# Cost Measures Matter for Mutation Testing Study Validity

Giovani Guizzo
Federica Sarro
Mark Harman
g.guizzo@ucl.ac.uk
f.sarro@ucl.ac.uk
m.harman@ucl.ac.uk
Department of Computer Science, University College London, London, United Kingdom

## ABSTRACT

Mutation testing research has often used the number of mutants as a surrogate measure for the true execution cost of generating and executing mutants. This poses a potential threat to the validity of the scientific findings reported in the literature. Out of 75 works surveyed in this paper, we found that 54 (72%) are vulnerable to this threat. To investigate the magnitude of the threat, we conducted an empirical evaluation using 10 real-world programs. The results reveal that: i) percentages of randomly sampled mutants differ from the true execution time, on average, by 44%, varying in difference from 19% to 91%; ii) errors arising from using the surrogate correlate with program size ($\rho = 0.74$) and number of mutants ($\rho = 0.76$), making the problem more pernicious for more realistic programs; iii) scientific findings concerning sampling strategies would have approximately 37% rank disagreement, indicating potentially dramatic impact on experiment validity. To investigate whether this threat matters in practice, we reproduced a seminal study on Selective Mutation (widely relied upon for more than two decades). The impact is stark: an inconclusive scientific finding using the surrogate is transformed to an unequivocal finding when using the true execution cost.

## CCS CONCEPTS

• **Software and its engineering** → **Software testing and debugging**; • **General and reference** → *Empirical studies*; Measurement.

## KEYWORDS

Software Testing, Mutation Testing, Mutation Analysis, Cost Reduction, Number of Mutants, Execution Time, Mutant Reduction

**ACM Reference Format:**
Giovani Guizzo, Federica Sarro, and Mark Harman. 2020. Cost Measures Matter for Mutation Testing Study Validity. In *Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '20), November 8–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3368089.3409742

## 1 INTRODUCTION

Mutation Testing has been widely studied and empirically evaluated in academia for several years [41, 67]. A mutant program is a copy of the original program with a seeded fault, which is revealed when the output of a test case for the original program differs from the output of the same test case when executed against the mutant (in which case the mutant is said to be killed). If no possible test case can kill a mutant, then the mutant is defined to be an equivalent mutant. Assessing the equivalence of mutants is generally undecidable [8] and usually is done, at least partly, manually by the engineer. The primary goal of mutation testing is to kill as many non-equivalent mutants as possible, increase the mutation score (i.e., percentage of killed mutants), and improve the test suite as a consequence. The greater the mutation score, the better the test suite. This makes mutation analysis[1] one of the strongest testing criteria for evaluating or guiding the creation of test suites [42, 57, 58]. However, its high cost is one of the primary reasons why it has not been extensively adopted in industry.

The cost of mutation analysis [14, 16, 64, 75] comes mainly from: i) mutant generation; ii) mutant execution; and iii) equivalent mutant assessment. During the first step, mutants are generated by mutation operators that seed a fault (mutation) in the original program. Then, test cases are executed against mutants in the second phase to try to reveal the faults represented by such mutants. Alive mutants are then evaluated to check for equivalence. As one can infer, generating several mutated programs, executing them against multiple test cases, and manually deciding which ones are equivalent demand a large amount of computational and manual resources. In the end, the cost of mutation analysis can be broken down into two: i) mutant generation and execution time (i.e., computational cost); and ii) person-hours for assessing mutant equivalence (i.e., human cost).

Several cost reduction techniques have been proposed to minimise the computational cost [75]. These techniques can be classified into three main categories [65]: i) "do faster"; ii) "do smarter"; and iii) "do fewer". "Do faster" approaches focus on generating and executing mutants as fast as possible, e.g., mutating compiled programs instead of source code, or using compiler related optimisation [9, 17]. "Do smarter" approaches try to distribute the execution

*ESEC/FSE '20, November 8–13, 2020, Virtual Event, USA*
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7043-1/20/11...$15.00
https://doi.org/10.1145/3368089.3409742

[1] Mutation analysis can be defined as the testing criterion used to measure the quality of a test suite, whereas mutation testing is the process of testing the software based on the results of mutation analysis [65].

of mutants over several machines or even avoid their complete execution, e.g., stopping the mutant execution after the mutated line is executed [37, 91]. "Do fewer" strategies try to generate or execute fewer mutants, consequently reducing the computational time needed for such tasks, e.g., by randomly sampling and executing $x\%$ out of all the generated ones (Random Mutant Sampling – RMS), or by using only a subset of operators to generate the mutants (Selective Mutation – SM) [64, 94, 95].

"Do faster" and "do smarter" approaches usually measure the cost reduction in terms of execution speed-up. On the other hand, "do fewer" approaches usually rely on the assumption that there is a correlation between the number of mutants and the computational cost of generating and executing them. The existence of a correlation is undeniable: the fewer mutants generated and executed, the faster the mutation analysis. This assumption is so widespread that, according to the recent findings of Pizzoleto et al. [75], number of mutants has become the single most used cost reduction measure in the mutation testing literature. However, even though the assumption of correlation holds, it does not mean that a reduction of $x\%$ mutants necessarily translates into $x\%$ reduction in the computational cost of executing them. Hence, stating that "by reducing the number of mutants, the mutation cost is also reduced" is not the same as stating that "by reducing the number of mutants by $x\%$, the mutation cost is also reduced by $x\%$". This is a threat to validity that can lead to misleading conclusions about which strategy will be more effective on reducing the cost of mutation analysis.

Despite being a known threat to validity [20, 24, 29, 59, 93], researchers usually report the cost reduction in terms of $x\%$ fewer mutants and do not measure the reduction in execution time. As described in Section 2, out of 75 analysed papers, only 15 (20%) report execution time, and 54 of them (72%) are vulnerable to the threat presented in this work. This might seem to be a mere detail concerning the choice of measures, but we present results that demonstrate that this choice has a profound impact on the scientific conclusions that have been and will continue to be drawn on mutation analysis.

To the best of our knowledge, there is no work in the literature that investigates the validity of using number of mutants as a surrogate for execution cost, nor the magnitude of the error that may result in consequent threats to the validity of scientific conclusions. Therefore, in this paper we investigate the effects of using and reporting number of mutants rather than the actual execution time.

To this end, we carry out an empirical study encompassing 10 programs and 19 cost reduction strategies (based on RMS and SM), for which we measure both number of mutants and execution time percentages in relation to all mutants. The results of our study show that there is an error of, on average, 44% between number of mutants and execution time measurements. This error varies from 19% to 91% depending on the program. We also found that the mean percentage error is significantly (p-values $< 0.05$) and positively correlated to the size of the program (Spearman's $\rho = 0.74$), and to the number of generated mutants ($\rho = 0.76$), i.e., as the program grows, the relative errors also grow, thus making number of mutants an unreliable measure for assessing the cost of mutant cost reduction techniques in large real-world programs.

When ranking and comparing RMS and SM strategies using both number of mutants and execution time, we found that 37% of the ranks are disrupted. The changes of ranks are always favourable to SM, i.e., for 9 out of 10 programs at least one SM strategy that is considered more expensive than RMS according to number of mutants turns out to be cheaper according to execution time.

In order to determine whether this threat to validity matters in practice, we reproduce the work by Offutt et al. [63] with 10 real-world large scale Java programs. The results show that RS-Selective is the cheapest strategy according to execution time for all 10 programs. However, if we use number of mutants instead, this scientific conclusion changes from "unequivocal" to "inconclusive": both E- and RS-Selective generate fewer mutants for 5 programs each. As a result, we conclude that all work that relies on selective mutation needs to be re-evaluated to take account of the true execution time rather than the widely used surrogate.

The main contributions of this work are:

(1) **Relevance:** a survey of 75 mutant reduction papers. 60 of them (80%) do not report execution time, and 54 (72%) also use the number of mutants as a proxy for cost, being directly affected by the threat presented in this work.
(2) **Problem:** the quantification of this threat. There is an average error of 44% between the number of mutants and execution cost, varying from 19% to 91% depending on the program.
(3) **Problem is worse at scale:** an assessment of correlations between the program's properties and errors. Positive significant correlations (p-values $< 0.05$) with the programs size ($\rho = 0.74$) and the number of generated mutants ($\rho = 0.76$).
(4) **Changes scientific findings:** an analysis of the impact of these errors when comparing RMS and SM strategies. Disagreement on 37% of the ranks when using execution time instead of number of mutants.
(5) **Changes foundational scientific conclusions:** the reproduction of an SM work [63]. Ranks between the cheapest strategies change for 5 out of 10 programs, fundamentally changing the scientific conclusions as a consequence.

The rest of the paper is organised as follows. We survey previous work on cost reduction in Section 2, assessing which ones report execution cost and which ones acknowledge the investigated threat. Section 3 describes the Research Questions (RQs) and the empirical evaluation conducted to answer them. In Section 4 we report the results, answer the RQs, and then highlight our main observations. Section 5 discusses the threats to validity. In Section 6 we describe related work that acknowledges the threat to validity investigated in this paper. Section 7 concludes the paper and presents future work.

## 2 MUTATION ANALYSIS COST REDUCTION

In a recent systematic review of the Mutation Cost Reduction literature, Pizzoleto et al. [75] collected and reported the most common types of strategies, goals, and metrics used for achieving cost reduction. After analysing 153 peer-reviewed papers published between 1989 and 2018, the authors found that "do fewer" approaches are the most common ones (approximately 74% of all analysed papers), for which the goal is usually to reduce the number of mutants in general or equivalent mutants. Furthermore, among all types of strategies, the number of mutants is the single most used measure and mutant execution speed-up follows as second.

Starting from the literature review of Pizzoleto et al. [75], we collected and then evaluated works that use "do fewer" strategies and that report number of mutants and/or execution time as a cost measure. Then, we performed snowballing on the selected set, and also searched for more papers in popular repositories such as Google Scholar. Based on this process, we identified 75 papers (while Pizzoleto et al. [75] include only 66 papers as "do fewer"). We analysed these papers with respect to two main aspects: the specific type of mutant reduction technique with a simplified categorisation (random mutation, selective mutation, or others), and if they acknowledge the threat investigated in our study (which was not analysed by Pizzoleto et al. [75]). In other words, we are interested in identifying which of those papers report both number of mutants and execution time, and which ones acknowledge the threat investigated in this paper about the accuracy of number of mutants in capturing execution time. Table 1 presents the results of our survey.

We observe that RMS and SM are the two most used strategies among the papers, whereas 39 of them use other kinds of strategies such as Higher Order Mutation (HOM) [35] or Evolutionary Algorithms [26]. 60 papers use either RMS, SM or both, usually comparing them to different types or even newly proposed strategies. These two strategies are also commonly used as a baseline for comparison, serving as "sanity check". Among all papers, 25 different tools are used, targeting C, C++, C#, Java, Python, and other languages.

Out of the 75 papers surveyed, only 15 use execution time to measure cost reduction. Similarly to what was observed by Pizzoleto et al. [75], number of mutants is the most common measure for cost reduction among the papers found. 46 out of 75 (61.3%) of the surveyed papers claim a relation between the number of mutants and cost reduction, and subsequently use it as a direct measure for comparisons in computational cost saving or efficiency gain. 8 out of 75 (10.7%) of the papers use the number of mutants indirectly (e.g. as an efficiency baseline for effectiveness comparison), but do not use it to draw a direct cost comparison. 21 out of 75 (28%) of the papers report the number of mutants, but either compare the computational cost in terms of execution time, or use it for other purposes (e.g. proxy for equivalence determination cost), thus they are not affected by the threat. All in all, 54 out of the 75 (72%) papers we surveyed are susceptible to the threat.

Six papers acknowledge that this is a threat [20, 24, 29, 34, 59, 93] and five of those try to deal with it [20, 24, 29, 34, 59], but none of them evaluated the degree to which this threat affects scientific conclusions of previous work. Only three [20, 24, 34] of the 15 papers that measure the time also acknowledge the threat of using number of mutants as a surrogate for execution time. The other 12 papers implicitly mitigate such threat by simply measuring and reporting the execution time.

If the assumption of equal ratio between the two measures does not hold, then it is a construct validity threat to the 54 papers that do not report execution time and use number of mutants as a proxy for real computational cost. The underlying questions are: what is the difference (if any) between the number of mutants and the true execution time, and how does this affect previous scientific conclusions?

# 3 EMPIRICAL STUDY DESIGN

This section describes the design of the empirical study we conducted to evaluate the differences between the execution time and number of mutants measures in mutation analysis. We do not evaluate the whole cost of mutation testing, i.e., we do not measure the manual or computational cost of determining mutant equivalence, neither the cost of creating and assessing new test cases, because our objective is to focus on the generation and execution of mutants during the mutation analysis task.

During this evaluation, we consider execution time as a direct measurement of cost: the less execution time spent on mutation analysis, the cheaper it is. Therefore, execution time means computational cost in the context of this work. For all RQs, the mutants are executed until they are killed, i.e., we use Partial Mutation as opposed to generating the whole killing matrix. With this approach, we intend to evaluate the cost of mutants in a real-world setting.[2] We executed the experiments on a local machine with an AMD Threadripper 2950X 3.5GHz (4.4GHz boost) CPU, 32GB (3200MHz) of RAM and using Windows 10 v1903.[3]

The goal of this evaluation is to answer the following four RQs:

**RQ1 – Random Mutant Sampling:** What is the difference between execution time and number of mutants when using RMS?

**RQ2 – Statistical Correlations:** What are the statistical correlations (if any) between errors and the properties of the systems under test?

**RQ3 – Strategy Differences:** How the rankings of RMS and SM strategies change when comparing their cost based on number of mutants and execution time?

**RQ4 – Does it Matter?** Can we reproduce previous foundational results on mutant selection?

## 3.1 RQ1 – Random Mutant Sampling

The first RQ is designed to assess the differences between execution time and number of mutants obtained by RMS strategies. By randomly sampling subsets of mutants in different percentages, we can evaluate whether the ratio is 1 for the percentage of sampled mutants to the execution time needed to execute the strategy. In other words, we are investigating whether by randomly reducing the number of mutants by $x\%$ we also reduce the execution time by $x\%$; the ratio assesses how close the empirical data come to this idealised parity. RMS is arguably the most naive type of mutant reduction approach, as mutants are discarded with no particular preference, thus representing an unbiased reduction. This strategy has also served as a baseline comparison for several other work in the literature [24, 53, 94, 97]. Therefore, assessing the error for this type of strategy provides a meaningful overview of what to expect from mutant reduction strategies in general.

To answer this question, we execute RMS strategies with percentages varying from 10% to 90% in steps of 10% (for a total of nine

[2] This is a threat to validity discussed in Section 5.

[3] The replication package and all other information about the programs and setup can be found at: https://doi.org/10.5522/04/11927208.

**Table 1: "Do fewer" papers that use number of mutants or execution time as cost measure. #M: use number of mutants as cost measure. Time: report execution time. RMS, SM and Other: use Random Mutant Sampling, Selective Mutation and other types of "do fewer" strategies, respectively. Ack: papers that acknowledge the validity threat of using number of mutants as cost indicator. Mitigates: if the paper acknowledges, how does it try to mitigate the threat? The last row shows the sum of each column (venue and mutation tool sums present unique values). Papers are sorted by authors.**

| Paper | Year | Venue | Mutation Tool | Metrics | | Strategies | | | Ack | Mitigates |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | #M | Time | RMS | SM | Other | | |
| Al-Hajjaji et al. [1] | 2017 | VACE | Proteum | X | | | X | X | | |
| Ammann et al. [2] | 2014 | ICST | Proteum | X | | X | X | | | |
| Barbosa et al. [4] | 2001 | JSTVR | Proteum | X | | | X | | | |
| Bluemke and Kulesza [5] | 2013 | AISC | MuJava | X | | X | | | | |
| Bluemke and Kulesza [7] | 2014 | AISC | MuJava | X | | | X | | | |
| Bluemke and Kulesza [6] | 2014 | ICSOFT-EA | MuJava | X | | | X | | | |
| Delamaro et al. [12] | 2014 | ICST | Proteum | X | | | X | | | |
| Delamaro et al. [14] | 2014 | ICST | Proteum | X | | | X | | | |
| Delamaro et al. [13] | 2014 | SBES | Proteum | X | | X | X | | | |
| Delgado-Pérez et al. [16] | 2017 | JSTVR | MuCpp | X | | | X | | | |
| Delgado-Pérez et al. [15] | 2017 | IST | MuCpp | X | X | | X | | | |
| Deng et al. [18] | 2013 | ICST | Mothra | X | | | X | | | |
| Derezińska and Rudnik [23] | 2012 | TOOLS | CREAM | X | | X | X | X | | |
| Derezińska [19] | 2013 | AISC | CREAM | X | X | | | X | | |
| Derezińska and Hałas [21] | 2014 | ICSTW | MuPy | X | X | | X | X | | |
| Derezińska and Hałas [22] | 2015 | AISC | MuPy | X | X | | X | X | | |
| Derezińska [20] | 2016 | AISC | CREAM | X | X | | X | X | X | Measures execution time |
| Derezińska and Rudnik [24] | 2017 | FedCSIS | CREAM | X | X | X | | | X | Measures execution time |
| Domínguez-Jiménez et al. [25] | 2011 | IST | GAmera | X | X | | | X | | |
| Gligoric et al. [29] | 2013 | ISSTA | Comutation | X | | X | X | | X | Measures the number of states |
| Gong et al. [30] | 2017 | IST | MuJava | X | X | | | X | | |
| Gopinath et al. [32] | 2015 | ISSRE | PIT | X | | X | X | | | |
| Gopinath et al. [33] | 2016 | ICSE | PIT | X | | X | | X | | |
| Gopinath et al. [31] | 2017 | TR | PIT | X | | X | X | X | | |
| Guizzo et al. [34] | 2020 | TSE | PIT | X | X | X | X | X | X | Measures execution time |
| Harman et al. [36] | 2014 | ASE | Bacterio | X | | | | X | | |
| Iida and Takada [38] | 2017 | Mutation | MuJava | X | | | X | X | | |
| Inozemtseva et al. [39] | 2013 | FSE | GiGAn | X | | X | | X | | |
| Ji et al. [40] | 2009 | SEKE | MuJava | X | | | | X | | |
| Just and Schweiggert [44] | 2014 | JSTVR | Major | X | X | | X | | | |
| Just et al. [43] | 2017 | ISSTA | Major | X | | | X | | | |
| Kaminski and Ammann [45] | 2009 | ICST | TRF-TIF | X | | | X | | | |
| Kaminski et al. [47] | 2011 | IST | TRF-TIF | X | | | X | X | | |
| Kaminski et al. [46] | 2013 | JSS | MUMCUT | X | | | X | | | |
| Kintis et al. [49] | 2010 | APSEC | MuJava | X | | | | X | | |
| Kurtz et al. [50] | 2016 | FSE | Proteum | X | | X | X | | | |
| Lacerda and Ferrari [51] | 2014 | SAST | Proteum/AJ | X | | | X | | | |
| Lima et al. [53] | 2016 | SAST | MuJava | X | | X | X | X | | |
| Ma et al. [54] | 2009 | ETRI | MuJava | X | | | X | | | |
| Madeyski et al. [55] | 2014 | TSE | Judy | X | X | | | X | | |
| Mateo et al. [56] | 2013 | TSE | Bacterio | X | | | | X | | |
| Mresa and Bottaci [59] | 1999 | JSTVR | Mothra | X | | X | X | X | X | Proposes a different metric |
| Nam et al. [60] | 2011 | ICSTW | Javalanche | X | | X | | X | | |
| Siami-Namin and Andrews [80] | 2006 | Mutation | Proteum | X | | | X | | | |
| Namin et al. [61] | 2008 | ICSE | Proteum | X | | | X | | | |
| Nobre et al. [62] | 2012 | IJNCR | Proteum/IM | X | | X | X | X | | |
| Offutt et al. [64] | 1993 | ICSE | Mothra | X | | | X | | | |
| Offutt et al. [63] | 1996 | TOSEM | Mothra | X | | | X | | | |
| de Oliveira et al. [11] | 2013 | CEC | MuJava | X | | X | | X | | |
| Omar and Ghosh [66] | 2012 | ISSRE | HOMAJ | X | | | | X | | |
| Papadakis and Malevris [70] | 2010 | ICSTW | Proteum | X | | X | | X | | |
| Papadakis and Le Traon [68] | 2014 | SAC | Proteum | X | | | X | | | |
| Papadakis and Le Traon [69] | 2013 | JSTVR | Proteum | X | | X | | | | |
| Parsai et al. [71] | 2016 | QRS | LittleDarwin | X | | | | X | | |
| Parsai et al. [72] | 2016 | EASE | LittleDarwin | X | | X | | | | |
| Patrick et al. [74] | 2012 | ICST | MuJava | X | | | | X | | |
| Patrick et al. [73] | 2014 | ICSTW | MuJava | X | | | | X | | |
| Polo et al. [76] | 2008 | JSTVR | MuJava | X | | | | X | | |
| Praphamontripong and Offutt [77] | 2017 | ICSTW | webMuJava | X | | | X | | | |
| Quyen et al. [78] | 2016 | ICEIC | MuSimulink | X | | | | X | | |
| Reuling et al. [79] | 2015 | SPLC | SPLAR | X | | X | | X | | |
| Sridharan and Siami-Namin [82] | 2010 | ISSRE | Proteum | X | | | X | | | |
| Steimann and Thies [83] | 2010 | ICSE | MuJava | X | X | | X | X | | |
| Sun et al. [85] | 2017 | IST | Proteum | X | | X | | X | | |
| Sun et al. [84] | 2017 | CJ | $\mu$BPEL | X | | | X | | | |
| Tuya et al. [86] | 2007 | IST | SQLMutation | X | | | X | | | |
| Untch [87] | 2009 | ACM-SE | Proteum/IM | X | | | | X | | |
| Vincenzi et al. [88] | 2001 | JSTVR | Proteum/IM | X | | | X | | | |
| Wong et al. [90] | 1995 | SQP | Mothra | X | | X | X | | | |
| Wong and Mathur [89] | 1995 | JSS | Mothra | X | | X | X | | | |
| Zhang et al. [95] | 2010 | ICSE | Proteum | X | | X | X | X | | |
| Zhang et al. [94] | 2013 | ASE | Javalanche | X | X | X | X | X | | |
| Zhang et al. [93] | 2014 | ISSRE | Javalanche | X | | | X | | X | |
| Zhu et al. [96] | 2017 | Mutation | PIT | X | X | X | | X | | |
| Zhu et al. [97] | 2018 | ICST | PIT | X | X | X | | X | | |
| Total: 75 | - | 35 | 25 | 75 | 15 | 28 | 47 | 39 | 6 | 5 |

strategies). Setting a $x\%$ step is a common practice when dealing with such kind of strategy [75], since it would be impractical to evaluate all percentages in the 0%–100% range.

In order to generate and execute mutants, we use PIT v1.2.0 [10], one of the most common and fastest Java mutation tools in the literature [52]. PIT has seven default operators and several default execution cost reduction strategies, such as bytecode manipulation, test case prioritisation, and test case filtering based on coverage. We used the default configuration of PIT in order to minimise threats to internal validity (see Section 5).

These nine RMS strategies are evaluated with 10 real-world programs and their original test suites. Table 2 presents data concerning the characteristics of the 10 programs used as subjects to answer all RQs. These systems are also used in other related work on cost reduction in mutation analysis [33, 42, 52, 92]. We selected the latest version of open source programs of various sizes, coverages, mutation score, number of mutants, number of test cases, that could be used with PIT, and that had green test suites (all test cases pass).

Each strategy was executed for 30 independent runs. The execution of multiple independent runs is conducted to minimise the threat related to variations in the mutants' execution, such as the order in which mutants and test cases are executed. It is also a common practice to cater for stochastic behaviour of algorithms [3], in this case, the prioritisation of test cases (default in PIT) that depends on the test case execution times, and the random sampling of mutants done by RMS. Running the mutation analysis process and the strategies multiple times is done to mitigate these uncertainties (more details in Section 5). Hence, in the end of the experiment execution, we have 30 execution times for each strategy per system.

The execution time of a strategy is computed from the start of its execution until the end of the execution of the sampled mutants. Therefore, the execution time measure comprises the whole process of generating, sampling, and then executing the mutants, which reportedly represents the computational cost of the mutation analysis activity [41, 67, 75]. The execution time of each strategy $s$ is then compared to the execution time of the conventional mutation process $CM$ (without any mutant reduction) for a given independent run $r$ as follows:

$$time(s,r) = \frac{time(s_r)}{time(CM_r)} \tag{1}$$

Equation 1 computes the percentage of execution time needed by a strategy to perform the mutation in relation to using the full set of operators and mutants. Similarly, the relative number of mutants is computed using Equation 2. It computes the percentage of mutants in $M^s$ obtained by $s$ in relation to all mutants $M$.

$$number(s) = \frac{|M^s|}{|M|} \tag{2}$$

Since our goal is to analyse the difference (i.e., error) between the two measures, we use a set of traditional error measures to quantify such errors. We compute and report the Mean Signed Difference (MSD) between execution time and number of mutants for each tested RMS strategy across 30 independent runs. The MSD of a given strategy $s$ is computed using Equation 3.

$$MSD(s) = \frac{\sum_{r=1}^{30} time(s,r) - number(s)}{30} \tag{3}$$

MSD indicates the difference (error) between the two measures and its direction (sign). A negative MSD means that $s$ obtained a lower execution time than predicted by the number of mutants measurement, or greater execution time otherwise. For completeness, we also report the Mean Absolute Error (MAE), Mean Squared Error (MSE), and the Mean Absolute Percentage Error (MAPE) for each strategy. MAE, MSE, and MAPE values for a given strategy $s$ are computed using Equations 4–6 respectively.

$$MAE(s) = \frac{\sum_{r=1}^{30} |time(s,r) - number(s)|}{30} \tag{4}$$

$$MSE(s) = \frac{\sum_{r=1}^{30} (time(s,r) - number(s))^2}{30} \tag{5}$$

$$MAPE(s) = \frac{\sum_{r=1}^{30} \frac{|time(s,r) - number(s)|}{time(s,r)}}{30} \tag{6}$$

We also report the average MAE, MSE, and MAPE values for each system and for the complete set of observed data. If the MSD, MAE, MSE, and MAPE values are close to zero, then the ratio between those two measures is close to 1 and the number of mutants can reliably be used as a surrogate for execution time.

## 3.2 RQ2 – Statistical Correlations

Based on the error values found when answering RQ1, we investigate if there are statistical correlations between these errors and system properties such as size of the test suite, testing time, mutation score, and number of mutants (shown in Table 2). For instance, we suspect that there might be a positive correlation between the sizes of the programs under test and the errors between number of mutants and the execution time when sampling mutants for that program. By assessing the correlations of properties such as size of the test suite, number of test cases, and mutation score, we can verify if any of those properties can indicate smaller or larger error when using execution time vs. number of mutants.

We use the Spearman's rank correlation coefficient test [81] to assess the correlations, because we cannot assume that any correlation is necessarily linear. According to Finkelstein et al. [27], a Spearman's $\rho$ threshold of $\pm 0.7$ indicates a strong correlation between the variables, thus following previous work, in this paper we consider as positive correlation any result of $\rho \geq 0.7$ and as negative correlation any result of $\rho \leq -0.7$. Furthermore, we discard any correlation with a $p$-value that is not $\leq 0.05$.

## 3.3 RQ3 – Strategy Differences

The random sampling of mutants gives each mutant an equal probability of being chosen. Observing differences for this kind of strategy gives us an overview of how much the differences can affect an unbiased mutant reduction. However, other strategies such as SM [63, 64] rely on specific properties of operators (e.g., type of mutation and number of generated mutants) to select or discard mutants. Removing operators from the set of operators before generating and executing mutants might affect, not only the cost of generating mutants, but also the execution cost of the reduced set depending on the type of the mutants discarded. For example, a mutant operator that changes the return value of a method might not have the same cost as an operator that changes conditions in *for* loops. Selecting such operators using SM may change the execution

**Table 2: Subject programs. LLOC: number of logical lines of code (executable lines); #T: test suite size of the program; T. LLOC: number of logical lines of test code; Cov: statement and branch coverage percentages; #M: number of mutants generated by PIT; #AM: number of alive mutants after executing T; MS: mutation score of the test suite; Mut. Time: execution time for the conventional mutation process with the test suite T; Test Time: execution time for the test suite against the original program.**

| Program | LLOC | #T | T. LLOC | Cov | #M | #AM | MS | Mut. Time | Test Time |
|---|---|---|---|---|---|---|---|---|---|
| beanutils | 11 645 | 1 293 | 21 808 | 64/64 | 24 349 | 3 652 | 0.85 | 20M 32S | 3S |
| codec | 9 044 | 1 081 | 13 276 | 96/92 | 48 729 | 6 335 | 0.87 | 1H 49M 38S | 26S |
| collections | 28 955 | 24 946 | 34 412 | 86/81 | 61 695 | 7 403 | 0.88 | 1H 47M 43S | 2M 31S |
| lang | 27 646 | 4 119 | 49 146 | 95/91 | 127 824 | 23 008 | 0.82 | 5H 05M 02S | 20S |
| validator | 7 409 | 536 | 8 352 | 86/75 | 18 738 | 3 185 | 0.83 | 24M 59S | 2S |
| jfreechart | 94 203 | 2 174 | 39 883 | 54/46 | 339 301 | 84 825 | 0.75 | 4H 11M 57S | 4S |
| jgrapht | 36 930 | 5 266 | 47 836 | 90/83 | 103 923 | 21 824 | 0.79 | 15H 46M 57S | 01M 16S |
| joda-time | 28 791 | 4 230 | 55 608 | 89/81 | 108 148 | 21 630 | 0.80 | 1H 50M 28S | 10S |
| ognl | 17 012 | 893 | 8 412 | 70/60 | 64 936 | 5 494 | 0.90 | 44M 39S | 27S |
| wire | 1 597 | 79 | 1 933 | 65/58 | 5 169 | 620 | 0.88 | 4M 30S | 1S |

cost results for the same number of mutants when compared to RMS.

We chose to investigate both SM and RMS because they are the two most used mutant reduction strategies in the literature [75]. As we showed in Section 2, 60 out of 75 evaluated papers use either one or the other. Furthermore, they both are commonly used as a baseline for comparison in mutant reduction work [24, 53, 94, 97].

For evaluating the differences between these two types of strategies, we compare the results of the same RMS strategies used in RQ1 to the results of six SM strategies also executed for 30 independent runs. Each SM strategy removes $N$ operators from the set of operators to be executed. To do so, the strategy first sorts the operators in descending order based on the number of mutants they generate and then removes the first $N$ operators from the list. This is a type of Selective Mutation also called Constrained Mutation or $N$-Selective Mutation [75]. We use six strategies because PIT has seven default operators [10], thus we test all six possible strategies of this type (removing $1 \leq N < 7$ of available operators).

Both types of strategies are evaluated using the execution time and number of mutants, as done for RQ1. For assessing the difference, we rank the strategies according to the two measures. Then we assess the disagreement between the two rankings. If the ranks of strategies change, then it indicates that reporting results based on one measure may yield different conclusions than reporting results based on the other measure.

### 3.4 RQ4 – Does It Matter?

Whereas RQ3 asks whether scientific findings might be affected in general, RQ4 targets a specific previous finding to check whether the practice of using surrogate cost measures impacts a specific important foundational finding on which the testing community relies. In order to assess the scientific impact of the differences between number of mutants and execution time, we conduct a set of experiments with the aim of reproducing the seminal work of Offutt et al. [63]. Offutt et al. answered the foundational problem of how to tackle the potentially prohibitive cost of mutation analysis. This was and remains one of the most important problems for the practical application of mutation testing.

In their work, Offutt et al. compare four types of SM strategies: i) Replacement and Statement operators (RS-Selective); ii) Replacement and Expression operators (RE-Selective); iii) Expression and Statement operators (ES-Selective); and iv) Expression operators (E-Selective). One of the key actionable questions asked by the authors was “which mutant operators generate the most mutants?”. This question was answered in 1996 by Offutt et al., and the answer has been relied upon in much of the subsequent work over the past quarter of a century.

Offutt et al. applied the four strategies to 10 small Fortran-77 programs with the Mothra mutation tool [48]. Of course, two and a half decades of change means that we now use different tools and platforms. It is also now customary for empirical software engineering research to use larger programs. Indeed, this is necessary in order to enhance the generalisability of any findings to real-world systems. With this in mind, our reproduction experiment uses programs that are two orders of magnitude larger than the original set in LLOC. However, for due diligence, we also repeated the experiment with the same set of programs that were used in the original work (modulo translation from Fortran-77 to Java).

To answer RQ4, we divided PIT's mutation operators[4] into the three categories defined by Offutt et al. [63] (expression, statement, and replacement) and then applied the RS-, RE-, ES-, and E-Selective strategies. We rank all strategies according to both measurements and then compare the ranks across the 10 real-world programs.

## 4 RESULTS

This section presents the results and answers to the RQs.

### 4.1 Answer to RQ1 – Random Mutant Sampling

Table 3 shows the MSD, MAE, MSE, and MAPE (Equations 3–6) between the obtained percentages for the average execution time and number of mutants. A positive MSD indicates that the strategy costs more in terms of execution time than number of mutants, whereas a negative error means that the strategy executed faster

[4] We used PIT v1.4.12 as opposed to PIT v1.2.0 (RQs 1–3). The latest version has more operators (we use 39 of them), including specific operators used by Offutt et al. [63], thus allowing a more faithful implementation of the original strategies.

**Table 3: RQ1 – MSD of execution time percentage minus percentage of mutants for each strategy. The last three columns present the MAE, MSE and MAPE error indicators for each program, and the last three rows present the MAE, MSE and MAPE for each strategy. The bottom right cells present the MAE, MSE and MAPE for all observations. The stronger the cell highlight, the greater the difference between the two measures.**

| Program | Random Mutant Sampling | | | | | | | | | MAE | MSE | MAPE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 10% | 20% | 30% | 40% | 50% | 60% | 70% | 80% | 90% | | | |
| beanutils | 24.86 | 16.89 | 10.12 | 4.54 | 0.19 | -3.06 | -5.26 | -6.24 | -6.27 | 9.07 | 0.0145 | 21.35% |
| codec | 17.41 | 9.68 | 3.18 | -2.04 | -5.92 | -8.68 | -10.31 | -9.96 | -9 | 8.46 | 0.0091 | 20.46% |
| collections | -1.21 | -8.61 | -14.47 | -18.91 | -21.61 | -22.84 | -22.54 | -20.69 | -17.76 | 16.52 | 0.0328 | 58.17% |
| lang | -3.63 | -11.13 | -17.14 | -21.55 | -24.45 | -25.87 | -25.71 | -23.89 | -20.64 | 19.33 | 0.0433 | 83.11% |
| validator | 21.36 | 13.53 | 7 | 1.74 | -2.13 | -4.75 | -6.2 | -6.29 | -5.03 | 7.56 | 0.0092 | 18.76% |
| jfreechart | -6.18 | -13.09 | -18.09 | -21.13 | -22.17 | -21.41 | -18.83 | -14.41 | -8.06 | 15.93 | 0.0285 | 91.05% |
| jgrapht | 11.92 | 4.48 | -1.62 | -6.15 | -9.3 | -11.31 | -11.63 | -10.53 | -8.19 | 8.35 | 0.0083 | 20.91% |
| joda-time | 5.52 | -2.22 | -8.41 | -12.98 | -16.06 | -18.13 | -18.29 | -17.46 | -14.91 | 12.67 | 0.0197 | 34.65% |
| ognl | -1.51 | -8.98 | -14.78 | -19.3 | -21.97 | -23.12 | -22.71 | -20.77 | -17.11 | 16.7 | 0.0332 | 60.34% |
| wire | 46.25 | 37.89 | 30.54 | 24.04 | 18.26 | 13.2 | 8.94 | 5.38 | 2.63 | 20.79 | 0.0635 | 33.43% |
| MAE | 13.99 | 12.65 | 12.54 | 13.27 | 14.51 | 15.28 | 15.05 | 13.57 | 10.98 | 13.54 | | |
| MSE | 0.0375 | 0.0248 | 0.0223 | 0.0247 | 0.0279 | 0.0297 | 0.0283 | 0.0236 | 0.0169 | | 0.0262 | |
| MAPE | 62.76% | 69.03% | 62.82% | 53.99% | 45.51% | 37.58% | 29.59% | 21.86% | 14.86% | | | 44.22% |

than expected, compared to number of mutants. Errors closer to 0 indicate a ratio closer to 1 for the two measures.

The MSD error between execution time and number of mutants varies widely from 46 to -26. For example, the mean error of 46 was yielded by RMS 10% when executed on `wire`, thus the execution time of this strategy for this system is on average 56% (10% number of mutants + error) of the total time used to perform the mutation analysis procedure with all mutants for this system. Conversely, this same strategy yielded an error of -6 for `jfreechart`, meaning that it takes on average 4% (10% + error) of the total mutation analysis time for this system to generate, select, and then execute the mutants. The MAE value of all observations is 14 and MAPE is 44%. In other words, the number of mutants differs from the execution time on average ±14 absolute percentage units representing a percentage error of ±44%. The relative MAPE variation between the two measures is 19%–91% across all systems studied, and 15%–69% over all strategies studied.

When analysing the relative error (MAPE) between number of mutants and execution time across strategies (last row), we observe a trend: the more mutants sampled, the less relative error between the measurements. Hence, as both measurements increase, the average relative gap between them decreases. This makes sense, since MAPE is computed in relation to the execution time percentage (Equation 6), thus the errors are relatively less meaningful when considering higher percentages, but potentially more impacting for lower sampling/execution times. This is an important observation, since several papers conclude that a lower percentage such as 10% [75, 94, 95] or even a constant sample of 1 000 mutants [32] are sufficient for testing the software. However, as shown by our analysis, MAPE values for low sampling percentages such as 10% can represent 63% of relative error to the real execution cost on average, thus making these sampling strategies cost from approximately a third (`jfreechart`) to 5.62 times more (`wire`) than the inaccurate prediction given by number of mutants. The MAE and MSE values do not catch the same problem, since these two indicators are based on absolute values.

Given the observed errors between the two measures, we can answer RQ1 as follows:

***Answer to RQ1:*** When sampling mutants, the average difference between percentage of sampled mutants and percentage of execution time of the sampled mutants is 44%, i.e., when compared to the whole set of mutants, a subset of mutants obtain a relative execution time that is on average 44% erroneous to the relative number of mutants (estimated cost). Hence, there is a non-trivial error between using number of mutants and execution time as a cost measure.

## 4.2 Answer to RQ2 – Statistical Correlations

There is a series of factors that can relate to the results presented in Section 4.1, such as the test suite of the program, the size of the program, and others. We found two positive correlations between the error metrics and the attributes presented in Table 2: i) MAPE vs. LLOC ($\rho$ = 0.74, p-value = 0.018); and ii) MAPE vs. total number of mutants generated for the program ($\rho$ = 0.76, p-value = 0.016). In other words, the greater the program in LLOC, the greater the MAPE for the sampled mutant sets. Furthermore, when the program grows in LLOC, the number of generated mutants also increases, thus the correlation with MAPE also holds for this property. Hence, the bigger the program, the less reliable the number of mutant measure is in predicting the real cost of the mutant subset.[5]

We did not find any correlation between the remaining programs attributes and the error metrics. All in all, program size and number

[5] We are not implying causation, as we have not tested for causation links.

of mutants that can be generated for a given program are the only observed indications of errors between the number of mutants and execution time measurements. With that in mind, we can answer RQ2 as follows:

***Answer to RQ2:*** The bigger the program, the greater the MAPE (errors) between the number of mutants and execution time measures for sampled subsets ($\rho = 0.74$, p-value = 0.018). The MAPE results of RQ1 are also positively correlated to number of generated mutants ($\rho = 0.76$, p-value = 0.016). In other words, the bigger the program, the less reliable the "number of mutants" measure is at capturing the true mutation cost of a given reduced mutant set.

## 4.3 Answer to RQ3 – Strategy Differences

Table 4 presents the disagreement between rankings for the strategies we considered. There are 15 evaluated strategies: nine RMS and six SM.[6] We can observe that, for 9 out of 10 systems, there is at least one rank change, i.e., a strategy that would be believed to be costlier than others when using number of mutants becomes cheaper when using execution time. There is a difference in 37% of the ranks across all systems.

Interestingly, all rank upgrades (cheaper than expected) from number of mutants to execution time happen for the SM strategies, whereas all rank downgrades (costlier than expected) happen for the RMS strategies. For 5 out of 10 systems, some SM strategies upgraded by two ranks, meaning that they were revealed to be actually cheaper than two other RMS strategies, i.e., the number of mutants of an SM strategy is greater than an RMS strategy in at least 20 percentage points, but the SM strategy is still cheaper in terms of execution time. This is most likely happening for two main reasons:

1. Removing operators with SM from the mutation procedure further reduces the cost of generating mutants, not only executing them, whereas RMS first generates all mutants and then sample them. This means that if an SM and an RMS strategy both select the same number of mutants, the former has the advantage of also reducing the mutant generation time. This reduces the overall computational time for the whole mutation process, and is specially impactful on smaller programs for which the overhead of generating mutants can be as costly as the execution of a small set of mutants.
2. Different mutation operators generate different mutants. Therefore, depending on the removed mutation operators and the program under test, the tester may exclude mutants that are potentially more expensive (as the results of Section 4.4 show). RMS does not distinguish mutants by their operators and sample them uniformly at random, thus it does not benefit from the same feature.

Summarising, there are two major implications about these observations: i) even though a given SM strategy generates more mutants than an RMS strategy, sometimes the general execution cost of using SM to generate mutants and then executing them is lower than the cost of the previously considered cheaper RMS strategy; and ii) when reporting results about which SM strategy is costlier than RMS, if one considers only number of mutants, then they may end up making incorrect scientific conclusions. These are important findings, because they suggest that some scientific conclusions drawn in previous work could be reversed when accounting for this threat to validity. With that in mind, we can answer RQ3 as follows:

***Answer to RQ3:*** When ranking strategies based on execution time instead of number of mutants, we observed a disagreement in 37% of the ranks on average. Rank changes usually favour SM rather than RMS, i.e., even though some SM strategies select more mutants than other RMS strategies, they can actually be cheaper in terms of execution cost.

## 4.4 Answer to RQ4 – Does It Matter?

Figure 1 presents slopegraphs summarising the results for our reproduction study of Offutt et al.'s [63] work. The graphs show the differences observed when using the two different ways of measuring mutation cost: true execution time and its surrogate (number of mutants). As revealed by Figure 1, using true execution time yields an unequivocal finding: RS-Selective is superior for all ten programs. However, using number of mutants (current practice) the results would have been inconclusive.

That is, E-Selective generated fewer mutants for 5 out of 10 programs (i.e., `beanutils`, `collections`, `ognl`, `validator`, and `wire`), whereas RS-Selective generated fewer mutants for the other 5 programs (i.e., `codec`, `lang`, `jfreechart`, `jgrapht`, and `joda-time`). By computing the true execution time, we can see that, in fact, RS-Selective is the cheapest selection strategy for all 10 programs; even for the 5 programs for which E-Selective generated fewer mutants (change of ranks is highlighted in the picture).

For completeness we also collected results for the ten original toy programs used in the 1996 study. We used EvoSuite [28] to generate test cases, similarly to the original study in which Godzilla was used as a test data generation tool. For these ten programs, RS-Selective is the cheapest in terms of number of mutants for 9 of them. Offutt et al. [63] observed that E-Selective was the cheapest for 9 out of the Fortran-77 versions, using Mothra [48]. This difference in the strategy that generates fewer mutants may be explained by the different experimental set-up, i.e., we used a different mutation tool (i.e., PIT) and programming language (i.e., Java). Also, and arguably more importantly, we know from RQ2 (Section 4.2), that results for toy programs will likely differ from those collected for more realistic larger programs. Nevertheless, we include these results for the original ten toy programs for completeness and in order to fully observe due diligence in our reproduction of the original study.

Considering the results presented in this section, we can answer RQ4 as follows:

***Answer to RQ4:*** The reproduction of Offutt et al. [63] work shows that it does, indeed, matter whether experiments use number of mutants or execution time. A key finding, relied upon for 25 years is transformed from "inconclusive" to "unequivocal" by choosing to use the true execution cost in place of its surrogate (number of mutants).

[6] For some systems, strategy S6 is missing because it failed to generate any mutant.

**Table 4: RQ3 – Disagreement between rankings with each measure in ascending order. SM strategies are labelled S$x$ and RMS strategies are labelled R$x$, where $x$ is respectively the number of discarded operators and the sampling percentage. The #M column shows the ranking based on number of mutants and the Time column shows the ranking based on execution time. Highlighted cells represent upgrades (blue with ↑) and downgrades (red with ↓).**

| beanutils | | codec | | collections | | lang | | validator | | jfreechart | | jgrapht | | joda-time | | ognl | | wire | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| #M | Time | #M | Time | #M | Time | #M | Time | #M | Time | #M | Time | #M | Time | #M | Time | #M | Time | #M | Time |
| S5 | S5 | S5 | S5 | S5 | S5 | S6 | S6 | S5 | S5 | S6 | S6 | S6 | S6 | S6 | S6 | S5 | S5 | S5 | S5 |
| S4 | S4 | R10 | ↑ S4 | S4 | S4 | S5 | S5 | S4 | S4 | S5 | S5 | S5 | S5 | S5 | S5 | S4 | S4 | R10 | ↑ S4 |
| S3 | S3 | S4 | ↑ S3 | R10 | ↑ S3 | R10 | ↑ S4 | R10 | ↑ S3 | S4 | S4 | R10 | ↑ S4 | S4 | S4 | R10 | ↑ S3 | S4 | ↑ S3 |
| R10 | ↑ S2 | R20 | ↓ R10 | S3 | ↓ R10 | S4 | ↓ R10 | S3 | ↑ S2 | R10 | R10 | S4 | ↑ S3 | R10 | ↑ S3 | S3 | ↓ R10 | R20 | ↑ S2 |
| S2 | ↓ R10 | S3 | ↓ R20 | R20 | R20 | S3 | S3 | R20 | ↓ R10 | R20 | R20 | R20 | ↓ R10 | S3 | ↓ R10 | R20 | R20 | S3 | ↓ R10 |
| R20 | R20 | R30 | R30 | S2 | S2 | R20 | R20 | S2 | ↓ R20 | S3 | S3 | S3 | ↓ R20 | R20 | ↑ S2 | R30 | ↑ S2 | R30 | ↓ R20 |
| R30 | R30 | R40 | ↑ S2 | R30 | R30 | R30 | ↑ S2 | R30 | R30 | R30 | R30 | R30 | ↑ S2 | S2 | ↓ R20 | S2 | ↓ R30 | R40 | ↓ R30 |
| R40 | ↑ S1 | S2 | ↓ R40 | R40 | R40 | S2 | ↓ R30 | R40 | R40 | R40 | R40 | S2 | ↓ R30 | R30 | R30 | R40 | R40 | S2 | ↓ R40 |
| R50 | ↓ R40 | R50 | R50 | R50 | R50 | R40 | R40 | R50 | R50 | S2 | S2 | R40 | R40 | R40 | R40 | R50 | R50 | R50 | R50 |
| S1 | ↓ R50 | R60 | ↑ S1 | R60 | R60 | R50 | R50 | R60 | ↑ S1 | R50 | R50 | R50 | R50 | R50 | R50 | S1 | S1 | R60 | ↑ S1 |
| R60 | R60 | R70 | ↓ R60 | S1 | S1 | R60 | ↑ S1 | S1 | ↓ R60 | R60 | R60 | S1 | S1 | R60 | ↑ S1 | R60 | R60 | S1 | ↓ R60 |
| R70 | R70 | S1 | ↓ R70 | R70 | R70 | S1 | ↓ R60 | R70 | R70 | S1 | S1 | R60 | R60 | S1 | ↓ R60 | R70 | R70 | R70 | R70 |
| R80 | R80 | R80 | R80 | R80 | R80 | R70 | R70 | R80 | R80 | R70 | R70 | R70 | R70 | R70 | R70 | R80 | R80 | R80 | R80 |
| R90 | R90 | R90 | R90 | R90 | R90 | R80 | R80 | R90 | R90 | R80 | R80 | R80 | R80 | R80 | R80 | R90 | R90 | R90 | R90 |
| | | | | | | R90 | R90 | | | R90 | R90 | R90 | R90 | R90 | R90 | | | | |

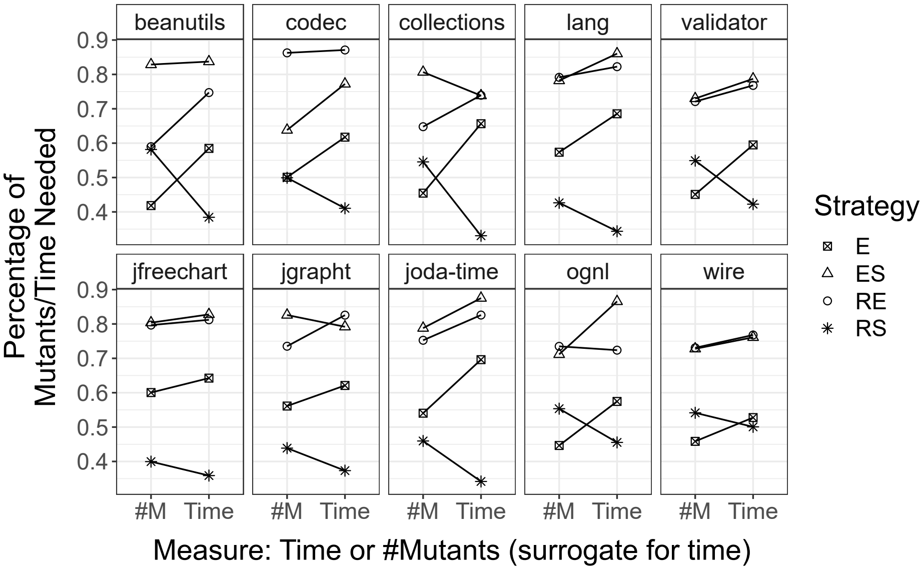

**Figure 1: RQ4: Slopegraph for the results of the reproduction study. The findings enclosed in rectangles fundamentally change when we adopt the true execution time measure in place of the previously used surrogate (number of mutants). The x-axis represents the two cost measures: points on the left represent the number of mutants (Equation 2) and points on the right represent the true execution time (Equation 1). The y-axis represents the percentages (of mutants or time) needed to perform the mutation analysis. Increasing slopes represent strategies that have an execution cost greater than the surrogate, whereas decreasing slopes represent strategies that are cheaper than the surrogate. The E-, RS-, ES-, and RE-Selective strategies are represented each by a different symbol.**

## 5 THREATS TO VALIDITY

*Threats to External Validity:* As happens with most software engineering experiments, the program subjects used in our empirical evaluation might not be representative of the whole software population. In order to minimise this threat, we selected systems of various sizes, types, mutation scores and levels of test coverage. To increase the relevance to the mutation literature we also chose systems used in previous work [33, 42, 52, 92]. Another threat to the external validity is that we only compared results for Java programs and used only one, yet popular, mutation tool (PIT). Caution is required when generalising to other languages and mutation tools.

We limited our experimentation to RMS and SM strategies. We decided to study those two strategies because they are two of the most common mutant reduction strategies in the literature [41, 67, 75] and because they are commonly used as a baseline for newly proposed strategies [53, 59, 94, 95]. In fact, 60 out of 75 papers we surveyed on mutant reduction (Section 2) use either RMS or SM, and 15 of them use both.

The reproduction study conducted to answer RQ4 (Section 4.4) was done in a different setting than that of Offutt et al. [63]. Therefore, our assessment might not generalise to the Mothra tool and to Fortran-77 programs. In order to minimise this threat, we checked the results on Java implementations of the same toy programs used by the authors in their original paper (while also testing the generalisability on real-world large scale Java programs), and we used a similar approach to generate test cases for the testing phase.

*Threats to Internal Validity:* PIT is one of the fastest Java mutation tools in the literature [10], as it performs several cost reduction tasks before generating and executing mutants. For instance, PIT mutates bytecode directly instead of source-code, and it uses test case prioritisation and coverage analysis to avoid running test cases that cannot reach a mutated line. Each of these optimisations may change the execution time of a strategy, yielding an undesirable cause of result variation and introducing bias that would reduce the internal validity of the study. Furthermore, RMS strategies sample different mutant sets in each execution, which can also introduce uncertainty to the results. In order to minimise this threat, we executed multiple independent runs for each of the strategies and computed their respective execution times. The average Coefficient Variation of the execution times across independent runs is 0.0569, which indicates that inter-execution variations are low. Moreover, we do not include non-default optimisations because these might introduce further bias that would reduce the internal validity of the study (for example, parallelisation and history-based mutation might be biased due to thread synchronisation and the use of different mutants in different histories, respectively).

Our experimental design tries to reflect as much as possible the real-world scenario where mutation analysis is applied. For example, we use the "partial mutation" procedure (each mutant is executed until it is killed), as opposed to the execution of all mutants against all test cases (generation of killing matrices). Partial mutation is what we expect an engineer to perform when applying mutation analysis, thus our cost evaluation aims to measure the differences in a real-world scenario.

We grouped PIT's mutation operators in three categories in order to allow the creation of the required SM strategies for the experiments conducted to answer RQ4. During this task, operators might have been misclassified. However, several operators have the same name and functionality as the ones used by Offutt et al. [63], making the grouping mostly straightforward.

*Threats to Construct Validity:* Measuring execution time for the multiple executions of strategies and mutants is not a trivial task. Several factors may add noise to the measurement. In order to mitigate this threat, we used the exact same environment for answering the RQs and performed multiple independent runs.

The indicators we used to assess the differences between number of mutants and execution time are mainly based on error differences and ranks. Other indicators may capture other views on the differences between number of mutants and execution time. Moreover, we carefully applied the statistical tests and verified all the assumptions required.

## 6 RELATED WORK

We consider as related work the papers that acknowledge the threat to validity of using number of mutants as a direct measurement of computational cost. 60 of the 75 (80%) papers surveyed in Section 2 do not consider nor report execution time. However, 15 papers do, correctly, account for the time, being less susceptible to the mentioned threat to validity. Such a threat is observed and directly reported in six papers, three of which address it by also computing and reporting execution time. We describe next what these six papers report in regard to this topic.

In terms of individual mutant cost, Mresa and Bottaci [59] observe that different types of mutants have different costs. Gligoric et al. [29] observe that the cost of mutants varies widely in concurrent code, for which multi-thread scheduling states impact the overall cost of mutants. They also could not tell beforehand how reducing the number of mutants affects the execution of multiple scheduling states in concurrent code. This reported variation in execution costs of different types of mutants is aligned to the results we reported in Sections 4.3 and 4.4 (answer to RQ3 and RQ4): by removing specific operators, and consequently not generating the respective mutant types, we observed changes in ranks where the strategy revealed to be cheaper than expected.

When reducing sets of mutants, Mresa and Bottaci [59] state that "50% reduction in the number of mutants does not imply a 50% reduction in the cost of the test". Derezińska [20] states that, based on her experiments, the lower the number of mutants, the lower the time needed to execute them, but on the other hand the mutant generation process demanded more time. She argues that the number of mutants cannot be overstated as a cost factor, the mutation time does not depend linearly on the number of mutants, and that a real cost measure is the mutation testing time including generation and execution of mutants. Derezińska and Rudnik [24] observe differences between number of mutants and execution time using RMS and state that, in practice, the number of mutants is not the most important factor. The authors suggest that the test suites, number of executed test cases, number of equivalent mutants, and other factors play more important roles in the cost of mutation analysis.

Regarding strategy comparisons, Mresa and Bottaci [59] observe that when considering the number of generated test cases, for instance, the cost reduction of SM is less impactful. Moreover, Zhang et al. [93] acknowledge that mutants have different compilation and execution times depending on the operator, while also suggesting that the investigation of SM scalability considering mutants' cost may incur in different results. Finally, Guizzo et al. [34] observe that the number of mutants is an inaccurate proxy for computational cost of mutation analysis and suggest engineers to use execution time whenever possible.

As other authors have observed [41, 67, 75], the real cost of mutation analysis is more than just the size of the set of mutants: it also depends on the set of test cases [24], number of equivalent mutants [75], whether the test cases were automatically created or designed by a tester [44, 59], program type (e.g., concurrent [29]), and many other factors. In fact, the results our work are aligned to the observations of these papers. Our work shows concrete evidence that the ratio of number of mutants to execution time is not 1 for the programs and strategies we investigated, while also quantifying the error between these two measures and showing that it indeed matters by reproducing a previous work.

## 7 CONCLUSION AND FUTURE WORK

In this paper we investigated the differences between measuring number of mutants and measuring execution time to assess computational cost reduction in mutation analysis. When surveying the mutation cost reduction literature, we found only 15 papers which compare “do fewer” strategies in terms of execution time, and only six of them acknowledge that number of mutants may not be able to fully capture the real cost of mutation analysis.

With our experiments, we show that when using RMS strategies, the error between number of mutants and execution time is on average 44%. Depending on the system, this difference varies from 19% to 91%. These errors are positively correlated to the size of the programs and to the number of mutants that can be generated according to the Spearman’s rank correlation coefficient test, i.e., the errors increase as the programs grow in size. After ranking RMS and SM strategies using the number of mutants and the execution time, we observed that 37% of the ranks would have been incorrectly reported by empirical studies. We further evaluated the impact of these findings by reproducing the work of Offutt et al. [63]. We found that the scientific conclusion as to which strategy is the cheapest changes from “inconclusive” to “unequivocal” when using the true execution time instead of the traditional number of mutants measurement. As a consequence, the findings of previous work are vulnerable to the notable and overlooked threat to validity reported in this paper. Taken together, our results indicate that it is important to consider and report on mutant execution time (and number of mutants) to avoid a potent threat to scientific validity. Previous results on mutation analysis cost need to be revisited in the light of this finding.

In future work we intend to compare the execution cost of HOM and assess if it is indeed the expected $\frac{1}{order}$. Moreover, the real cost of determining mutant equivalence might be assessed by measuring the human effort required for this task as opposed to estimating the cost by using the number of alive mutants.

## ACKNOWLEDGMENTS

Mark Harman, Federica Sarro and Giovani Guizzo are supported by the ERC advanced fellowship under grant number 741278 (https://cordis.europa.eu/project/rcn/210972/factsheet/en) – EPIC: Evolutionary Program Improvement Collaborators. We also would like to thank Microsoft for sponsoring this work with a Microsoft Azure Sponsorship.

## REFERENCES

[1] M. Al-Hajjaji, J. Krüger, F. Benduhn, T. Leich, and G. Saake. 2017. Efficient Mutation Testing in Configurable Systems. In *Proceedings of the IEEE/ACM 2nd International Workshop on Variability and Complexity in Software Design (VACE)*. Buenos Aires, Argentina, 2–8. https://doi.org/10.1109/VACE.2017.3

[2] P. Ammann, M. E. Delamaro, and J. Offutt. 2014. Establishing Theoretical Minimal Sets of Mutants. In *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation*. 21–30.

[3] A. Arcuri and L. Briand. 2014. A Hitchhiker’s guide to statistical tests for assessing randomized algorithms in software engineering. *Software Testing, Verification and Reliability* 24, 3 (2014), 219–250.

[4] E. F. Barbosa, J. C. Maldonado, and A. M. R. Vincenzi. 2001. Toward the determination of sufficient mutant operators for C. *Software Testing, Verification and Reliability* 11, 2 (2001), 113–136.

[5] I. Bluemke and K. Kulesza. 2013. Reduction of Computational Cost in Mutation Testing by Sampling Mutants. In *Proceedings of the 8th International Conference on Dependability and Complex Systems (DepCoS-RELCOMEX)*. Brunów, Poland, 41–51.

[6] I. Bluemke and K. Kulesza. 2014. Reduction in Mutation Testing of Java Classes. In *Proceedings of the 9th International Conference on Software Engineering and Applications (ICSOFT-EA)*. Vienna, Austria, 297–304.

[7] I. Bluemke and K. Kulesza. 2014. Reductions of Operators in Java Mutation Testing. In *Proceedings of the 9th International Conference on Dependability and Complex Systems (DepCoS-RELCOMEX)*. 93–102. https://doi.org/10.1007/978-3-319-07013-1_9

[8] T. Budd and D. Angluin. 1982. Two Notions of Correctness and their Relation to Testing. *Acta Informatica* 8, 1 (1982), 31–45. https://doi.org/10.1007/BF00625279

[9] B. Choi and A. P. Mathur. 1993. High-Performance Mutation Testing. *Journal of Systems and Software* 20, 2 (1993), 135–152.

[10] H. Coles, T. Laurent, C. Henard, M. Papadakis, and A. Ventresque. 2016. PIT: A Practical Mutation Testing Tool for Java (Demo). In *International Symposium on Software Testing and Analysis*. ACM, 449–452.

[11] A. A. L. de Oliveira, C. G. Camilo-Junior, and A. M. R. Vincenzi. 2013. A coevolutionary algorithm to automatic test case selection and mutant in Mutation Testing. In *Congress on Evolutionary Computation*. 829–836.

[12] M. E. Delamaro, L. Deng, V. H. S. Durelli, N. Li, and J. Offutt. 2014. Experimental Evaluation of SDL and One-Op Mutation for C. In *International Conference on Software Testing, Verification and Validation*. 203–212.

[13] M. E. Delamaro, L. Deng, N. Li, V. H. S. Durelli, and A. J. Offutt. 2014. Growing a Reduced Set of Mutation Operators. In *Proceedings of the 28th Brazilian Symposium on Software Engineering (SBES)*. 81–90.

[14] M. E. Delamaro, A. J. Offutt, and P. Ammann. 2014. Designing Deletion Mutation Operators. In *Proceedings of the 7th International Conference on Software Testing, Verification and Validation (ICST)*. 11–20.

[15] P. Delgado-Pérez, I. Medina-Bulo, F. Palomo-Lozano, A. García-Domínguez, and J. J. Domínguez-Jiménez. 2017. Assessment of Class Mutation Operators for C++ with the MuCPP Mutation System. *Information and Software Technology* (2017), 169–184. https://doi.org/10.1016/j.infsof.2016.07.002

[16] P. Delgado-Pérez, S. Segura, and I. Medina-Bulo. 2017. Assessment of C++ Object-Oriented Mutation Operators: A Selective Mutation Approach. *Software Testing, Verification and Reliability* 27, 4-5 (2017), 1630–1649. https://doi.org/10.1002/stvr.1630

[17] R. A. DeMillo, E. W. Krauser, and A. P. Mathur. 1991. Compiler-integrated Program Mutation. In *Proceedings of the 15th IEEE Annual Computer Software and Applications Conference (COMPSAC)*. Tokyo, Japan, 351–356. https://doi.org/10.1109/CMPSAC.1991.170202

[18] L. Deng, A. J. Offutt, and N. Li. 2013. Empirical Evaluation of the Statement Deletion Mutation Operator. In *Proceedings of the 6th International Conference on Software Testing, Verification and Validation (ICST)*. Luxembourg City, Luxembourg, 84–93. https://doi.org/10.1109/ICST.2013.20

[19] A. Derezińska. 2013. A Quality Estimation of Mutation Clustering in C# Programs. In *Proceedings of the 8th International Conference on Dependability and Complex Systems (DepCoS-RELCOMEX)*. Brunów, Poland, 119–129.

[20] A. Derezińska. 2016. Evaluation of Deletion Mutation Operators in Mutation Testing of C# Programs. In *Proceedings of the 11th International Conference on Dependability and Complex Systems (DepCoS-RELCOMEX)*. Brunów, Poland, 97–108. https://doi.org/10.1007/978-3-319-39639-2_9

[21] A. Derezińska and K. Hałas. 2014. Experimental Evaluation of Mutation Testing Approaches to Python Programs. In *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops*. 156–164. https://doi.org/10.1109/ICSTW.2014.24

[22] A. Derezińska and K. Hałas. 2015. Improving Mutation Testing Process of Python Programs. In *Proceedings of the 4th Computer Science On-line Conference Software Engineering in Intelligent Systems (CSOC)*. 233–242. https://doi.org/10.1007/978-3-319-18473-9_23

[23] A. Derezińska and M. Rudnik. 2012. Quality Evaluation of Object-Oriented and Standard Mutation Operators Applied to C# Programs. In *Proceedings of the 50th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS)*. Prague, Czech Republic, 42–57. https://doi.org/10.1007/978-3-642-30561-0_5

[24] A. Derezińska and M. Rudnik. 2017. Evaluation of Mutant Sampling Criteria in Object-Oriented Mutation Testing. In *Proceedings of the Federated Conference on Computer Science and Information Systems (ACSIS)*. Polskie Towarzystwo Informatyczne, Prague, Czech Republic, 1315–1324. https://doi.org/10.15439/2017F375

[25] J. J. Domínguez-Jiménez, A. Estero-Botaro, A. García-Domínguez, and I. Medina-Bulo. 2011. Evolutionary Mutation Testing. *Information and Software Technology* 53 (2011), 1108–1123. Issue 10. https://doi.org/10.1016/j.infsof.2011.03.008

[26] A. E. Eiben and J. E. Smith. 2003. *Introduction to evolutionary computing*. Springer Science & Business Media.

[27] A. Finkelstein, M. Harman, Y. Jia, W. Martin, F. Sarro, and Y. Zhang. 2017. Investigating the relationship between price, rating, and popularity in the Blackberry World App Store. *Information and Software Technology* 87 (2017), 119–139.

[28] G. Fraser and A. Arcuri. 2011. Evolutionary Generation of Whole Test Suites. In *International Conference On Quality Software (QSIC)*. 31–40.

[29] M. Gligoric, L. Zhang, C. Pereira, and G. Pokam. 2013. Selective Mutation Testing for Concurrent Code. In *Proceedings of the 22nd International Symposium on Software Testing and Analysis (ISSTA)*. Lugano, Switzerland, 224–234.

[30] D. Gong, G. Zhang, X. Yao, and F. Meng. 2017. Mutant reduction based on dominance relation for weak mutation testing. *Information and Software Technology* 81 (2017), 82–96.

[31] R. Gopinath, I. Ahmed, M. A. Alipour, C. Jensen, and A. Groce. 2017. Mutation Reduction Strategies Considered Harmful. *IEEE Transactions on Software Reliability* 66, 3 (2017), 854–874. https://doi.org/10.1109/TR.2017.2705662

[32] R. Gopinath, A. Alipour, I. Ahmed, C. Jensen, and A. Groce. 2015. How hard does mutation analysis have to be, anyway?. In *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*. 216–227.

[33] R. Gopinath, M. A. Alipour, I. Ahmed, C. Jensen, and A. Groce. 2016. On The Limits of Mutation Reduction Strategies. *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)* (2016), 511–522.

[34] G. Guizzo, F. Sarro, J. Krinke, and S. R. Vergilio. 2020. Sentinel: A Hyper-Heuristic for the Generation of Mutant Reduction Strategies. *Transactions on Software Engineering* (2020).

[35] M. Harman, Y. Jia, and W. B. Langdon. 2010. A Manifesto for Higher Order Mutation Testing. In *International Conference on Software Testing, Verification, and Validation Workshops*. 80–89.

[36] M. Harman, Y. Jia, P. R. Mateo, and M. Polo. 2014. Angels and monsters: An empirical investigation of potential test effectiveness and efficiency improvement from strongly subsuming higher order mutation. In *International Conference on Automated Software Engineering*. 397–408.

[37] W. E. Howden. 1982. Weak Mutation Testing and Completeness of Test Sets. *IEEE Transactions on Software Engineering* 8, 4 (1982), 371–379.

[38] C. Iida and S. Takada. 2017. Reducing Mutants with Mutant Killable Precondition. In *Proceedings of the 12th International Workshop on Mutation Analysis (Mutation)*. Tokyo, Japan, 128–133. https://doi.org/10.1109/ICSTW.2017.29

[39] L. Inozemtseva, H. Hemmati, and R. Holmes. 2013. Using Fault History to Improve Mutation Reduction. In *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering: New Ideas Track (ESEC/FSE)*. Saint Petersburg, Russia, 639–642.

[40] C. Ji, Z. Chen, B. Xu, and Z. Zhao. 2009. A Novel Method of Mutation Clustering Based on Domain Analysis. In *International Conference on Software Engineering and Knowledge Engineering*, Vol. 9. 422–425.

[41] Y. Jia and M. Harman. 2011. An Analysis and Survey of the Development of Mutation Testing. *IEEE Transactions on Software Engineering* 37, 5 (2011), 649–678. https://doi.org/10.1109/TSE.2010.62

[42] R. Just, D. Jalali, L. Inozemtseva, M. D. Ernst, R. Holmes, and G. Fraser. 2014. Are Mutants a Valid Substitute for Real Faults in Software Testing?. In *Proc. of FSE*. 654–665.

[43] R. Just, B. Kurtz, and P. Ammann. 2017. Inferring Mutant Utility from Program Context. In *Proceedings of the 26th International Symposium on Software Testing and Analysis (ISSTA)*. Santa Barbara, CA, USA, 284–294. https://doi.org/10.1145/3092703.3092732

[44] R. Just and F. Schweiggert. 2015. Higher Accuracy and Lower Run Time: Efficient Mutation Analysis Using Non-redundant Mutation Operators. *Software Testing, Verification and Reliability* 25, 5-7 (2015), 490–507. https://doi.org/10.1002/stvr.1561

[45] G. Kaminski and P. Ammann. 2009. Using a Fault Hierarchy to Improve the Efficiency of DNF Logic Mutation Testing. In *Proceedings of the 2nd International Conference on Software Testing, Verification and Validation (ICST)*. Denver, CO, USA, 386–395.

[46] G. Kaminski, P. Ammann, and A. J. Offutt. 2013. Improving logic-based testing. *Journal of Systems and Software* 86, 8 (2013), 2002–2012. https://doi.org/10.1016/j.jss.2012.08.024

[47] G. Kaminski, U. Praphamontripong, P. Ammann, and A. J. Offutt. 2011. A Logic Mutation Approach to Selective Mutation for Programs and Queries. *Information and Software Technology* 53, 10 (2011), 1137–1152.

[48] K. N. King and A. J. Offutt. 1991. A Fortran Language System for Mutation-based Software Testing. *Software – Practice and Experience* 21, 7 (1991), 685–718.

[49] M. Kintis, M. Papadakis, and N. Malevris. 2010. Evaluating Mutation Testing Alternatives: A Collateral Experiment. In *Proceedings of the 17th Asia-Pacific Software Engineering Conference (APSEC)*. Sydney, Australia, 300–309.

[50] B. Kurtz, P. Ammann, A. J. Offutt, M. E. Delamaro, M. Kurtz, and N. Gökçe. 2016. Analyzing the Validity of Selective Mutation with Dominator Mutants. In *Proceedings of the 24th International Symposium on Foundations of Software Engineering (FSE)*. Seattle, WA, USA, 571–582. https://doi.org/10.1145/2950290.2950322

[51] J. T. S. Lacerda and F. C. Ferrari. 2014. Towards the Establishment of a Sufficient Set of Mutation Operators for AspectJ Programs. In *Proceedings of the 8th Brazilian Workshop on Systematic and Automated Software Testing (SAST)*. Maceio, AL, Brazil, 21–30.

[52] T. Laurent, M. Papadakis, M. Kintis, C. Henard, Y. L. Traon, and A. Ventresque. 2017. Assessing and Improving the Mutation Testing Practice of PIT. In *2017 IEEE International Conference on Software Testing, Verification and Validation*. 430–435.

[53] J. A. P. Lima, G. Guizzo, S. R. Vergilio, A. P. C. Silva, H. L. J. Filho, and H. V. Ehrenfried. 2016. Evaluating Different Strategies for Reduction of Mutation Testing Costs. In *Simpósio Brasileiro de Teste de Software Sistemático e Automatizado*.

[54] Y.-S. Ma, Y.-R. Kwon, and S.-W. Kim. 2009. Statistical Investigation on Class Mutation Operators. *ETRI Journal* 31, 2 (2009), 140–150. https://doi.org/10.4218/etrij.09.0108.0356

[55] L. Madeyski, W. Orzeszyna, R. Torkar, and M. Józala. 2014. Overcoming the Equivalent Mutant Problem: A Systematic Literature Review and a Comparative Experiment of Second Order Mutation. *IEEE Transactions on Software Engineering* 40, 1 (2014), 23–42. https://doi.org/10.1109/TSE.2013.44

[56] P. R. Mateo, M. P. Usaola, and J. L. F. Alemán. 2013. Validating Second-Order Mutation at System Level. *IEEE Transactions on Software Engineering* 39, 4 (2013), 570–587.

[57] A. P. Mathur. 1991. Performance, effectiveness, and reliability issues in software testing. In *Computer Software and Applications Conference*. 604–605.

[58] A. P. Mathur and W. E. Wong. 1994. An empirical comparison of data flow and mutation-based test adequacy criteria. *Software Testing, Verification and Reliability* 4, 1 (1994), 9–31.

[59] E. S. Mresa and L. Bottaci. 1999. Efficiency of Mutation Operators and Selective Mutation Strategies: an Empirical Study. *Software Testing, Verification and Reliability* 9, 4 (1999), 205–232.

[60] J. Nam, D. Schuler, and A. Zeller. 2011. Calibrated Mutation Testing. In *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*. 376–381.

[61] A. S. Namin, J. Andrews, and D. Murdoch. 2008. Sufficient mutation operators for measuring test effectiveness. In *International Conference on Software Engineering*. 351–360.

[62] T. Nobre, S. R. Vergilio, and A. Pozo. 2012. Reducing Interface Mutation Costs with Multiobjective Optimization Algorithms. *International Journal of Natural Computing Research* (2012), 21–40. https://doi.org/10.4018/jncr.2012070102

[63] A. J. Offutt, A. Lee, G. Rothermel, R. H. Untch, and C. Zapf. 1996. An Experimental Determination of Sufficient Mutant Operators. *ACM Transactions on Software Engineering and Methodology* 5, 2 (1996), 99–118.

[64] A. J. Offutt, G. Rothermel, and C. Zapf. 1993. An experimental evaluation of selective mutation. In *International Conference on Software Engineering*. 100–107.

[65] A. J. Offutt and R. H. Untch. 2001. *Mutation Testing for the New Century*. Springer, Chapter Mutation 2000: Uniting the Orthogonal, 34–44.

[66] E. Omar and S. Ghosh. 2012. An Exploratory Study of Higher Order Mutation Testing in Aspect-Oriented Programming. In *Proceedings of the 23th International Symposium on Software Reliability Engineering (ISSRE)*. Dallas, TX, USA, 1–10.

[67] M. Papadakis, M. Kintis, J. Zhang, Y. Jia, Y. Le Traon, and M. Harman. 2017. Mutation Testing Advances: An Analysis and Survey. *Advances in Computers* (2017).

[68] M. Papadakis and Y. Le Traon. 2014. Effective Fault Localization via Mutation Analysis: A Selective Mutation Approach. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing* (Gyeongju, Republic of Korea) *(SAC '14)*. ACM, New York, NY, USA, 1293–1300. https://doi.org/10.1145/2554850.2554978

[69] M. Papadakis and Y. Le Traon. 2015. Metallaxis-FL: mutation-based fault localization. *Software Testing, Verification and Reliability* 25, 5-7 (2015), 605–628.

[70] M. Papadakis and N. Malevris. 2010. An Empirical Evaluation of the First and Second Order Mutation Testing Strategies. In *International Conference on Software Testing, Verification and Validation Workshops*. 90–99.

[71] A. Parsai, A. Murgia, and S. Demeyer. 2016. A model to estimate first-order mutation coverage from higher-order mutation coverage. In *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 365–373.

[72] A. Parsai, A. Murgia, and S. Demeyer. 2016. Evaluating Random Mutant Selection at Class-level in Projects with Non-adequate Test Suites. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering (EASE)*. Limerick, Ireland, 1–10. https://doi.org/10.1145/2915970.2915992

[73] M. Patrick, R. Alexander, M. Oriol, and J. A. Clark. 2014. Probability-based semantic interpretation of mutants. In *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops*. IEEE, 186–195.

[74] M. Patrick, M. Oriol, and J. A. Clark. 2012. MESSI: Mutant Evaluation by Static Semantic Interpretation. In *Proceedings of the 5th International Conference on Software Testing, Verification and Validation (ICST)*. Montreal, QC, Canada, 711–719.

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

[76] M. Polo, M. Piattini, and I. García-Rodríguez. 2009. Decreasing the cost of mutation testing with second-order mutants. *Software Testing, Verification and Reliability* 19, 2 (2009), 111–131.

[77] U. Praphamontripong and A. J. Offutt. 2017. Finding Redundancy in Web Mutation Operators. In *Proceedings of the 12th International Workshop on Mutation Analysis (Mutation)*. Tokyo, Japan, 134–142. https://doi.org/10.1109/ICSTW.2017.30

[78] N. T. H. Quyen, K. T. Tung, L. T. M. Hanh, and N. T. Binh. 2016. Improving Mutant Generation for Simulink Models Using Genetic Algorithm. In *Proceedings of the International Conference on Electronics, Information, and Communications (ICEIC)*. Da Nang, Vietnam, 1–4. https://doi.org/10.1109/ELINFOCOM.2016.7562970

[79] D. Reuling, J. Bürdek, S. Rotärmel, M. Lochau, and U. Kelter. 2015. Fault-based Product-line Testing: Effective Sample Generation Based on Feature-Diagram Mutation. In *Proceedings of the 19th International Conference on Software Product Line (SPLC)*. Nashville, TN, USA, 131–140.

[80] A. Siami-Namin and J. H. Andrews. 2006. Finding Sufficient Mutation Operators via Variable Reduction. In *Proceedings of the 2nd Workshop on Mutation Analysis (Mutation)*. Raleigh, NC, USA, 1–10. https://doi.org/10.1109/MUTATION.2006.7

[81] C. Spearman. 1904. The proof and measurement of association between two things. *American journal of Psychology* 15, 1 (1904), 72–101.

[82] M. Sridharan and A. Siami-Namin. 2010. Prioritizing Mutation Operators based on Importance Sampling. In *Proceedings of the IEEE 21th International Symposium on Software Reliability Engineering (ISSRE)*. San Jose, CA, USA, 378–387.

[83] F. Steimann and A. Thies. 2010. From Behaviour Preservation to Behaviour Modification: Constraint-Based Mutant Generation. In *Proceedings of the 32th International Conference on Software Engineering (ICSE)*. Cape Town, South Africa, 425–434.

[84] C. Sun, L. Pan, Q. Wang, H. Liu, and X. Zhang. 2017. An Empirical Study on Mutation Testing of WS-BPEL Programs. *Computer Journal* 60, 1 (2017), 143–158. https://doi.org/10.1093/comjnl/bxw076

[85] C. Sun, F. Xue, H. Liu, and X. Zhang. 2017. A Path-aware Approach to Mutant Reduction in Mutation Testing. *Information and Software Technology* 81 (2017), 65–81.

[86] J. Tuya, M. J. Suárez-Cabal, and C. de la Riva. 2007. Mutating database queries. *Information and Software Technology* 49, 4 (2007), 398–417.

[87] R. H. Untch. 2009. On Reduced Neighborhood Mutation Analysis Using a Single Mutagenic Operator. In *Proceedings of the 47th Annual Southeast Regional Conference (ACM-SE)*. Clemson, SC, USA, 71–75.

[88] A. M. R. Vincenzi, J. C. Maldonado, E. F. Barbosa, and M. E. Delamaro. 2001. Unit and Integration Testing Strategies for C Programs Using Mutation. *Software Testing, Verification and Reliability* 11, 4 (2001), 249–268.

[89] W. E. Wong and A. P. Mathur. 1995. Reducing the Cost of Mutation Testing: An Empirical Study. *Journal of Systems and Software* 31, 3 (1995), 185–196.

[90] W. E. Wong, A. P. Mathur, and J. C. Maldonado. 1995. *Software Quality and Productivity: Theory, practice, education and training*. Springer, Chapter Mutation versus All-uses: An Empirical Evaluation of Cost, Strength and Effectiveness, 258–265.

[91] M. R. Woodward and K. Halewood. 1988. From weak to strong, dead or alive? an analysis of some mutation testing issues. In *Workshop on Software Testing, Verification, and Analysis*. 152–158.

[92] J. Zhang, Z. Wang, L. Zhang, D. Hao, L. Zang, S. Cheng, and L. Zhang. 2016. Predictive Mutation Testing. In *International Symposium on Software Testing and Analysis*. 342–353.

[93] J. Zhang, M. Zhu, D. Hao, and L. Zhang. 2014. An empirical study on the scalability of selective mutation testing. In *2014 IEEE 25th International Symposium on Software Reliability Engineering*. IEEE, 277–287.

[94] L. Zhang, M. Gligoric, D. Marinov, and S. Khurshid. 2013. Operator-based and random mutant selection: Better together. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 92–102.

[95] L. Zhang, S.-S. Hou, J.-J. Hu, T. Xie, and H. Mei. 2010. Is Operator-based Mutant Selection Superior to Random Mutant Selection?. In *Proc. of the 32Nd ACM/IEEE International Conference on Software Engineering (ICSE'10)*. 435–444.

[96] Q. Zhu, A. Panichella, and A. Zaidman. 2017. Speeding-Up Mutation Testing via Data Compression and State Infection. In *Proceedings of the 12th International Workshop on Mutation Analysis (Mutation)*. Tokyo, Japan, 103–109. https://doi.org/10.1109/ICSTW.2017.25

[97] Q. Zhu, A. Panichella, and A. Zaidman. 2018. An Investigation of Compression Techniques to Speed up Mutation Testing. In *Proceedings of the 11th International Conference on Software Testing, Verification and Validation (ICST)*. Västerås, Sweden, 274–284. https://doi.org/10.1109/ICST.2018.00035