than from the entire model. We could also consider mutant subsumption using the reachability matrix together with kill information. Our filtering approach is related to the Reach condition of the RIPR model [35], as it uses information on which tests can reach mutated elements. Future work could extend this idea to the Infect condition by using internal states and signal values recorded during the original model execution to reduce unnecessary test–mutant executions. We plan to repeat the experiments with more random seeds and use statistical tests and effect size measures [36] to compare random mutant selection and coverage-based filtering in terms of mutation score and execution time across different models and time budgets.

9. Data Availability Statement

The authors confirm that the data supporting the findings and experiments of this study for the closed-source version of the MUT4SLX tool and industrial Simulink model are subject to third-party restrictions and cannot be shared due to the high sensitivity of our industrial partner's data. However, we share other resources[7] to replicate the experiments using the open-source MUT4SLX version or other mutation testing tools. Those supplementary materials include experiment scripts, analysis code, and seed-level results via our reproduction package.

10. Acknowledgments

This work is supported by (a) the Flanders Innovation & Entrepreneurship (VLAIO) under grant number HBC.2021.0010 entitled "EFFECTS"; (b) the Research Foundation Flanders (FWO) under grant number S000323N entitled "Basecamp Zero"; (d) Flanders Make, the strategic research centre for the manufacturing industry. (e) the Agency for Innovation & Entrepreneurship (VLAIO) via the project "TTRUST" under grant number HBC20230612. (f) the Industrial Research Fund (IOF) via the project "MUT4SLX" under grant number 54104.

11. Declaration on Generative AI

During the preparation of this work, the author(s) used Grammarly for grammar and spelling checks, paraphrasing, and rewording. The author(s) used Claude (Anthropic) to write the Figure 2 code. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

[7] https://github.com/EmirayDurmaz/benevol2026-reproduction-package, Last Accessed: 11.09.2026

References

[1] R. G. Hamlet, Testing programs with the aid of a compiler, IEEE transactions on software engineering (1977) 279–290. doi:10.1109/TSE.1977.231145.

[2] R. A. DeMillo, R. J. Lipton, F. G. Sayward, Hints on test data selection: Help for the practicing programmer, Computer 11 (1978) 34–41. doi:10.1109/C-M.1978.218136.

[3] Y. Jia, M. Harman, An analysis and survey of the development of mutation testing, IEEE transactions on software engineering 37 (2010) 649–678. doi:10.1109/TSE.2010.62.

[4] W. E. Wong, A. P. Mathur, Reducing the cost of mutation testing: An empirical study, Journal of Systems and Software 31 (1995) 185–196. doi:10.1016/0164-1212(94)00098-0.

[5] A. V. Pizzoleto, F. C. Ferrari, J. Offutt, L. Fernandes, M. Ribeiro, A systematic literature review of techniques and metrics to reduce the cost of mutation testing, Journal of Systems and Software 157 (2019) 110388. doi:10.1016/j.jss.2019.07.100.

[6] M. P. Usaola, P. R. Mateo, Mutation testing cost reduction techniques: a survey, IEEE software 27 (2010) 80–86. doi:10.1109/MS.2010.79.

[7] M. Polo, M. Piattini, I. García-Rodríguez, Decreasing the cost of mutation testing with second-order mutants, Software Testing, Verification and Reliability 19 (2009) 111–131. doi:10.1002/stvr.392.

[8] E. A. Lee, The past, present and future of cyber-physical systems: A focus on models, Sensors 15 (2015) 4837–4869. doi:10.3390/s150304837.

[9] H. I. Ceylan, O. Kilincceker, M. Beyazit, S. Demeyer, MUT4SLX: Fast mutant generation for Simulink, in: 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, 2023, pp. 2086–2089. doi:10.1109/ASE56229.2023.00093.

[10] S. Nuyens, H. I. Ceylan, O. Kilincceker, M. Beyazit, S. Demeyer, MUT4SLX: extensions for mutation testing of Stateflow models, in: 2024 IEEE International Conference on Software Analysis, Evolution and Reengineering-Companion (SANER-C), IEEE, 2024, pp. 215–218. doi:10.1109/SANER-C62648.2024.00035.

[11] Y. Zhan, J. A. Clark, Search-based mutation testing for Simulink models, in: Proceedings of the 7th annual conference on Genetic and evolutionary computation, 2005, pp. 1061–1068. doi:10.1145/1068009.1068188.

[12] A. Brillout, N. He, M. Mazzucchi, D. Kroening, M. Purandare, P. Rümmer, G. Weissenbacher, Mutation-based test case generation for Simulink models, in: International Symposium on Formal Methods for Components and Objects, Springer, 2009, pp. 208–227. doi:10.1007/978-3-642-17071-3_11.

[13] L. T. M. Hanh, N. T. Binh, Mutation operators for Simulink models, in: 2012 Fourth International Conference on Knowledge and Systems Engineering, IEEE, 2012, pp. 54–59. doi:10.1109/KSE.2012.22.

[14] M. Stephan, M. H. Alalfi, J. R. Cordy, Towards a taxonomy for Simulink model mutations, in: 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops, IEEE, 2014, pp. 206–215. doi:10.1109/ICSTW.2014.17.

[15] I. Pill, I. Rubil, F. Wotawa, M. Nica, Simultate: A toolset for fault injection and mutation testing of Simulink models, in: 2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), IEEE, 2016, pp. 168–173. doi:10.1109/ICSTW.2016.21.

[16] E. Bartocci, L. Mariani, D. Ničković, D. Yadav, Fim: fault injection and mutation for Simulink, in: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2022, pp. 1716–1720. doi:10.1145/3540250.3558932.

[17] J. Kisaakye, O. Kilincceker, S. Demeyer, Towards mutation testing of simulink models, in: Proceedings of the 20th Belgium-Netherlands Software Evolution Workshop, volume Vol-3071, 2021, pp. 1–9. URL: https://ceur-ws.org/Vol-3071/paper15.pdf.

[18] J. Zhang, D. Ghobari, M. Sabetzadeh, S. Nejati, Simulink mutation testing using CodeBERT, in: 2025 IEEE/ACM International Conference on Automation of Software Test (AST), IEEE, 2025, pp. 24–28. doi:10.1109/AST66626.2025.00009.

[19] P. Valle, S. Ali, A. Arrieta, Exploring the potential of large language models in simulink-stateflow mutant generation, in: Proceedings of the 2026 IEEE/ACM Third International Conference on AI Foundation Models and Software Engineering, FORGE '26, Association for Computing Machinery, New York, NY, USA, 2026, p. 99–109. doi:10.1145/3793655.3793737.

[20] A. J. Offutt, R. H. Untch, Mutation 2000: Uniting the orthogonal, Mutation testing for the new century 4 (2001). doi:10.1007/978-1-4757-5939-6_7.

[21] G. Petrović, M. Ivanković, State of mutation testing at google, in: Proceedings of the 40th international conference on software engineering: Software engineering in practice, 2018, pp. 163–171. doi:10.1145/3183519.318352.

[22] Q. Zhu, A. Panichella, A. Zaidman, An investigation of compression techniques to speed up mutation testing, in: 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST),

IEEE, 2018, pp. 274–284. doi:10.1109/ICST.2018.00035.
[23] P. McMinn, C. J. Wright, C. J. McCurdy, G. M. Kapfhammer, Automatic detection and removal of ineffective mutants for the mutation analysis of relational database schemas, IEEE Transactions on Software Engineering 45 (2017) 427–463. doi:10.1109/TSE.2017.2786286.
[24] S. Vercammen, S. Demeyer, M. Borg, S. Eldh, Speeding up mutation testing via the cloud: lessons learned for further optimisations, in: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 2018, pp. 1–9. doi:10.1145/3239235.3240506.
[25] N. Li, M. West, A. Escalona, V. H. Durelli, Mutation testing in practice using ruby, in: 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), IEEE, 2015, pp. 1–6. doi:10.1109/ICSTW.2015.7107453.
[26] P. R. Mateo, M. P. Usaola, Parallel mutation testing, Software Testing, Verification and Reliability 23 (2013) 315–350. doi:10.1002/stvr.1471.
[27] X. Devroey, G. Perrouin, M. Papadakis, A. Legay, P.-Y. Schobbens, P. Heymans, Featured model-based mutation analysis, in: Proceedings of the 38th International Conference on Software Engineering, 2016, pp. 655–666. doi:10.1145/2884781.2884821.
[28] B. Wang, S. Lu, Y. Xiong, F. Liu, Faster mutation analysis with fewer processes and smaller overheads, in: 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, 2021, pp. 381–393.
[29] M.-C. Hsueh, T. K. Tsai, R. K. Iyer, Fault injection techniques and tools, Computer 30 (1997) 75–82. doi:10.1109/2.585157.
[30] M. Moradi, T. Fabarisov, O. Kilincceker, M. Challenger, J. Denil, Optimization methods for model-implemented fault injection in cyber-physical systems: A systematic literature review, Journal of Systems and Software (2026) 112957. doi:10.1016/j.jss.2026.112957.
[31] S. Reschke, T. Mattis, F. Niephaus, R. Hirschfeld, Toward just-in-time and language-agnostic mutation testing, in: Companion Proceedings of the 6th International Conference on the Art, Science, and Engineering of Programming (Programming '22 Companion), ACM, 2022, pp. 6–11. doi:10.1145/3532512.3532514.
[32] P. Runeson, M. Höst, Guidelines for conducting and reporting case study research in software engineering, Empirical Softw. Engineering 14 (2009) 131–164. doi:10.1007/s10664-008-9102-8.
[33] R. K. Yin, Case Study Research: Design and Methods, 3 edition, Sage Publications, —, 2002.
[34] The MathWorks, Inc., Migrate from Signal Builder block to Signal Editor block, https://www.mathworks.com/help/simulink/ug/migrate-from-signal-builder-block-to-signal-editor-block.html, n.d. Accessed: Aug. 29, 2026.
[35] P. Ammann, J. Offutt, Introduction to Software Testing, 2 ed., Cambridge University Press, 2016. doi:10.1017/9781316771273.
[36] A. Arcuri, L. Briand, A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering, Software Testing, Verification and Reliability 24 (2014) 219–250. doi:10.1002/stvr.1486.