# Does Choice of Mutation Tool Matter?

Rahul Gopinath*, Iftekhar Ahmed†, Amin Alipour‡, Carlos Jensen§, and Alex Groce¶
Department of EECS, Oregon State University
Email: *gopinatr@oregonstate.edu, †ahmed@eecs.orst.edu, ‡alipour@eecs.orst.edu, §cjensen@eecs.orst.edu, ¶agroce@gmail.com

***Abstract*—Though mutation analysis is the primary means of evaluating the quality of test suites, though it suffers from inadequate standardization. Mutation analysis tools vary based on language, when mutants are generated (phase of compilation), and target audience. Mutation tools rarely implement the complete set of operators proposed in the literature, and most implement at least a few domain-specific mutation operators. Thus different tools may not always agree on the mutant kills of a test suite, and few criteria exist to guide a practitioner in choosing a tool, or a researcher in comparing previous results.**

**We investigate an ensemble of measures such as traditional difficulty of detection, strength of minimal sets, diversity of mutants, as well as the information carried by the mutants produced, to evaluate the efficacy of mutant sets. By these measures, mutation tools rarely agree, often with large differences, and the variation due to project, even after accounting for difference due to test suites, is significant. However, the mean difference between tools is very small indicating that no single tool consistently skews mutation scores high or low for all projects.**

**These results suggest that research using a single tool, a small number of projects, or small increments in mutation score may not yield reliable results. There is a clear need for greater standardization of mutation analysis; we propose one approach for such a standardization.**

## I. Introduction

Mutation analysis [1], [2] is one of the best known methods for evaluating the quality of a test suite. Traditional mutation analysis involves exhaustive generation of first order faults, under the assumption that programmers make simple mistakes (the *competent programmer hypothesis*), and any larger faults can be found by tests able to detect simpler faults (the *coupling effect*).

The ability of a test suite to signal a failure for the mutated program is taken as its effectiveness in preventing faults in a program. Mutation analysis has been validated many times in the past. Daran et al. [3] Andrews et al. [4], [5], Do et al. [6], and more recently Just et al. [7] suggest that failures generated by mutants resemble failures from real faults, that mutation analysis is capable of generating faults that resemble real bugs, that the ease of detection for mutants is similar to that for real faults, and that the effectiveness of a test suite in detecting real faults is reflected in its mutation score.

These qualities have led developers to create a large number of mutation analysis tools [8], [9], with different tools for different languages, virtual machines, and introducing mutations at various stages — including design level and specification [10], [11], directly from source code [12], [13], abstract syntax tree [14]–[16], intermediate representation [17], byte code of various virtual machines [18], [19], and even machine code [20]. This also means that there is often no direct translation between modifications carried out at a later phase to an earlier phase[1], or a direct first-order translation between an earlier phase and a later one[2]. Tools may also choose to implement uncommon domain-specific mutation operators such as those targeting multi-threaded [22] code, or using memory-related [23] operators, higher order mutations [13], object-oriented operators [24], [25], or database-targeting operators [26].

Further, it is well known that not all mutants are similar in their fault emulation capabilities [27]–[29], with a number of redundant mutants that tend to inflate the mutation score [30], [31], an unknown number of equivalent mutants that can deflate the mutation score [32]–[36], a large number of easy to detect mutants, and a few stubborn mutants that are hard to find [37].

This presents a predicament for the practicing tester. Which tool should one use to evaluate quality of one's test suites? How good is a new mutation analysis tool? For the researcher, the challenge is how to compare the results of studies evaluated using different tools. This paper aims to provide guidance in how to resolve these dilemmas, and also provide a comparative benchmark for extant tools.

How best to compare the mutants produced by different tools? A naive answer to that question may be to consider the mutation scores obtained by different tools on similar subjects, and assume the mutation tools that produced the lowest scores produced the hardest to find mutants, which are hence better (in some sense). However, this criteria fails to account for problems due to equivalent mutants (which are undetectable, and hence skews in favor of tools that produce a large number of equivalent mutants) and also ignores the variation in detection capabilities of tests. This method is strongly influenced by tests that can kill a large number of mutants, especially since only the first test to kill a mutant is taken into account. Note that any large set of mutants can be reduced to a much smaller number by sampling without significant loss of efficacy [38], while selective mutation has questionable benefits at best [39]. Hence generating a smaller number of mutants is in itself not necessarily a desirable attribute.

Traditionally, the evaluation criteria for a set of mutants (part of a larger set) is to evaluate the effectiveness of a

[1] Very often, a single high level statement is implemented as multiple lower level instructions. Hence a simple change in assembly may not have an equivalent source representation. See Pit switch mutator [21] for an example which does not have a direct source equivalent.

[2] See Pit return values mutator [21] for an example where first order source changes imply much larger bytecode changes.

minimal test suite that is adequate for the subset on the full set. If a minimal test suite that is able to kill all mutants in the subset can also kill all mutants in the full set, the subset is deemed to be equal in effectiveness, or the mutation score of the minimum test suite is taken as its (reduced) effectiveness. The problem with extending it to mutants from different tools is that it requires a superset with which to compare for a standardized effectiveness measure. Since the mutants being compared are not necessarily just first order mutants, the superset is the full set of faults a program can have, which is effectively infeasible to evaluate.

Previous comparisons of mutation tools [9], [40]–[42] have focused on syntactic features, the number of mutants produced and tool support, with few considering the actual *semantic* characteristics of mutants. Mutant semantics assumes new importance in the wake of recent questions regarding the efficacy of various mutation tools [43]–[45].

We benchmark multiple tools using an ensemble of measures from different fields. We use raw mutation scores (without removing non-detected mutants first — which may be subject to skew due to equivalent mutants) and refined mutation scores (removing non-detected mutants which may contain equivalent mutants), and compare the scores produced by different random subsets of test suites. Next we consider the strength[3] of mutants produced, using the minimal set of mutants [46] as a measure of utility of a mutant set, and also a slightly relaxed criterion — using non-subsumed (we call these *surface* mutants) mutants rather than a minimal set for comparison. We then measure the diversity[4] of a set of mutants using statistical measures such as sum of covariance (which we have shown previously [38] to be related to the sample size required for any set of mutants for accurate estimation of mutation score), and the mutual information of mutants (which measures the redundancy of a mutant set). Finally, consider that a test suite is often considered to be one of the ways to specify the program behavior [47]. The quality of the test suite is defined by how much of the specification it is able to accurately provide and verify [48], [49]. Hence a set of mutants of a program may be considered to be representing the program behavior with respect to the possibilities of deviation, and the information carried by a set of mutants is a reasonable measure of quality. We use entropy as a measure of the information content of a set of mutants. We also evaluate whether the number of mutants used has an impact on the scores by using a constant number of mutants (100 mutants sampled 100 times) in each measurement. We further evaluate whether the **phase** of generation (source or bytecode) or the **audience** targeted (industry or research) has an impact on the measures since these are seen as causes for variation [44].

[3] For any set of mutants, the strength of a test suite required to detect them depends on the number of non-redundant mutants within that set. Thus for this paper, we define the *strength* of a set of mutants as the number of non-redundant mutants within that set.

[4] We define diversity of a set of mutants to be how different are two random mutants are from each other in terms of the test cases that kills them. It is related, but not same as the strength of a set of mutants. For example, a set of mutants and test cases that kills them is $\{(m_1, t_1), (m_2, t_2)\}$, while another is $\{(m_1, t_1), (m_2, t_2), (m_3, t_3)\}$. Both have same diversity, but different strength.

Our evaluation suggests that there is often a wide variation in the mutation scores for mutants produced by different tools (low correlation by $R^2$ and $\tau_b$). However, there is very little difference in mean across multiple projects.

Researchers [50]–[53] often rely on the difference of a few percentage points to evaluate the adequacy or superiority of a technique. Hence, for a standardized measure, we expect the variance from the tools to be *much less* than a percentage (ideally it should be exactly the same irrespective of the tool used).

Comparing the quality of mutants produced, there is some variation between tools, with Pit producing the most diverse and strongest set of mutants. However, the difference with other tools is often very small. We also note that project is a significant variable on all measures, generally larger than the impact of tool, phase of generation or target audience, even after accounting for the variability due to difference of test suites (same test suites are used for all tools) and number of mutants. This suggests that the characteristics of individual projects have a larger impact on the mutants produced than the tool used.

It is worrying that the quality of mutants and mutation score varies so much across tools. Hence publications that use a limited number of projects, or those that rely on a few percentage points of improvement suffer a considerable threat to validity. However, the danger is lower if the study uses a larger number of projects and the effect size detected was larger than a few percentage points.

The rest of this paper is organized as follows. The previous research that is related and relevant to ours is given in Section II. Section IV details the different measures we evaluated. Our methodology is given in Section III, and its implications are discussed in Section V. Threats to validity of our research is discussed in Section VI, and we summarize our research in Section VII.

## II. Related Work

The idea of mutation analysis was first proposed by Lipton [1], and its main concepts were formalized by DeMillo et al. in the "Hints" [54] paper. The first implementation of mutation analysis was provided in the PhD thesis of Budd [55] in 1980.

Previous research on mutation analysis suggests that it subsumes different coverage measures, including *statement*, *branch*, and *all-defs* dataflow coverage [56]–[58]. There is also some evidence that the faults produced by mutation analysis are similar to real faults in terms of error trace produced [3] and the ease of detection [4], [5]. Recent research by Just et al. [7] using 357 real bugs suggests that mutation score increases with test effectiveness for 75% of the cases, which was better than the 46% reported for structural coverage.

The validity of mutation analysis rests upon two fundamental assumptions: "The competent programmer hypothesis" – which states that programmers tend to make simple mistakes, and "The coupling effect" – which states that test cases capable of detecting faults in isolation continue to be effective even

TABLE I: Mutation data from Delahaye et al. [9]

Subject programs, test suites and mutation scores

| Project | TestSuite | Judy | Major | Pit | Jumble | Javalanche |
|---|---|---|---|---|---|---|
| codec1.5 | 380 | 78.33 | 70.49 | 91.35 | 84.94 | |
| codec1.7 | 519 | 81.42 | 72.52 | 88.23 | 76.98 | |
| codec1.6 | 1973 | | 72.17 | 85.54 | 79.99 | |
| jdom2 | 1813 | | 71.83 | 82.24 | 44.99 | |
| jopt-simple | 677 | 87.36 | 80.27 | 94.62 | 43.67 | 83.00 |
| json-simple | 3 | 51.85 | 21.37 | 58.52 | 53.90 | 68.00 |

Number of mutants

| Project | Judy | Major | Pit | Jumble | Javalanche |
|---|---|---|---|---|---|
| codec1.5 | 5302 | 5809 | 1826 | 1082 | |
| codec1.7 | 7206 | 6922 | 2651 | 1525 | |
| codec1.6 | | 19472 | 9544 | 4657 | |
| jdom2 | | 6699 | 4978 | 1958 | |
| jopt-simple | 1060 | 674 | 539 | 229 | 100 |
| json-simple | 677 | 1783 | 393 | 141 | 100 |

when faults appear in combination with other faults [54]. Evidence of the coupling effect comes from theoretical analysis by Wah [59], [60], and empirical studies by Offutt [61], [62] and Langdon [63]. While the competent programmer hypothesis is harder to verify, the mean syntactic difference between faults was quantified in our previous work [64].

One theoretical (and practical) difficulty in mutation analysis is identifying equivalent mutants — mutants that are syntactically different, but semantically indistinguishable from the original program, leading to incorrect mutation scores, because in general, identifying equivalent mutants is undecidable. The work on identifying equivalent mutants is generally divided into prevention and detection [36], with prevention focusing on reducing the incidence of equivalent mutants [37] and detection focusing on identifying the equivalent mutants by examining their static and dynamic properties. These include efforts to identify them using compiler equivalence [33], [36], [65] dynamic analysis of constraint violations [35], [66], and coverage [34].

A similar problem is that of redundant mutants [30], where multiple syntactically different mutants represent a single fault, resulting in a misleading mutation score. A number of studies have measured the redundancy among mutants. Ammann et al. [46] compared the behavior of each mutant under all tests and found a large number of redundant mutants. More recently, Papadakis et al. [36] used the compiled representation of programs to identify equivalent mutants. They found that on average 7% of mutants are equivalent and 20% are redundant.

Another fruitful area of research has been reducing the cost of mutation analysis, broadly categorized as *do smarter*, *do faster*, and *do fewer* by Offutt et al. [67]. The *do smarter* approaches include space-time trade-offs, weak mutation analysis, and parallelization of mutation analysis. The *do faster* approaches include mutant schema generation, code patching, and other methods to make the mutation analysis faster as a whole. Finally, the *do fewer* approaches try to reduce the number of mutants examined, and include selective mutation and mutant sampling.

Various studies have tried to tackle the problem of approximating the full mutation score without running a full mutation analysis. The idea of using only a subset of mutants (*do fewer*) was conceived first by Budd [56] and Acree [68] who showed that using just 10% of the mutants was sufficient to achieve 99% accuracy of prediction for the final mutation score. This idea was further investigated by Mathur [69], Wong et al. [70], [71], and Offutt et al. [72] using the Mothra [73] mutation operators for FORTRAN.

Barbosa et al. [29] provides guidelines for operator selection, such as considering at least one operator in each mutation class, and evaluating empirical inclusion among the operators. Lu Zhang et al. [50] compared operator-based mutant selection techniques to random mutant sampling, and found that random sampling performs as well as the operator selection methods. Lingming Zhang et al. [51] compared various forms of sampling such as stratified random sampling based on operator strata, stratified random sampling based on program element strata, and a combination of the two. They found that stratified random sampling when strata were used in conjunction performed best in predicting the final mutation score, and as few as 5% of mutants was a sufficient sample for a 99% correlation with the actual mutation score. The number of samples required for larger projects were found to be still smaller [74], and recently, it was found [38] that $9,604$ mutants were sufficient for obtaining 1% accuracy for 99% of the projects, irrespective of the independence of mutants, or their total population.

A number of researchers have tried to approximate mutation score. Gligoric et al. [52] found that branch coverage is closely correlated with mutation score. Cai et al. [75] found that decision coverage was closely correlated with mutation coverage. Namin et al. [76] found that fault detection ratio was well correlated with block coverage, decision coverage, and two different data-flow criteria. Our own analysis [53] of 232 projects using both manually generated test suites and test suites generated by randoop suggests that, of the different coverage criteria we tested — statement, branch, and path — statement coverage had the closest correlation with mutation score.

Researchers have evaluated different mutation tools in the past. Delahaye et al. [9] compared tools based on fault model (operators used), order (syntactic complexity of mutations), selectivity (eliminating most frequent operators), mutation strength (weak, firm, and strong), and the sophistication of the tool in evaluating mutants. The details of subject programs and mutations are given in Table I[5], and the correlations found (computed by us using the reported data in the paper) are given in Table II.

Our evaluation differs from their research in focusing on the semantic impact of mutants produced by different tools.

[5] Note that the LOC given by Delahaye et al. is ambiguous. The text suggests that the LOC is that of the program. However, checking the LOC of some of the programs such as *jopt-simple* and *commons-lang* suggests that the given LOC is that of the test suite (and it is reported in the table as details of the test suite). Hence we do not include LOC details here.

TABLE II: Correlation for the mutation scores — Data from Delahaye et al. [9]

| | $R^2$ | $\tau_b$ | %Difference $\mu$ | $\sigma$ |
|---|---|---|---|---|
| $Jumble \times Judy$ | 0.15 | -0.33 | | |
| $Jumble \times Major$ | 0.16 | -0.33 | -0.70 | 26.10 |
| $Jumble \times Pit$ | 0.26 | 0.07 | -19.34 | 19.80 |
| $Judy \times Major$ | 1.00 | 1.00 | | |
| $Judy \times Pit$ | 0.98 | 0.67 | | |
| $Major \times Pit$ | 0.96 | 0.60 | -18.64 | 9.70 |

TABLE III: Subject programs, test suite size and mutation scores

| Project | TestSuite | Judy | Major | PIT |
|---|---|---|---|---|
| annotation-cli | 126.00 | 42.42 | 43.27 | 59.38 |
| asterisk-java | 214.00 | 13.54 | 21.54 | 20.64 |
| beanutils | 1185.00 | 50.71 | 42.69 | 56.78 |
| beanutils2 | 680.00 | 59.47 | 52.49 | 61.85 |
| clazz | 205.00 | 24.46 | 39.45 | 30.20 |
| cli | 373.00 | 71.17 | 76.61 | 86.14 |
| collections | 4407.00 | 76.99 | 58.63 | 34.69 |
| commons-codec | 605.00 | 92.72 | 73.52 | 82.66 |
| commons-io | 964.00 | 88.38 | 70.65 | 77.34 |
| config-magic | 111.00 | 55.19 | 29.80 | 60.69 |
| csv | 173.00 | 53.01 | 68.08 | 79.68 |
| dbutils | 239.00 | 44.23 | 65.20 | 47.34 |
| events | 206.00 | 77.14 | 70.03 | 59.95 |
| faunus | 172.00 | 2.55 | 58.65 | 49.07 |
| java-api-wrapper | 125.00 | 14.95 | 84.91 | 76.03 |
| java-classmate | 219.00 | 66.17 | 77.23 | 90.26 |
| jopt-simple | 566.00 | 84.50 | 79.32 | 94.50 |
| mgwt | 103.00 | 40.72 | 6.61 | 8.85 |
| mirror | 303.00 | 58.73 | 74.73 | 75.47 |
| mp3agic | 206.00 | 72.46 | 51.70 | 54.51 |
| ognl | 113.00 | 13.96 | 6.46 | 56.32 |
| pipes | 138.00 | 65.99 | 62.64 | 67.66 |
| primitives | 2276.00 | 93.35 | 71.33 | 35.71 |
| validator | 382.00 | 50.27 | 59.06 | 68.21 |
| webbit | 146.00 | 73.95 | 67.17 | 52.41 |
| $\mu$ | 569.48 | 55.48 | 56.47 | 59.45 |
| $\sigma$ | 930.91 | 26.03 | 21.78 | 21.68 |

## III. Methodology for Assessment

Mutation tools vary along different dimensions. As Ammann suggests in his keynote [77], tools targeting different communities tend to have different priorities, with theoretical completeness a bone of contention between researchers and industry. Further, mutants in different phases of program compilation often do not have first order equivalents in other phases. Hence it is important to ensure that representatives of as many different dimensions of variation are included.

The major avenues of variation are: variation due to mutant distribution in individual projects, variation due to the language used, and variation due to the mutation generation tools used (especially the phase during which the mutants were produced). Unfortunately, the language choice is not orthogonal to other sources of variation. That is, language choice determines the projects, and the tool being used, which makes it difficult to compare different tools, and variation introduced due to projects. Hence, we avoided variation due to languages, and focused solely on Java projects. Keeping the goal of real world projects that best represent real world software, we looked for large Java projects in Github and from Apache foundation, and selected those that could be compiled and tested successfully using multiple mutation analysis tools. Thus we found 25 large Java projects from Github [78] and Apache Software Foundation [79], that had large test suites (Table III).

TABLE IV: Number of mutants by tools in subject programs

| Project | LOC | Judy | Major | PIT |
|---|---|---|---|---|
| annotation-cli | 870.00 | 777.00 | 512.00 | 981.00 |
| asterisk-java | 29477.00 | 12658.00 | 5812.00 | 15476.00 |
| beanutils | 11640.00 | 6529.00 | 4382.00 | 9665.00 |
| beanutils2 | 2251.00 | 990.00 | 615.00 | 2069.00 |
| clazz | 5681.00 | 2784.00 | 2022.00 | 5165.00 |
| cli | 2667.00 | 2308.00 | 1411.00 | 2677.00 |
| collections | 25400.00 | 1006.00 | 10301.00 | 24141.00 |
| commons-codec | 6603.00 | 44.00 | 7362.00 | 9953.00 |
| commons-io | 9472.00 | 164.00 | 6486.00 | 9799.00 |
| config-magic | 1251.00 | 527.00 | 650.00 | 1181.00 |
| csv | 1384.00 | 1154.00 | 991.00 | 1798.00 |
| dbutils | 2596.00 | 1159.00 | 677.00 | 1922.00 |
| events | 1256.00 | 2353.00 | 615.00 | 1155.00 |
| faunus | 9000.00 | 3723.00 | 3771.00 | 9668.00 |
| java-api-wrapper | 1760.00 | 929.00 | 611.00 | 1711.00 |
| java-classmate | 2402.00 | 1423.00 | 952.00 | 2543.00 |
| jopt-simple | 1617.00 | 497.00 | 695.00 | 1790.00 |
| mgwt | 16250.00 | 1394.00 | 6654.00 | 12030.00 |
| mirror | 2590.00 | 1316.00 | 449.00 | 1876.00 |
| mp3agic | 4842.00 | 1272.00 | 4822.00 | 7182.00 |
| ognl | 13139.00 | 8243.00 | 5616.00 | 21227.00 |
| pipes | 3513.00 | 590.00 | 1171.00 | 3001.00 |
| primitives | 11965.00 | 14.00 | 4916.00 | 11312.00 |
| validator | 5807.00 | 3320.00 | 3655.00 | 5846.00 |
| webbit | 5018.00 | 144.00 | 1327.00 | 3707.00 |
| $\mu$ | 7138.04 | 2212.72 | 3059.00 | 6715.00 |
| $\sigma$ | 7471.65 | 2931.64 | 2786.07 | 6369.23 |

Note that we have a much larger set of large sized projects (25 projects with mean 7138 LOC) than previous studies such as Ammann et al. [46], Sridharan et al. [80], Namin et al. [27], Zhang et al. [50], all of which use Siemens test suites and programs (7 projects with mean 312 LOC), Zhang et al. [51] (7 projects with mean 15083 LOC), and Zhang et al. [74] (12 projects with mean 6209 LOC). While our test suites are small (mean=569.48,sd=930.908) in comparison to previous studies using the Siemens test suites[6] —
Ammann et al. [46] (mean=3293.714, sd=1588.226), Sridharan et al. [80] (mean=3115.286, sd=1572.038), Namin et al. [27] (mean=3115.286, sd=1572.038), Zhang et al. [50] (mean=3115.286, sd=1572.038), Zhang et al. [51] (mean=3115.286, sd=1572.038), and Zhang et al. [74] (mean=81, sd=29.061), we believe that the number and size of projects, and the extent of comparison more than makes up for it.

We started our evaluation with the list of all known tools for Java which were available (the first mutation system, JavaMut [82] is no longer available). We also discarded Insure++ [83] which did not actually implement mutation testing [84], [85]. The tools we investigated were PIT [18], Major [16], Judy [41], Javalanche [86], Bacterio [87], MuJava [88], Jumble [19], Jester [89], and Mutator [90]. Our choice of mutation tools for assessment were driven by three key concerns: First, each tool had to provide a way to evaluate the full test suite against each mutant, and obtain the pass or fail status of each mutant against each test. This eliminated Mutator, Jester, and Jumble. Second, we had to be able to get it to work in a distributed cluster, which provided only command line access. Bacterio could not work in a non GUI

[6]The Siemens test suite is a curated test suite by researchers [81] that is at best a questionable representative for real world test suites.

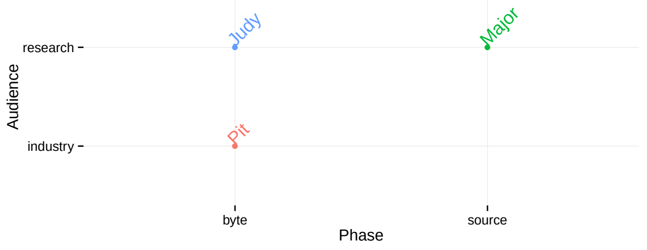


Fig. 1: Tools used for benchmark

environment[7]. While unmodified PIT does not provide the full test kill matrix, we modified PIT to run the full test suite against each mutant (as has been done in numerous studies using PIT), and provide the result. Third, and more important, the tools had to work with a majority of the projects and test suites we had. MuJava could not handle package hierarchies, and an examination of the source code suggested that fixing this shortcoming was non-trivial. Javalanche had large problems in analyzing the projects we chose; while we could get it to work on simple projects, it had problems with newer projects and Junit libraries. A large number of tests caused the JVM to either hang or crash, and eliminating these, the tests that remained were a small fraction of the original test suites. (We also note that Javalanche was last updated on 2012, and is not actively developed anymore. Further, Javalanche adopts only selective mutation, while other tools examined leave that choice to the tester. Hence we removed both MuJava and Javalanche from the benchmark. We note that Javalanche could not complete successfully in a majority of the projects in the previous comparative study by Delahaye [9]).

Thus we were left with three tools: (1) PIT, which uses byte code mutation and is a tool used in industry, (2) Judy, which uses byte code mutation but is mostly used by researchers, and (3) Major, which uses manipulation of the AST, providing source-based mutants, and is primarily used by researchers. Note that as Figure 1 shows, we have a representative for all variations except (source, industry). We also note that Pit and Major are polar opposites along both dimensions. We worked with the authors of each tool to ensure that we had the latest version (Judy 2.1.x, Major 1.1.5). For each tool, we used the settings for the maximum number of operators to mutate. In the case of PIT we extended PIT to provide a more complete set of mutants; a modification which was later accepted to the main line (PIT 1.0). (Note that our intent is not to verify whether historically there were any differences between the tools, but rather, whether such differences still exist — which is what concerns a practitioner).

Unlike other structural coverage measures such as statement, branch or path coverage, there is very little agreement on what constitutes an acceptable set of mutants in mutation analysis. This means that we can expect a wide variation in the number of mutants produced. The mutants produced by each tool for each program is given in Table IV. A histogram of the mutants by each tool is given in Figure 2. Unfortunately,

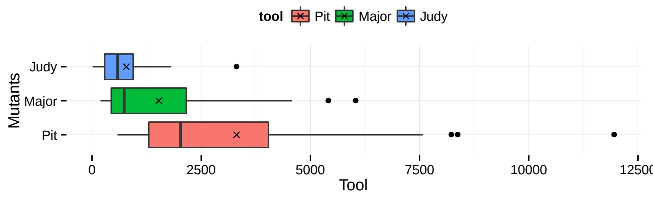


Fig. 2: Number of mutants produced by different tools across all projects in our sample. Judy produces the smallest number, while Pit produces the largest number of mutants. The cross in the center is the mean while the central black line is the median.

this also means that the mutation scores do not necessarily agree as we see in Table III. One of the culprits is the presence of equivalent mutants — mutants that do not produce a measurable semantic variation to the original program. There is no foolproof way of separating equivalent mutants from the merely stubborn mutants at this time. Hence, to avoid skewing the results due to the presence of equivalent mutants, the best one can do is to remove the non detected mutants completely, and assume that if a tool produces a significant number of stubborn mutants that the test suite was inadequate to detect, these would also be reflected in the population of mutants that were killed by a small number of mutants. Hence, we removed the mutants that were not killed by any of the test cases we had, to avoid the impact of equivalent mutants, as done in similar studies [27], [50], [51], [74]. We call the original set the *raw mutants*, and the set of mutants after removing undetected ones the *refined mutant set*.

The sampling was conducted in two dimensions. First, we sampled the test cases of each project randomly in increasingly smaller fractions $\{\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}, \frac{1}{64}\}$. For each fraction, we took 100 samples (that is, using full mutants, but with $\frac{1}{2}$ the number of test cases of the full suite, $\frac{1}{4}$ the number of test cases of the full suite etc.), and the mutation score was computed for each. Second, we sampled 100 mutants each from the refined set of mutants for each project. This was again done 100 times (using the complete test suite for each project). That is, the effective size of sampling was $10,000$ which is large enough for sufficient accuracy as recommended in our previous study on mutant sampling [38].

## IV. Measures

We considered multiple measures that can lead to insights about the characteristics of mutants

### A. *Raw mutation score*

Simple mutation scores are one of the traditional means of comparison between tools, with tools producing low mean scores deemed to have created hard to detect, and hence good mutants. We use different visualizations to inspect the distribution of mutation scores. Figures 3 and 4 show the distribution of mutation score, and the mean values respectively. Note that this is for raw mutants (without removing equivalent mutants

[7]Even though a script mode is available, it still requires GUI to be present, and communication with its authors did not produce any assistance on this point.

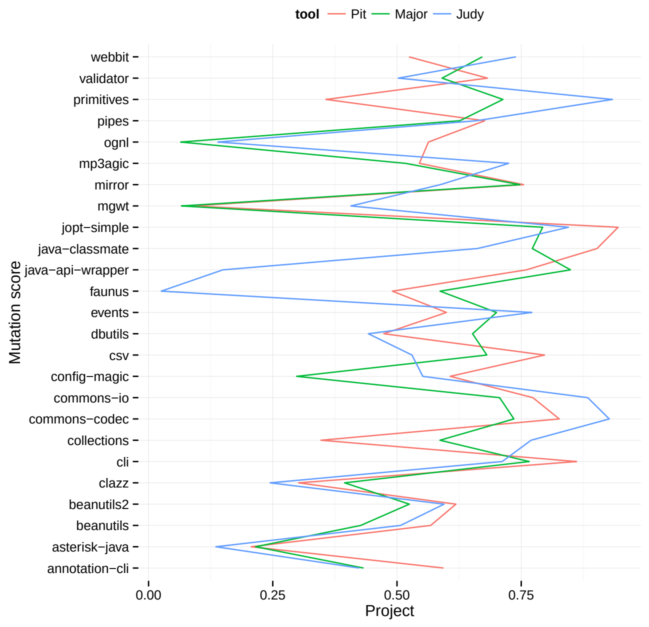


Fig. 3: Distribution of raw mutation score by different tools: mutation scores from different tools rarely agree, and none of the tools produce a consistently skewed score compared to the other tools.

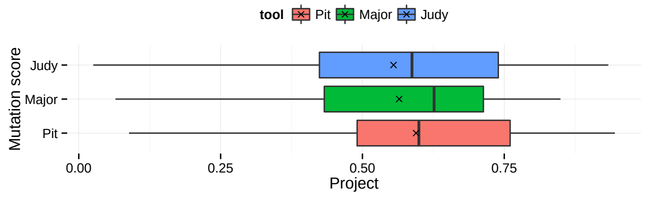


Fig. 4: Mean raw mutation score produced by different tools. Mutation scores produced by different tools are on average not consistently skewed compared to other tools. This could suggest that the strength of mutants produced are comparable on average.

and sampling). The correlations between mutation tools are given in Table V.

We rely on two different correlations here: The first is $R^2$, which suggests how close variables are linearly. $R^2$ (Pearson's correlation coefficient) is a statistical measure of the goodness of fit, that is, the amount of variation in one variable that is explained by the variation in the other. For our purposes, it is the ability of mutation scores produced by one tool to predict the score of the other. We expect $R^2 = 1$ if either A) scores given by both tools for same program are the same, or B) they are always separated by same amount. The Kendall's $\tau_b$ is a measure of monotonicity between variables compared, measuring the difference between concordant and discordant pairs. Kendall's $\tau_b$ rank correlation coefficient is a non-parametric measure of association between two variables. It requires only that the dependent and independent variables

TABLE V: Correlation between mutation tools

| | $R^2$ | $\tau_b$ | %Difference $\mu$ | $\sigma$ |
|---|---|---|---|---|
| $Judy \times Pit$ | 0.37 | 0.27 | -3.97 | 26.93 |
| $Judy \times Major$ | 0.52 | 0.41 | -0.99 | 23.72 |
| $Pit \times Major$ | 0.67 | 0.54 | 2.98 | 17.53 |

(here mutation scores from two different tools) are connected by a monotonic function. It is defined as

$$\tau_b = \frac{\text{concordant pairs} - \text{discordant pairs}}{\frac{1}{2}n(n-1)}$$

$R^2$ and $\tau_b$ provide information along two different dimensions of comparison. That is, $R^2$ can be close to 1 if the scores from both tools are different by a small amount, even if there is no consistency in which one has the larger score. However, such data would result in very low $\tau_b$, since the difference between concordant and discordant pairs would be small. On the other hand, say the mutation scores of one tool is linearly proportional to the test suite, while another tool has a different relation to the test suite – say squared increase. In such a case, the $R^2$ would be low since the relation between the two tools is not linear, while $\tau_b$ would be high. Hence both measures provide useful comparative information. Note that low $\tau_b$ indicates that the troubling situation in which tools would rank two test suites in opposite order of effectiveness is more frequent — this could lead to a change in the results of software testing experiments using mutation analysis to evaluate techniques, just by changing the tool used for measurement.

We also provide the mean difference between the mutation scores measured (denoted by *Difference* $\mu$ in the table), and the standard deviation (denoted by $\sigma$ in the table) for this measurement. The mean difference is important as it provides the effect size — the consistent difference between mutation scores produced by two tools if they have a high correlation and a low spread (standard deviation). That is, even if two tools are found to be different with statistical significance[8], they may not be practically different if the mean difference is in low percentages. Similarly a large spread (standard deviation) indicates that there is a wide variation in the difference, while a small spread indicates that the mean difference is consistent across samples.

A few observations are in order. The first is that the correlation between the mutation scores from different tools is weaker than we expected for a standard measure. However, the mean difference in mutation scores is less than 4% (paired $t - test$ $p < 0.05$). The standard deviation is high, indicating a large spread.

**Q:** *Does the phase of generation or target audience have an impact?*

To answer this question, we rely on *analysis of vari-*

[8]Statistical significance is the confidence we have in our estimates. It says nothing about the effect size. That is, we can be highly confident of a small consistent difference, but it may not be practically relevant.

TABLE VI: Model Fit – mutation score for raw mutants

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 2.62 | 0.11 | 4.12 | 0.0000 |
| phase | 1 | 0.00 | 0.00 | 0.06 | 0.8034 |
| Residuals | 49 | 1.30 | 0.03 | | |

Model fit with phase $R^2$ =0.5

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 2.62 | 0.11 | 4.18 | 0.0000 |
| audience | 1 | 0.02 | 0.02 | 0.77 | 0.3836 |
| Residuals | 49 | 1.28 | 0.03 | | |

Model fit with audience $R^2$ =0.507

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 2.62 | 0.11 | 4.10 | 0.0000 |
| tool | 2 | 0.02 | 0.01 | 0.40 | 0.6713 |
| Residuals | 48 | 1.28 | 0.03 | | |

Model fit with tool $R^2$ =0.497

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 2.62 | 0.11 | 4.20 | 0.0000 |
| Residuals | 50 | 1.30 | 0.03 | | |

Base model fit $R^2$ =0.509

*ance* (*ANOVA*). By running *ANOVA*[9] on a model containing project, phase and audience we determine whether the factor considered has an impact in predicting mutation score. The *ANOVA* equations in general are given in Equation 1, where we compare the ability of each model to predict the variability in the measure being analyzed. In this case the *measure* is *raw mutation score*.

$$\begin{aligned} \mu\{Measure|Project, Phase\} &= Project + Phase \\ \mu\{Measure|Project, Audience\} &= Project + Audience \\ \mu\{Measure|Project, Tool\} &= Project + Tool \\ \mu\{Measure|Project\} &= Project \end{aligned} \tag{1}$$

However, as Table VI shows, *ANOVA* suggests that there is no evidence that a complex model containing either of the variables *phase* or *audience* contributes to a better model fit.

## B. Refined mutation scores after eliminating undetected mutants

The problem with equivalent mutants is that, without identifying them, the true mutation score can not be determined. This means that the premise of mutation analysis — an exhaustive analysis of all faults implies an exhaustive analysis of all failures — can not be fulfilled. Many researchers [27], [50], [51], [74] opt to use a lower bound instead, removing all the mutants that are undetected to remove any influence equivalent mutants have. We take the same route. First, we remove the mutants that were not detected from our pool, leaving mutants that were detected by at least one test case. Next, we sample progressively smaller fractions of test suites,

[9] Analysis of variance — *ANOVA* — is a statistical procedure used to compare the goodness of fit of statistical models. It can tell us if a variable contributes significantly (statistical) to the variation in the dependent variable by comparing against a model that does not contain that variable. If the *p-value* — given in tables as $Pr(>F)$ — is not statistically significant, it is an indication that the variable contributes little to the model fit. Note that the $R^2$ reported is adjusted $R^2$ after adjusting for the effect of complexity of the model due to the number of variables considered.

TABLE VII: Correlation between mutation tools for refined mutants ($\frac{1}{2^x}$ test suite sample)

| Tool | $R^2$ | $\tau_b$ | %Difference $\mu$ | $\sigma$ | TestSuite |
|---|---|---|---|---|---|
| $Judy \times Pit$ | 0.55 | 0.41 | 1.07 | 11.94 | 1/2 |
| $Judy \times Major$ | 0.54 | 0.42 | 0.92 | 12.16 | 1/2 |
| $Pit \times Major$ | 0.66 | 0.44 | -0.16 | 7.74 | 1/2 |
| $Judy \times Pit$ | 0.56 | 0.44 | 3.14 | 15.76 | 1/4 |
| $Judy \times Major$ | 0.63 | 0.47 | 2.22 | 14.70 | 1/4 |
| $Pit \times Major$ | 0.61 | 0.45 | -0.92 | 11.37 | 1/4 |
| $Judy \times Pit$ | 0.52 | 0.42 | 3.07 | 19.78 | 1/8 |
| $Judy \times Major$ | 0.61 | 0.49 | 1.82 | 18.26 | 1/8 |
| $Pit \times Major$ | 0.62 | 0.49 | -1.25 | 12.70 | 1/8 |
| $Judy \times Pit$ | 0.47 | 0.35 | 3.10 | 19.57 | 1/16 |
| $Judy \times Major$ | 0.61 | 0.46 | 1.41 | 17.28 | 1/16 |
| $Pit \times Major$ | 0.63 | 0.50 | -1.69 | 12.63 | 1/16 |
| $Judy \times Pit$ | 0.52 | 0.39 | 0.67 | 17.27 | 1/32 |
| $Judy \times Major$ | 0.63 | 0.48 | -0.69 | 15.64 | 1/32 |
| $Pit \times Major$ | 0.69 | 0.52 | -1.36 | 10.65 | 1/32 |
| $Judy \times Pit$ | 0.51 | 0.38 | 0.35 | 14.22 | 1/64 |
| $Judy \times Major$ | 0.57 | 0.44 | -0.53 | 13.39 | 1/64 |
| $Pit \times Major$ | 0.65 | 0.50 | -0.88 | 9.72 | 1/64 |

with $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}$ and $\frac{1}{64}$ of the original test suite randomly sampled. This is repeated 100 times, and the mutation scores of each sample is taken. The mutation scores using randomly sampled fractions of the original test cases are given in Table VII.

Figure 5 visualises the relationship of mutation scores by different tools. In the figure, the fraction of test suite determines the darkness of the point, and we can see that the light colors cluster near the origin, while darker colors cluster around unity as one would expect (larger fractions of test suite will have higher mutation scores).

The refined mutation scores produced after removing the undetected mutants also tend to follow the same pattern. In Table VII, we see that the maximum $R^2$ is 0.69, with high spread. Further, maximum Kendall's $\tau_b$ is 0.52. This suggests that the mutation scores often do not agree. However the mean difference in mutation score is still less than 3%, which suggests that none of the different tools produce mutants with consistently higher or lower mutation scores than others.

**Q:** *Does the phase of generation or target audience have an impact?*

To answer this question, we rely on *analysis of variance*(*ANOVA*) for which the models are given in Equation 1. The measure to be explained by the models is *refined mutation score*. The analysis of variance in Table VIII suggests that there is no evidence that *phase* and *audience* contribute towards model fit.

## C. Minimal set of mutants

One of the problems with mutation analysis is that a number of faults map to the same failure. This leads to redundant mutants which may inflate the mutation score of a program if any one of them is killed, thus skewing the results. Ammann et al. [46] came up with a practical means of avoiding the effects of redundant mutants. They make use of the concept of dynamically subsuming mutants. A mutant is dynamically subsumed by another if all the tests that detect the former is guaranteed to detect the later — which in effect means that the later is weaker than the former. A minimal test suite for a set of mutants is any test suite from which removing even a

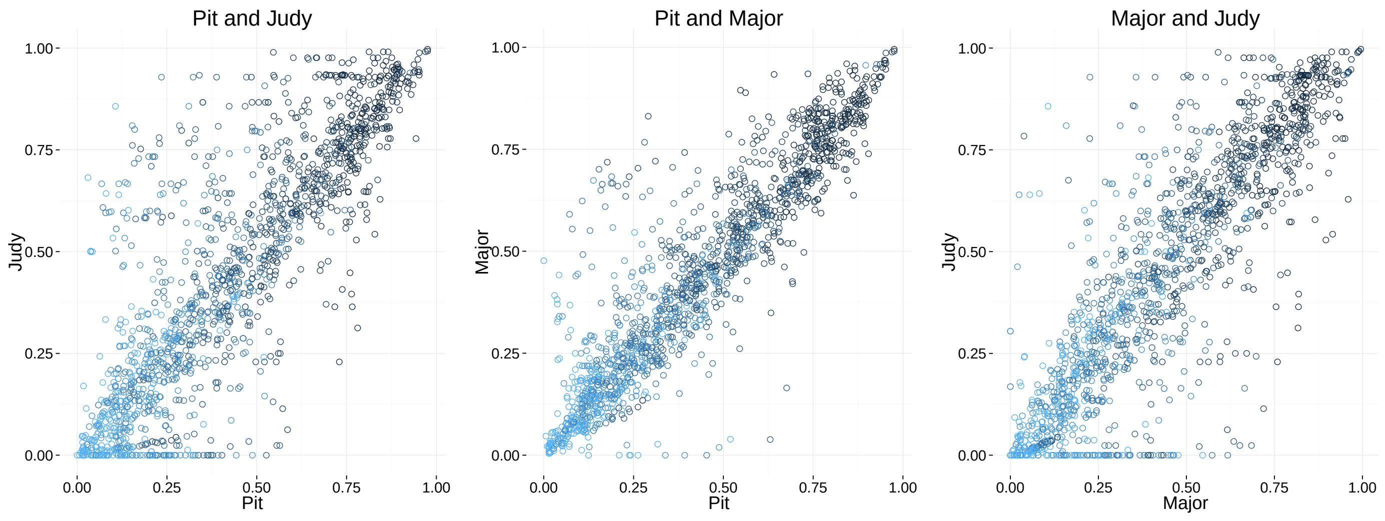


Fig. 5: The mutation scores of different tools plotted against each other for different fractions of the full test suite. The mutation score at larger fractions of the full test suites are colored darker.

TABLE VIII: Model fit with refined mutation score after removing undetected mutants

|  | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 34.15 | 1.42 | 21.33 | 0.0000 |
| phase | 1 | 0.00 | 0.00 | 0.01 | 0.9084 |
| Residuals | 4474 | 298.41 | 0.07 |  |  |

Model fit with phase $R^2$ =0.098

|  | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 34.15 | 1.42 | 21.35 | 0.0000 |
| audience | 1 | 0.22 | 0.22 | 3.25 | 0.0715 |
| Residuals | 4474 | 298.20 | 0.07 |  |  |

Model fit with audience $R^2$ =0.098

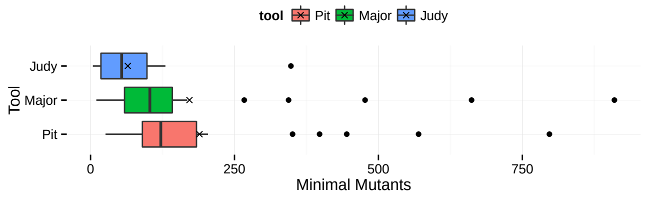
|  | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 34.15 | 1.42 | 21.35 | 0.0000 |
| tool | 2 | 0.27 | 0.14 | 2.04 | 0.1306 |
| Residuals | 4473 | 298.14 | 0.07 |  |  |

Model fit with tool $R^2$ =0.098

|  | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 34.15 | 1.42 | 21.34 | 0.0000 |
| Residuals | 4475 | 298.42 | 0.07 |  |  |

Base model fit $R^2$ =0.098

(a) Full mutant set



(b) 100 mutant sample

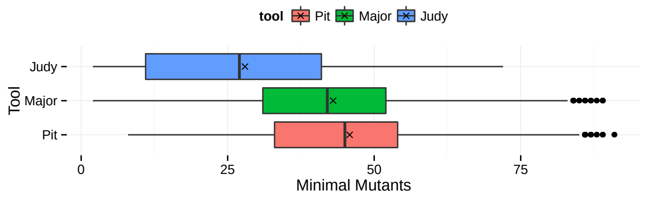


Fig. 6: Minimal mutant sizes between tools

single test case causes the mutation score to drop. Using these two concepts, Ammann et al. [46] defined the minimal set of mutants as the set of non-subsuming mutants that require a minimal test suite that is capable of killing the complete set of mutants. That is, the minimal set of mutants is as effective as the full set of mutants, and hence may be considered as a reasonable measure of the effectiveness of a set of mutants.

Figure 6(a) provides the mean and variance for the size of minimum mutant set for complete projects. The size of minimal mutant set is given in Table IX and a comparison between the tools is given in Table X. As the *number* of minimal mutants determines the **strength** of a set of mutants (the factor of reduction is not the important aspect here), we provide the number rather than the ratio of reduction.

**Impact of minimal set**: Mutant sets with larger sized minimal sets are stronger. Hence tools that produces larger sized minimal sets are better.

Table X suggests that the correlation between different tools is a maximum of 0.96 (between Pit and Major) which is very strong. However, Judy and Major have a very low correlation (0.26). Similarly Kendall's $\tau_b$ is strong (maximum 0.85 between Pit and Major) suggesting a more conforming

TABLE IX: Minimal set of mutants from different mutation tools

| Project | Judy | Major | Pit |
|---|---|---|---|
| annotation-cli | 20.00 | 20.00 | 26.00 |
| asterisk-java | 121.00 | 142.00 | 171.00 |
| beanutils | 348.00 | 344.00 | 398.00 |
| beanutils2 | 67.00 | 105.00 | 145.00 |
| clazz | 18.00 | 59.00 | 49.00 |
| cli | 106.00 | 130.00 | 136.00 |
| collections | 130.00 | 910.00 | 797.00 |
| commons-codec | 4.00 | 267.00 | 351.00 |
| commons-io | 33.00 | 477.00 | 570.00 |
| config-magic | 33.00 | 45.00 | 49.00 |
| csv | 64.00 | 91.00 | 99.00 |
| dbutils | 73.00 | 60.00 | 104.00 |
| events | 21.00 | 10.00 | 30.00 |
| faunus | 7.00 | 103.00 | 122.00 |
| java-api-wrapper | 10.00 | 42.00 | 90.00 |
| java-classmate | 98.00 | 108.00 | 184.00 |
| jopt-simple | 67.00 | 95.00 | 131.00 |
| mgwt | 55.00 | 70.00 | 74.00 |
| mirror | 127.00 | 112.00 | 173.00 |
| mp3agic | 54.00 | 108.00 | 116.00 |
| ognl | 14.00 | 27.00 | 81.00 |
| pipes | 29.00 | 81.00 | 95.00 |
| primitives | 9.00 | 662.00 | 445.00 |
| validator | 102.00 | 168.00 | 204.00 |
| webbit | 9.00 | 59.00 | 90.00 |
| $\mu$ | 64.76 | 171.80 | 189.20 |
| $\sigma$ | 71.94 | 215.33 | 185.83 |

TABLE X: Correlation between minimal set of mutants from different mutation tools

| | $R^2$ | $\tau_b$ | Difference $\mu$ | $\sigma$ |
|---|---|---|---|---|
| $Judy \times Pit$ | 0.34 | 0.34 | -124.44 | 175.05 |
| $Judy \times Major$ | 0.26 | 0.35 | -107.04 | 208.59 |
| $Pit \times Major$ | 0.96 | 0.85 | 17.40 | 62.86 |

relationship between Pit and Major.

**Q:** *Does the phase of generation or target audience have an impact?*

We use *analysis of variance* (*ANOVA*) to answer this question, for which models are given in Equation 1. The measure is the size of minimum mutant set.

The *ANOVA* (Table XI) suggests that *audience* is statistically significant ($p < 0.05$). We note that the full models explain 46.095% (phase), 49.411% (audience) and 57.987% (tool) of the variance ($R^2$). Further the difference explained by *phase* is 1.083%, that explained by *audience* is 4.399% and that explained by *tool* is 12.976% (difference in $R^2$ from a model containing only project).

*1) What is the impact of tools after controlling the number of mutants produced?:* To answer this question, we sample 100 mutants at a time from each project 100 times. Figure 6(b) provides the mean and variance for the size of minimum mutant set for complete projects controlling the number of mutants.

TABLE XI: Model fit with minimum mutant set

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 1440910.19 | 60037.92 | 3.59 | 0.0001 |
| phase | 1 | 33480.54 | 33480.54 | 2.00 | 0.1631 |
| Residuals | 49 | 818380.79 | 16701.65 | | |

Model fit with phase $R^2$ =0.461

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 1440910.19 | 60037.92 | 3.83 | 0.0000 |
| audience | 1 | 83827.44 | 83827.44 | 5.35 | 0.0250 |
| Residuals | 49 | 768033.89 | 15674.16 | | |

Model fit with audience $R^2$ =0.494

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 1440910.19 | 60037.92 | 4.61 | 0.0000 |
| tool | 2 | 227046.96 | 113523.48 | 8.72 | 0.0006 |
| Residuals | 48 | 624814.37 | 13016.97 | | |

Model fit with tool $R^2$ =0.58

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 1440910.19 | 60037.92 | 3.52 | 0.0001 |
| Residuals | 50 | 851861.33 | 17037.23 | | |

Base model fit $R^2$ =0.45

TABLE XII: Minimal set of mutants from different mutation tools (100-sample)

| | $R^2$ | $\tau_b$ | Difference $\mu$ | $\sigma$ |
|---|---|---|---|---|
| $Judy \times Pit$ | 0.49 | 0.35 | -17.86 | 17.97 |
| $Judy \times Major$ | 0.52 | 0.40 | -15.05 | 17.49 |
| $Pit \times Major$ | 0.93 | 0.73 | 2.82 | 7.01 |

Table XII suggests that the correlation between Pit and Major is very strong (0.926), and that the correlation between Judy and Major improved. We find the same with Kendall's $\tau_b$, with strong correlation between Pit and Major (0.73). Finally, spread is large compared to the mean (except for Pit and Major).

**Q:** *Does the phase of generation or target audience have an impact?*

We use Equation 1 for *analysis of variance* between models, where the measure is the size of minimum mutant set, after controlling the number of mutants.

The *ANOVA* (Table XIII) suggests that *phase* and *audience* are indeed statistically significant. We note that the full models explain 65.31% (phase), 69.365% (audience) and 79.272% (tool) of the variance ($R^2$). Further the difference explained by *phase* is 2.178%, that explained by *audience* is 6.233% and that explained by *tool* is 16.14% (difference in $R^2$ from a model containing only project).

## *D. Surface mutants*

One of the problems with the minimal set of mutants from minimal test suites is that it is rather extreme in terms of

TABLE XIII: Model fit with minimal mutant set (100 sample)

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 1812892.77 | 75537.20 | 569.70 | 0.0000 |
| phase | 1 | 62358.74 | 62358.74 | 470.31 | 0.0000 |
| Residuals | 7474 | 990978.33 | 132.59 | | |

Model fit with phase $R^2$ =0.653

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 1812892.77 | 75537.20 | 645.12 | 0.0000 |
| audience | 1 | 178199.56 | 178199.56 | 1521.89 | 0.0000 |
| Residuals | 7474 | 875137.50 | 117.09 | | |

Model fit with audience $R^2$ =0.694

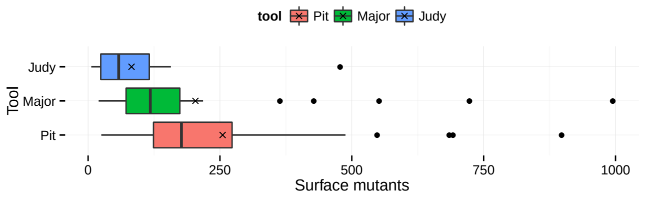
| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 1812892.77 | 75537.20 | 953.47 | 0.0000 |
| tool | 2 | 461297.59 | 230648.79 | 2911.36 | 0.0000 |
| Residuals | 7473 | 592039.48 | 79.22 | | |

Model fit with tool $R^2$ =0.793

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 1812892.77 | 75537.20 | 536.05 | 0.0000 |
| Residuals | 7475 | 1053337.06 | 140.91 | | |

Base model fit $R^2$ =0.631

(a) Full mutant set



(b) 100 mutant sample

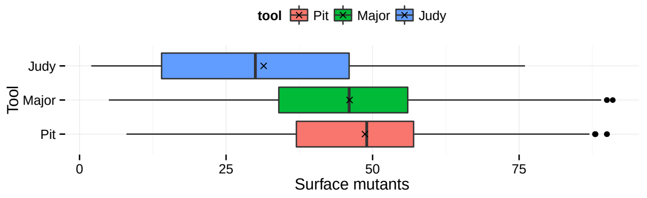


Fig. 7: Surface mutant sizes between tools

TABLE XIV: Surface set of mutants from different mutation tools

| Project | Judy | Major | Pit |
|---|---|---|---|
| annotation-cli | 29.00 | 20.00 | 31.00 |
| asterisk-java | 148.00 | 170.00 | 211.00 |
| beanutils | 478.00 | 428.00 | 548.00 |
| beanutils2 | 80.00 | 105.00 | 149.00 |
| clazz | 24.00 | 73.00 | 64.00 |
| cli | 157.00 | 174.00 | 207.00 |
| collections | 148.00 | 995.00 | 898.00 |
| commons-codec | 6.00 | 364.00 | 488.00 |
| commons-io | 30.00 | 552.00 | 692.00 |
| config-magic | 42.00 | 50.00 | 60.00 |
| csv | 67.00 | 104.00 | 139.00 |
| dbutils | 78.00 | 69.00 | 124.00 |
| events | 25.00 | 22.00 | 25.00 |
| faunus | 8.00 | 126.00 | 179.00 |
| java-api-wrapper | 12.00 | 72.00 | 137.00 |
| java-classmate | 116.00 | 136.00 | 217.00 |
| jopt-simple | 90.00 | 118.00 | 177.00 |
| mgwt | 58.00 | 77.00 | 85.00 |
| mirror | 148.00 | 124.00 | 205.00 |
| mp3agic | 88.00 | 149.00 | 160.00 |
| ognl | 20.00 | 39.00 | 379.00 |
| pipes | 41.00 | 110.00 | 130.00 |
| primitives | 10.00 | 723.00 | 685.00 |
| validator | 140.00 | 218.00 | 273.00 |
| webbit | 15.00 | 69.00 | 114.00 |
| $\mu$ | 82.32 | 203.48 | 255.08 |
| $\sigma$ | 97.04 | 237.65 | 230.22 |

reduction. The total number of mutants in a minimal set of mutants is same as the minimal test suite, and hence is bounded by the size of the test suite. However, the test suite of most programs is much smaller than the complete set of mutants. Hence it may be argued that minimal set of mutants as given by Ammann et al. [46] may miss actual mutants which map to different failures than the ones uniquely checked for by the minimal test suite. To avoid this problem, we relax our definition of minimality in mutants. That is, we remove the requirement that we use the minimal test suite before removing subsumed mutants, and instead use the results from full test suite to obtain non-subsumed mutants. Using such a definition, we have **surface** mutants, given in Table XIV, and the relation between different tools is given in Table XV. The mean surface mutant sizes are plotted in Figure 7(a).

**Impact of surface set**: Mutant set with a larger surface set is stronger, and hence tools that produce larger sized surface sets are better.

Table XV suggests that the $R^2$ correlation between Pit and Major is strong (0.944), while that between Judy and Pit is low (0.274), and that between Judy and Major is low (0.25). Similarly with Kendall's $\tau_b$, which ranges from 0.758 to 0.245. We also note that the values observed are very close to those observed for minimal mutants, which is as we expect given that the surface mutants are obtained by a small modification to the definition of minimal mutants.

**Q:** *Does the phase of generation or target audience have an impact?*

As before, we rely on *analysis of variance* (*ANOVA*) to compare the models (Equation 1 – measure is size of surface mutants).

TABLE XV: Correlation between surface set of mutants from different mutation tools

| | $R^2$ | $\tau_b$ | Difference $\mu$ | $\sigma$ |
|---|---|---|---|---|
| $Judy \times Pit$ | 0.27 | 0.24 | -172.76 | 223.98 |
| $Judy \times Major$ | 0.25 | 0.31 | -121.16 | 233.17 |
| $Pit \times Major$ | 0.94 | 0.76 | 51.60 | 78.48 |

TABLE XVI: Model fit for surface mutants

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 1967892.21 | 81995.51 | 3.19 | 0.0003 |
| phase | 1 | 20160.81 | 20160.81 | 0.78 | 0.3800 |
| Residuals | 49 | 1258614.53 | 25686.01 | | |

Model fit with phase $R^2$ =0.415

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 1967892.21 | 81995.51 | 3.76 | 0.0000 |
| audience | 1 | 209739.21 | 209739.21 | 9.61 | 0.0032 |
| Residuals | 49 | 1069036.13 | 21817.06 | | |

Model fit with audience $R^2$ =0.503

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 1967892.21 | 81995.51 | 4.44 | 0.0000 |
| tool | 2 | 393236.03 | 196618.01 | 10.66 | 0.0001 |
| Residuals | 48 | 885539.31 | 18448.74 | | |

Model fit with tool $R^2$ =0.58

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 1967892.21 | 81995.51 | 3.21 | 0.0003 |
| Residuals | 50 | 1278775.33 | 25575.51 | | |

Base model fit $R^2$ =0.417

TABLE XVII: Correlation between surface set of mutants from different mutation tools (100 sample)

| | $R^2$ | $\tau_b$ | Difference $\mu$ | $\sigma$ |
|---|---|---|---|---|
| $Judy \times Pit$ | 0.47 | 0.34 | -17.28 | 18.12 |
| $Judy \times Major$ | 0.51 | 0.39 | -14.66 | 17.69 |
| $Pit \times Major$ | 0.92 | 0.72 | 2.62 | 7.08 |

TABLE XVIII: Model fit for surface mutants (100 sample)

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 1708095.67 | 71170.65 | 539.73 | 0.0000 |
| phase | 1 | 60356.53 | 60356.53 | 457.72 | 0.0000 |
| Residuals | 7474 | 985549.60 | 131.86 | | |

Model fit with phase $R^2$ =0.641

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 1708095.67 | 71170.65 | 603.93 | 0.0000 |
| audience | 1 | 165130.22 | 165130.22 | 1401.25 | 0.0000 |
| Residuals | 7474 | 880775.91 | 117.85 | | |

Model fit with audience $R^2$ =0.679

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 1708095.67 | 71170.65 | 868.84 | 0.0000 |
| tool | 2 | 433760.06 | 216880.03 | 2647.64 | 0.0000 |
| Residuals | 7473 | 612146.07 | 81.91 | | |

Model fit with tool $R^2$ =0.777

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 1708095.67 | 71170.65 | 508.65 | 0.0000 |
| Residuals | 7475 | 1045906.13 | 139.92 | | |

Base model fit $R^2$ =0.619

The *ANOVA* (Table XVI) suggests that *audience* is statistically significant. We note that the full models explain 41.455% (phase), 50.273% (audience), and 57.951% (tool), of the variance ($R^2$). Further, the difference explained by *phase* is -0.252%, that explained by *audience* is 8.566% and that explained by *tool* is 16.244% (difference in $R^2$ from a model that contained only project).

*1) What is the impact of tools after controlling the number of mutants produced?:* To answer this question, we sample 100 mutants at a time from each project 100 times. The correlation between different tools is given in Table XVII. The mean surface mutant sizes after controlling number of mutants are plotted in Figure 7(b).

As we expect from the values for minimal mutants, controlling the number of mutants have a large impact. Table XVII suggests that the correlation between Pit and Major is very strong (0.917), while that between Judy and Major improved, as did the correlation between Judy and Pit. Similarly for Kendall's $\tau_b$ (0.723 to 0.344).

**Q:** *Does the phase of generation or target audience have an impact?*

We rely on *analysis of variance* (*ANOVA*) (Equation 1 — measure is the size of surface mutants after controlling the number of mutants).

The *ANOVA* (Table XVIII) suggests that *phase* and *audience* are indeed statistically significant. We note that the full models explain 64.094% (phase), 67.911% (audience), and 77.695% (tool) of the variance ($R^2$). Further, the difference explained by *phase* is 2.194%, that explained by *audience* is 6.011%, and that explained by *tool* is 15.795% (difference in $R^2$ from a model that contained only project).

## E. Covariance between mutants

We have shown previously [38] that for mutation analysis, the maximum number of mutants to be sampled for given tolerance has an upper bound provided by the binomial distribution.

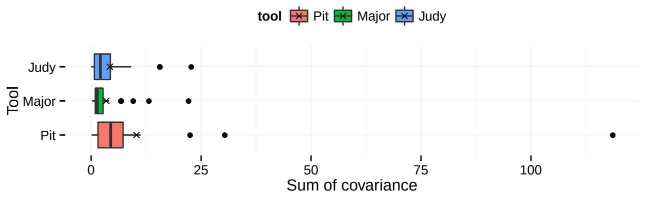
To recap, let the random variable $D_n$ denote the number of detected mutants out of our sample $n$. The estimated mutation score is given by $M_n = \frac{D_n}{n}$. The random variable $D_n$ can be modeled as the sum of all random variables representing mutants $X_{1..n}$. That is, $D_n = \sum_i^n X_i$. The expected value of $E(M_n)$ is given by $\frac{1}{n}E(D_n)$. The variance $V(M_n)$ is given by $\frac{1}{n^2}V(D_n)$, which can be written in terms of component random variables $X_{1..n}$ as:

$$\frac{1}{n^2}V(D_n) = \frac{1}{n^2}\sum_i^n V(X_i) + 2\sum_{i<j}^n Cov(X_i, X_j)$$

Using a simplifying assumption that mutants are more similar to each other than dissimilar, we can assume that

$$2\sum_{i<j}^n Cov(X_i, X_j) >= 0$$

(a) Full mutant set



(b) 100 sample

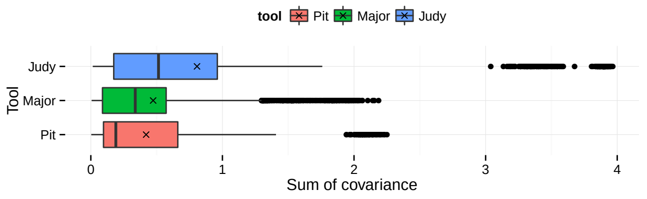


Fig. 8: Covariance between mutants produced

The sum of covariance will be zero when the mutants are independent. That is, the variance of the mutants $V(M_n)$ is strictly greater than or equal to that of a similar distribution of *independent* random variables.

This means that the covariance between mutants determines the size of the sample required. That is, the larger the covariance (or correlation) between mutants, the smaller the diversity. Hence the sum of the covariance can be used to measure the independence of the underlying mutants. The best set of mutants only includes mutants that are completely independent of each other, and the worst set of mutants only includes mutants that are completely dependent (redundant). Figure 8(a) shows the mean covariance across the different tools.

**Impact of sum of covariance**: Mutant sets with smaller sum of covariance are more independent compared to other sets of similar size, and hence tools that produce mutants with smaller sum of covariance are better.

Table XIX provides the sum of covariance for different projects, and Table XX provides the correlation between the sum of covariance from different tools. Table XX suggests that in terms of sum of covariance, Judy and Pit are in closest agreement (0.452), with medium correlation, while the correlation between other tools is low. However, Kendall's $\tau_b$ indicates a medium correlation between all tools (0.427, 0.433, 0.567).

**Q:** *Does the phase of generation or target audience have an impact?*

We rely on *analysis of variance*(*ANOVA*) (Equation 1 — measure is the *sum of covariance*)

The *ANOVA* (Table XXI) suggests that *audience* is statistically significant. We note that the full models explain 13.754% (phase), 18.141% (audience), and 16.517% (tool) of the variance ($R^2$). Further, the difference explained by *phase* is 0.618%, *audience* is 5.005%, and *tool* is 3.381% (difference in $R^2$ from a model that contained only project). We note that project is not statistically significant.

TABLE XIX: Covariance different mutation tools

| Project | Judy | Major | Pit |
|---|---|---|---|
| annotation-cli | 4.69 | 2.71 | 11.27 |
| asterisk-java | 0.74 | 0.44 | 1.36 |
| beanutils | 5.13 | 1.31 | 3.77 |
| beanutils2 | 3.42 | 1.25 | 1.56 |
| clazz | 22.76 | 6.86 | 2.22 |
| cli | 1.93 | 1.04 | 4.60 |
| collections | 0.10 | 0.20 | 0.13 |
| commons-codec | 0.41 | 2.34 | 2.01 |
| commons-io | 0.04 | 0.28 | 0.38 |
| config-magic | 1.59 | 0.97 | 4.44 |
| csv | 4.38 | 2.69 | 5.13 |
| dbutils | 0.80 | 1.40 | 0.73 |
| events | 15.60 | 4.05 | 5.32 |
| faunus | 0.12 | 6.79 | 8.00 |
| java-api-wrapper | 0.34 | 9.59 | 7.27 |
| java-classmate | 4.15 | 1.68 | 7.19 |
| jopt-simple | 3.86 | 2.69 | 9.92 |
| mgwt | 1.53 | 0.54 | 1.22 |
| mirror | 2.64 | 0.26 | 2.04 |
| mp3agic | 9.09 | 22.17 | 30.40 |
| ognl | 15.65 | 1.50 | 118.63 |
| pipes | 2.10 | 1.00 | 1.57 |
| primitives | 0.03 | 0.20 | 0.15 |
| validator | 1.02 | 0.94 | 5.90 |
| webbit | 4.16 | 13.15 | 22.53 |
| $\mu$ | 4.25 | 3.44 | 10.31 |
| $\sigma$ | 5.73 | 5.05 | 23.64 |

TABLE XX: Correlation of sum of covariance between different mutation tools

| | $R^2$ | $\tau_b$ | Difference $\mu$ | $\sigma$ |
|---|---|---|---|---|
| $Judy \times Pit$ | 0.45 | 0.43 | -6.06 | 21.65 |
| $Judy \times Major$ | 0.29 | 0.43 | 0.81 | 6.45 |
| $Pit \times Major$ | 0.19 | 0.57 | 6.87 | 23.23 |

*1) What is the impact of tools after controlling the number of mutants produced?:* To answer this question, we sample 100 mutants at a time from each project 100 times. Figure 8(b) shows the mean covariance across the different tools after controlling the number of mutants.

Table XXII provides the correlation between sum of covariance from different tools after controlling the number of mutants. We see that both $R^2$ and $\tau_b$ increases across all the tools, with the Pit and Major correlation being highest (0.681), with medium correlation. Similar improvement is also

TABLE XXI: Model fit for covariance

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 6407.53 | 266.98 | 1.48 | 0.1229 |
| phase | 1 | 245.55 | 245.55 | 1.36 | 0.2495 |
| Residuals | 49 | 8858.36 | 180.78 | | |

Model fit with phase $R^2$ =0.138

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 6407.53 | 266.98 | 1.56 | 0.0945 |
| audience | 1 | 696.10 | 696.10 | 4.06 | 0.0495 |
| Residuals | 49 | 8407.81 | 171.59 | | |

Model fit with audience $R^2$ =0.181

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 6407.53 | 266.98 | 1.53 | 0.1057 |
| tool | 2 | 704.28 | 352.14 | 2.01 | 0.1448 |
| Residuals | 48 | 8399.62 | 174.99 | | |

Model fit with tool $R^2$ =0.165

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 6407.53 | 266.98 | 1.47 | 0.1261 |
| Residuals | 50 | 9103.91 | 182.08 | | |

Base model fit $R^2$ =0.131

TABLE XXIII: Model fit for covariance (100 sample)

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 2320.14 | 96.67 | 547.48 | 0.0000 |
| phase | 1 | 16.69 | 16.69 | 94.53 | 0.0000 |
| Residuals | 7174 | 1266.77 | 0.18 | | |

Model fit with phase $R^2$ =0.647

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 2320.14 | 96.67 | 564.47 | 0.0000 |
| audience | 1 | 54.82 | 54.82 | 320.09 | 0.0000 |
| Residuals | 7174 | 1228.64 | 0.17 | | |

Model fit with audience $R^2$ =0.658

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 2320.14 | 96.67 | 611.28 | 0.0000 |
| tool | 2 | 149.07 | 74.54 | 471.31 | 0.0000 |
| Residuals | 7173 | 1134.39 | 0.16 | | |

Model fit with tool $R^2$ =0.684

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 2320.14 | 96.67 | 540.43 | 0.0000 |
| Residuals | 7175 | 1283.46 | 0.18 | | |

Base model fit $R^2$ =0.643

TABLE XXII: Correlation of sum of covariance between different mutation tools (100 sample)

| | $R^2$ | $\tau_b$ | Difference $\mu$ | $\sigma$ |
|---|---|---|---|---|
| $Judy \times Pit$ | 0.50 | 0.61 | 0.34 | 0.86 |
| $Judy \times Major$ | 0.60 | 0.66 | 0.29 | 0.80 |
| $Pit \times Major$ | 0.68 | 0.61 | -0.05 | 0.41 |

seen over Kendall's $\tau_b$ — highest is between Judy and Major (0.661).

**Q:** *Does the phase of generation or target audience have an impact?*

We rely on *analysis of variance* (*ANOVA*) (Equation 1 — measure is the *sum of covariance* after controlling the number of mutants).

The *ANOVA* (Table XXIII) suggests that both *phase* and *audience* are statistically significant. We note that the full models explain 64.725% (phase), 65.786% (audience), and 68.407% (tool) of the variance ($R^2$). Further, the difference explained by *phase* is 0.46%, *audience* 1.522%, and *tool* 4.142% (difference in $R^2$ from a model that contained only project). We note that project is statistically significant once number of mutants is controlled.

## F. Mutual information (Total correlation) between mutants

Covariance between mutants is a measure of the quality of mutants. The more independent mutants are, the lower the covariance. There is a measure from information theory that lets us evaluate the redundancy of mutants more directly — *mutual information*.

The *mutual information* of a variable is defined as the reduction in uncertainty of a variable due to knowledge of another. That is, given two variables $X$ and $Y$, the redundancy between them is estimated as:

$$I(X;Y) = I(Y;X) = \sum_{y \in Y} \sum_{x \in X} p(x,y) log\left(\frac{p(x,y)}{p(x)p(y)}\right)$$

To extend this to a set of mutants, we use one of the multivariate generalizations of mutual information proposed by Watanabe [91] — *multi information* also called *total correlation*. The important aspects of *multi information* that are relevant to us are that it is well behaved — that is it allows only positive values, and is zero only when all variables are completely independent. The multi information for a set of random variables $x_i \in X$ is defined formally as:

$$C(X_1..X_n) = \sum_{x_1 \in X_1} .. \sum_{x_n \in X_n} p(x_1..x_n) log\left(\frac{p(x_1..x_n)}{p(x_1)..p(x_n)}\right)$$

Figure 9(a) shows the mean multi information between mutants produced.

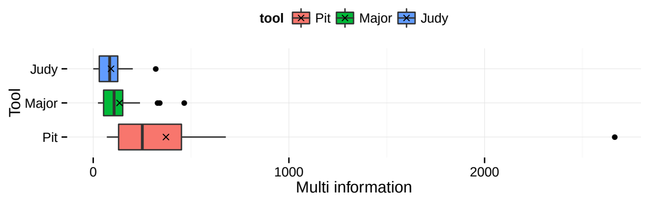
**Impact of multi information**: Mutant sets with smaller multi information have more diverse mutants compared to similar sized mutant sets. Hence tools that produce mutants with smaller multi information are better.

Table XXIV shows the multi-information of different tools across projects, and Table XXV provides the correlation of multi-information by different tools.

Table XXV suggests that the correlation between different tools is rather weak in terms of both $R^2$ (0.286 to -0.063) and $\tau_b$ (0.433 to -0.14).

**Q:** *Does the phase of generation or target audience have an impact?*

(a) Full mutant set



(b) 100 sample

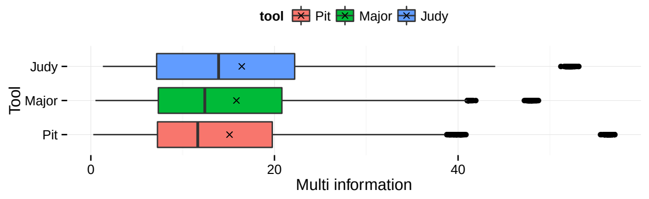


Fig. 9: Multi Information

We rely on *analysis of variance* (*ANOVA*) (Equation 1 — measure is the *multi information*).

The *ANOVA* (Table XXVI) suggests *audience* is the only statistically significant variable. We note that the full models explain -0.015% (phase), 18.741% (audience), and 17.519% (tool) of the variance ($R^2$). Further the difference explained by *phase* is -64.28%, *audience* is -45.524%, and by *tool* is -46.745% (difference in $R^2$ from a model that contained only project).

*1) What is the impact of tools after controlling the number of mutants produced?:* To answer this question, we sample 100 mutants at a time from each project 100 times. Figure 9(b) shows the mean multi information between mutants produced, after controlling the number of mutants.

Table XXVII provides the correlation of multi-information by different tools after controlling number of mutants. Table XXVII suggests that the correlation between different tools improves across all tools in terms of both $R^2$ (0.865 to 0.785) and $\tau_b$ (0.594 to 0.53).

**Q:** *Does the phase of generation or target audience have an impact?*

We rely on *analysis of variance* (*ANOVA*) (Equation 1 — measure is the *multi information* after controlling the number of mutants).

The *ANOVA* (Table XXVIII) suggests both *phase*, and *audience* are statistically significant. We note that the full models explain 89.572% (phase), 89.551% (audience), and 89.577% (tool) of the variance ($R^2$). Further the difference explained by *phase* is 0.06%, *audience* is 0.039%, and *tool* is 0.065% (difference in $R^2$ from a model that contained only project).

TABLE XXIV: Multi information different mutation tools

| Project | Judy | Major | Pit |
|---|---|---|---|
| annotation-cli | 90.94 | 58.00 | 173.20 |
| asterisk-java | 130.34 | 96.99 | 297.53 |
| beanutils | 320.20 | 150.88 | 461.81 |
| beanutils2 | 84.11 | 37.02 | 114.10 |
| clazz | 159.27 | 227.02 | 129.75 |
| cli | 202.29 | 127.75 | 360.27 |
| collections | 13.81 | 106.72 | 103.23 |
| commons-codec | 2.86 | 329.49 | 460.89 |
| commons-io | 3.06 | 140.78 | 250.94 |
| config-magic | 75.37 | 42.38 | 185.80 |
| csv | 98.43 | 115.39 | 271.59 |
| dbutils | 46.28 | 53.06 | 75.68 |
| events | 125.51 | 35.33 | 83.69 |
| faunus | 3.54 | 340.23 | 616.76 |
| java-api-wrapper | 9.73 | 121.26 | 235.57 |
| java-classmate | 118.77 | 80.57 | 288.54 |
| jopt-simple | 83.76 | 80.80 | 301.70 |
| mgwt | 79.93 | 45.56 | 126.94 |
| mirror | 103.04 | 24.29 | 169.21 |
| mp3agic | 162.77 | 465.32 | 677.62 |
| ognl | 176.89 | 43.10 | 2665.86 |
| pipes | 56.44 | 87.35 | 225.87 |
| primitives | 0.27 | 137.59 | 69.53 |
| validator | 98.11 | 178.66 | 450.97 |
| webbit | 31.03 | 239.10 | 497.23 |
| $\mu$ | 91.07 | 134.59 | 371.77 |
| $\sigma$ | 75.57 | 110.12 | 507.05 |

TABLE XXV: Multi information correlation of different mutation tools

| | $R^2$ | $\tau_b$ | Difference $\mu$ | $\sigma$ |
|---|---|---|---|---|
| $Judy \times Pit$ | 0.29 | 0.19 | -280.70 | 490.79 |
| $Judy \times Major$ | -0.06 | -0.14 | -43.52 | 137.45 |
| $Pit \times Major$ | 0.10 | 0.43 | 237.19 | 507.78 |

## G. Entropy carried by mutants

The measures we looked at previously evaluated how redundant a set of mutants is. Another way to think about a set of mutants is to think of mutants as expressing all possible failures of a system. In this sense, the utility of a set of mutants and test cases is the ability of such a system to express the possible behaviors of the system (each failure of a test in a mutant expresses a behavior of the original program). This suggests that a measure of information contained in the mutant×test-case matrix can be a good comparative measure of how good a set of mutants is.

In information theory *Shannon entropy* [92] is a measure of the information content in the given data. Entropy is related to *multi information*. That is, *multi information* is the difference between the sum of independent entropies of random variables

TABLE XXVI: Model fit for multi information

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 2457752.40 | 102406.35 | 0.98 | 0.5078 |
| phase | 1 | 156285.44 | 156285.44 | 1.49 | 0.2274 |
| Residuals | 49 | 5125806.30 | 104608.29 | | |

Model fit with phase $R^2$ =0

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 2457752.40 | 102406.35 | 1.20 | 0.2840 |
| audience | 1 | 1117537.88 | 1117537.88 | 13.15 | 0.0007 |
| Residuals | 49 | 4164553.86 | 84990.90 | | |

Model fit with audience $R^2$ =0.187

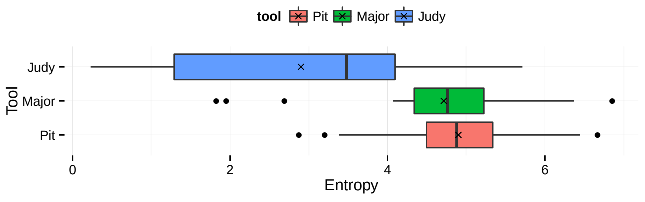
| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 2457752.40 | 102406.35 | 1.19 | 0.2997 |
| tool | 2 | 1141207.98 | 570603.99 | 6.61 | 0.0029 |
| Residuals | 48 | 4140883.76 | 86268.41 | | |

Model fit with tool $R^2$ =0.175

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 2320.14 | 96.67 | 540.43 | 0.0000 |
| Residuals | 7175 | 1283.46 | 0.18 | | |

Base model fit $R^2$ =0.643

Full set of mutants



100 sample

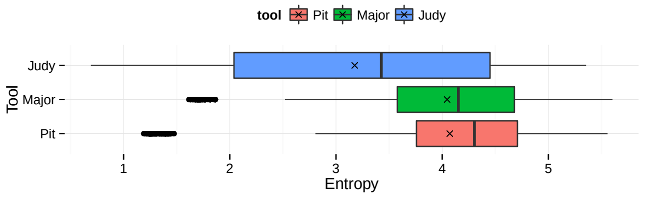


Fig. 10: Entropy

TABLE XXVII: Multi information correlation of different mutation tools (100 sample)

| | $R^2$ | $\tau_b$ | Difference $\mu$ | $\sigma$ |
|---|---|---|---|---|
| $Judy \times Pit$ | 0.86 | 0.53 | 0.14 | 6.48 |
| $Judy \times Major$ | 0.85 | 0.59 | -0.42 | 6.73 |
| $Pit \times Major$ | 0.78 | 0.58 | -0.56 | 8.13 |

TABLE XXVIII: Model fit for multi information (100 sample)

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 980113.02 | 40838.04 | 2575.91 | 0.0000 |
| phase | 1 | 672.77 | 672.77 | 42.44 | 0.0000 |
| Residuals | 7174 | 113735.56 | 15.85 | | |

Model fit with phase $R^2$ =0.896

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 980113.02 | 40838.04 | 2570.60 | 0.0000 |
| audience | 1 | 437.78 | 437.78 | 27.56 | 0.0000 |
| Residuals | 7174 | 113970.54 | 15.89 | | |

Model fit with audience $R^2$ =0.896

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 980113.02 | 40838.04 | 2577.02 | 0.0000 |
| tool | 2 | 737.74 | 368.87 | 23.28 | 0.0000 |
| Residuals | 7173 | 113670.58 | 15.85 | | |

Model fit with tool $R^2$ =0.896

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 980113.02 | 40838.04 | 2561.12 | 0.0000 |
| Residuals | 7175 | 114408.33 | 15.95 | | |

Base model fit $R^2$ =0.895

and their joint entropy. Formally,

$$C(X_1..X_n) = \sum_{i=1}^{N} H(X_i) - H(X_1..X_n)$$

Another reason we are interested in the entropy of a set of mutants is that the properties of entropy are also relevant to how good we judge a set of mutants to be. That is, as we expect from a measure of quality of a set of mutants, the value can never be negative (adding a mutant to a set of mutants should not decrease the utility of a mutant set). Secondly, a mutant set where all mutants are killed by all test cases has minimal value (think of a minimal set of mutants for such a matrix). This is mirrored by the entropy property that $I(1) = 0$. Similarly, a mutant set where no mutants are killed by any test cases is also of no value (again consider the minimal set of mutants for such a matrix), which is also mirrored by entropy $I(0) = 0$. Finally, we expect that if two mutant sets representing independent failures are combined, the measure should reflect the sum of their utilities. With entropy, the joint information of two independent random variables is their sum of respective informations. Finally, the maximum entropy for a set of mutants happens when none of the mutants in the set are subsumed by any other mutants in the set. The entropy of a random variable is given by $I(p) = -plog_2(p)$.

The entropy of the set of mutants produced by each tool is given in Table XXIX, and the comparison between different tools in terms of entropy is given in Table XXX. Figure 10(a) shows the entropy carried by the mutants.

**Impact of entropy**: Mutation sets with larger entropy carry more information. Hence tools that produce mutant sets with larger entropy are better.

TABLE XXIX: Entropy of different mutation tools

| Project | Judy | Major | Pit |
|---|---|---|---|
| annotation-cli | 2.81 | 2.69 | 3.38 |
| asterisk-java | 4.37 | 5.13 | 5.27 |
| beanutils | 5.71 | 6.37 | 6.44 |
| beanutils2 | 3.97 | 4.13 | 4.80 |
| clazz | 1.06 | 4.62 | 3.20 |
| cli | 5.27 | 5.22 | 5.28 |
| collections | 1.51 | 6.34 | 4.41 |
| commons-codec | 0.23 | 5.80 | 6.27 |
| commons-io | 1.52 | 6.35 | 6.67 |
| config-magic | 4.19 | 4.35 | 4.41 |
| csv | 3.34 | 4.72 | 5.06 |
| dbutils | 4.10 | 4.07 | 4.73 |
| events | 3.48 | 1.95 | 3.94 |
| faunus | 0.35 | 4.99 | 4.91 |
| java-api-wrapper | 0.72 | 4.59 | 4.82 |
| java-classmate | 4.01 | 4.30 | 5.35 |
| jopt-simple | 4.41 | 4.79 | 5.34 |
| mgwt | 3.79 | 4.34 | 4.50 |
| mirror | 4.22 | 4.76 | 5.63 |
| mp3agic | 3.93 | 5.10 | 5.15 |
| ognl | 0.89 | 1.82 | 4.70 |
| pipes | 3.53 | 4.77 | 4.88 |
| primitives | 0.47 | 6.85 | 2.87 |
| validator | 3.36 | 5.40 | 5.62 |
| webbit | 1.29 | 4.42 | 4.88 |
| $\mu$ | 2.90 | 4.71 | 4.90 |
| $\sigma$ | 1.66 | 1.23 | 0.92 |

TABLE XXX: Entropy correlation of different mutation tools

| | $R^2$ | $\tau_b$ | Difference $\mu$ | $\sigma$ |
|---|---|---|---|---|
| $Judy \times Pit$ | 0.28 | 0.25 | -2.00 | 1.66 |
| $Judy \times Major$ | -0.07 | 0.01 | -1.81 | 2.13 |
| $Pit \times Major$ | 0.35 | 0.41 | 0.19 | 1.25 |

We compare the number of mutants produced by each tool (Figure 2) and entropy from mutants by each (Figure 10(a)). As expected, a larger number of mutants can carry more information.

**Q:** *Does the phase of generation or target audience have an impact?*

We rely on *analysis of variance* (*ANOVA*) (Equation 1 — measure is *entropy*).

The *ANOVA* (Table XXXI) suggests that both *phase* and *audience* is is statistically significant, while *project* is not. We note that the full models explain 0.661% (phase), 7.956% (audience), and 40.513% (tool) of the variance ($R^2$). Further the difference explained by *phase* is 6.895%, by *audience* is 14.19%, and by *tool* is 46.747%. (difference in $R^2$ from a model that contained only project).

TABLE XXXI: Model fit for entropy

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 51.89 | 2.16 | 0.88 | 0.6293 |
| phase | 1 | 11.03 | 11.03 | 4.47 | 0.0396 |
| Residuals | 49 | 120.95 | 2.47 | | |

Model fit with phase $R^2$ =0.007

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 51.89 | 2.16 | 0.95 | 0.5468 |
| audience | 1 | 19.92 | 19.92 | 8.71 | 0.0048 |
| Residuals | 49 | 112.07 | 2.29 | | |

Model fit with audience $R^2$ =0.08

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 51.89 | 2.16 | 1.46 | 0.1298 |
| tool | 2 | 61.04 | 30.52 | 20.65 | 0.0000 |
| Residuals | 48 | 70.95 | 1.48 | | |

Model fit with tool $R^2$ =0.405

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 51.89 | 2.16 | 0.82 | 0.6971 |
| Residuals | 50 | 131.99 | 2.64 | | |

Base model fit $R^2$ =-0.062

TABLE XXXII: Entropy correlation of different mutation tools (100 sample)

| | $R^2$ | $\tau_b$ | Difference $\mu$ | $\sigma$ |
|---|---|---|---|---|
| $Judy \times Pit$ | 0.53 | 0.40 | -1.03 | 1.17 |
| $Judy \times Major$ | 0.54 | 0.35 | -0.92 | 1.18 |
| $Pit \times Major$ | 0.69 | 0.50 | 0.11 | 0.74 |

*1) What is the impact of tools after controlling the number of mutants produced?:* To answer this question, we sample 100 mutants at a time from each project 100 times. Figure 10(b) shows the entropy carried by the mutants, after controlling the number of mutants.

Table XXXII provides the correlation of entropy of mutants produced by different tools for different projects. As in other statistical measures, we note an across the board improvement in both correlation measures — $R^2$ (0.688 to 0.533) and $\tau_b$ (0.501 to 0.354).

**Q:** *Does the phase of generation or target audience have an impact?*

We rely on *analysis of variance* (*ANOVA*) (Equation 1 — measure is *entropy* after controlling the number of mutants).

The *ANOVA* (Table XXXIII) suggests that both *phase* and *audience* is is statistically significant. We note that the full models explain 61.995% (phase), 62.543% (audience), and 73.047% (tool) of the variance ($R^2$). Further the difference explained by *phase* is 2.889%, by *audience* is 3.436%, and by *tool* is 13.941%. (difference in $R^2$ from a model that contained only project).

TABLE XXXIII: Model fit for entropy (100 sample)

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 5963.01 | 248.46 | 467.59 | 0.0000 |
| phase | 1 | 290.32 | 290.32 | 546.37 | 0.0000 |
| Residuals | 7174 | 3812.02 | 0.53 | | |

Model fit with phase $R^2$ =0.62

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 5963.01 | 248.46 | 474.42 | 0.0000 |
| audience | 1 | 345.22 | 345.22 | 659.18 | 0.0000 |
| Residuals | 7174 | 3757.12 | 0.52 | | |

Model fit with audience $R^2$ =0.625

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 5963.01 | 248.46 | 659.32 | 0.0000 |
| tool | 2 | 1399.24 | 699.62 | 1856.53 | 0.0000 |
| Residuals | 7173 | 2703.10 | 0.38 | | |

Model fit with tool $R^2$ =0.73

| | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|---|---|---|---|---|---|
| project | 24 | 5963.01 | 248.46 | 434.55 | 0.0000 |
| Residuals | 7175 | 4102.34 | 0.57 | | |

Base model fit $R^2$ =0.591

## V. Discussion

We evaluated an ensemble of measures including traditional mutation scores, strength of mutants using minimal and surface mutants, statistical measures of diversity of mutants, and information content in the mutant kills to find whether any tool produced mutants that were consistently better than the others by a meaningful margin.

### A. *Comparison of difficulty of detection with mutation scores*

We include two measures in the traditional comparison; the raw mutation score (Section IV-A), and the refined mutation score (Section IV-B). Considering the raw mutation scores that were reported by different tools, we find that while Pit does produce a much larger number of mutants than other tools (Figure 2), and that the mean mutation scores across projects produced by different tools are quite close (Figure 4). Surprisingly, the correlation between the mutation scores produced ranges from low ($Judy \times Pit$ 0.375) to medium ($Pit \times Major$ 0.675). Similarly Kendall's $\tau_b$ is also low ($Judy \times Pit$ 0.273) to medium ($Pit \times Major$ 0.54). We also see a large standard deviation (up to 26.927 for $Judy \times Pit$).

Our measures of correlation are markedly different from those obtained from mutation scores reported by Delahaye et al. (Table II) where the three tools we compare were found to have high correlation. However, we note that we have a much larger variety of subject programs, and that the data from Delahaye et al. is incomplete (only 4 complete observations with all the three tools under consideration), which may explain the discrepancy. We also note that the tools we have not considered — Jumble, and Javalanche — have even worse correlation with other tools in Delahaye's data. (However Madeyski [41] reports that Judy and Jumble had a high correlation of 0.89). Taken together, this corroborates our finding that the relationship between tools varies based on the project being tested.

Our data from raw mutation scores suggests that tools rarely agree with each other on mutation score, and often differ by a large amount.

Mutation score from any single tool can be severely misleading. The scores from different tools sometimes differ by a large amount.

However, we find no tool consistently under or over reporting the mutation scores with respect to what is reported by other tools.

This suggests two things: (1) mutation score from a single tool can be severely misleading; (2) there is no single tool that can be said to produce consistently hard-to-detect mutants or easy-to-detect mutants.

For raw mutation scores, we do not consider equivalent mutants. For our second measure (Section IV-B), we removed all undetected mutants, and for the remaining mutants, evaluated mutation scores produced by partial test suites containing a fixed fraction of the original test suite. Our results from refined mutation scores on partial test suites with undetected mutants removed corroborate our findings with raw mutation scores. As Table VII shows, the difference between Judy and Pit reduced for both $R^2$, and $\tau_b$, while measurements of other pairings remain very close. The spread, while slightly smaller than raw mutation scores, still remain high.

Note that mutation score is often the only measure from a set of mutants that is obtained by testers and researchers who are interested in the quality of a test suite. Hence, irrespective of the other measures of quality, the low agreement between tools for such a standard measure is disheartening. Interestingly, Pit and Major are polar opposites in both dimensions we evaluate — phase of generation, and target audience. However we find that they show a consistently higher correlation with each other, especially when compared with Judy, which suggests that at least as far as mutation score is concerned, the impact of phase of generation and target audience is minimal.

### B. *Comparison of mutant strength with minimal mutation sets*

The strength of a set of mutants irrespective of the size of the set provides a good measure of the quality of a set of mutants. After all, when one uses mutation analysis to choose test cases, the unique mutants identified are the only factor that contributes to the quality of the test suite thus chosen. Mutants produced by various mutation operators often have a high redundancy, with multiple mutants causing the same failure. There have been various efforts to reduce the amount of redundancy in mutants generated. Ammann et al. [46] suggests that a theoretical limit on such reductions can be obtained by using a set of minimal mutants such that a test suite that is able to kill that set of mutants is guaranteed to kill the full set of mutants. Thus we can consider the minimal set of mutants to be the actual number of true mutants generated (avoiding redundant mutants), and hence a reasonable measure of the quality of a set of mutants. We analysed (Section IV-C) the minimal mutants from different tools across the subject programs using both the entire set of mutants, and also restricting the number of mutants to just 100.

Note that there is a very clear interpretation for the size of minimum mutant set — the larger the better.

Our results from Section IV-C suggest that the minimal set of mutants produced by different tools varies widely, from low ($Judy \times Major$ 0.259) to high ($Pit \times Major$ 0.961). Kendall's $\tau_b$ ranges from lowest ($Judy \times Major$ 0.346) to highest ($Pit \times Major$ 0.846). We also see the maximum standard deviation for $Judy \times Major$ (208.588 ). Further, the mean difference between minimum mutants produced by tools ranges from $Judy \times Pit$ (-124.44) to $Pit \times Major$ (17.4).

Interestingly, the situation is different when the number of mutants is controlled. We see a correlation between the minimum set from different tools from medium ($Judy \times Pit$ 0.486) to high ($Pit \times Major$ 0.926). Similarly Kendall's $\tau_b$ is also medium ($Judy \times Pit$ 0.351) to high ($Pit \times Major$ 0.73).

It is instructive to think what a high correlation means. It means that the difference between minimum mutant sets produced by different tools is consistent. That is, for the tools that exhibit a high correlation, we can say that a constant number of mutants picked from one of the tools will consistently be qualitatively better. The mean difference between minimum mutants produced by tools ranges from $Judy \times Pit$ (-17.865) to $Pit \times Major$ (2.816). That is, in our comparison, the tools that exhibit high correlation — Pit and Major — differ only by a mean small amount compared to either the total number of mutants, or the mean for $Judy \times Pit$.

We find a similar result from our analysis of surface mutants (Section IV-D). Surface mutants are similar to minimal set of mutants, but with a small relaxation in their construction — we do not require that the test suite be minimized. Rather, we remove all mutants that are dynamically subsumed by another. This results in a slightly larger, but more accurate estimation of redundancy. The measures vary from low ($Judy \times Major$ 0.25) to high ($Pit \times Major$ 0.944). Kendall's $\tau_b$ is lowest ($Judy \times Pit$ 0.245) to highest ($Pit \times Major$ 0.758). We also see maximum standard deviation for $Judy \times Major$ (233.169). Further, the mean difference between minimum mutants produced by tools range from $Judy \times Pit$ (-172.76) to $Pit \times Major$ (51.6).

We find a similar result as that of minimal mutants when the number of mutants is controlled. We see a correlation between the minimum set from different tools from medium ($Judy \times Pit$ 0.47) to high ($Pit \times Major$ 0.917). Similarly Kendall's $\tau_b$ is also medium ($Judy \times Pit$ 0.344) to high ($Pit \times Major$ 0.723). Remember that a comparison is reasonable only for those tools that exhibit a high correlation (otherwise the results are too dependent on individual projects). As before, the mean difference between minimum mutants produced by tools range from $Judy \times Pit$ (-17.284) to $Pit \times Major$ (2.624). That is, like in minimum mutants, the tools that exhibit high correlation — Pit and Major differ only by a mean small number of mutants on average.

Using strength of mutants measured by minimal and surface mutant sets, Pit and Major produce high strength mutants, and differ only by a small amount, with Pit slightly better.

### C. *Comparison of mutant diversity with statistical measures*

We had shown before [38] that sum of covariance of a set of mutants reduces the fraction of mutants that can represent the mutation score accurately. A smaller sum of covariance is strong evidence that one set of mutants is more varied than a set with a larger sum of covariance if both have a similar number of mutants.

Our evaluation (Section IV-E) shows that the correlation between tools for covariance is small ($Pit \times Major$ 0.187) to ($Judy \times Pit$ 0.452). The Kendall $\tau_b$ correlations are ($Judy \times Pit$ 0.427) to ($Pit \times Major$ 0.427). However, this is as expected. Remember that a larger covariance essentially means a lower fraction of mutants can represent the full mutant set. Here, the number of mutants are different, and hence not really comparable. The interesting part is the mean difference. That is, if the mutants are comparable we would expect a larger difference in covariance between Pit and other tools since it generates a larger set of mutants. Similarly, Major and Judy should differ little. This is confirmed by our results with $Pit \times Major$ 6.867, $Judy \times Pit$ -6.058 and $Judy \times Major$ 0.809.

Controlling the number of mutants (either by sampling or by using average covariance – dividing by number of mutants) should on the other hand lead to a higher correlation and lower difference. Our results show that this is as expected ($Pit \times Major$ -0.053) to ($Judy \times Pit$ 0.341).

Our *multi information* observations (Section IV-F) paint a similar picture. A low correlation between tools, but larger difference in mutual information between mutants produced by tools generating a larger number of mutants and those producing a smaller number. ($Judy \times Pit$ -280.702 $Pit \times Major$ 237.187 and $Judy \times Major$ -43.516).

As before, if the number of mutants is controlled, we have high correlation ($Judy \times Pit$ 0.865) to ($Pit \times Major$ 0.785). and very small mean difference ($Judy \times Pit$ 0.143) and ($Pit \times Major$ -0.561).

These measures again show that there is very little difference between mutants generated by different tools, although Pit comes out slightly better once the number of mutants is controlled.

Our statistical measures of diversity of mutants show that once the number of mutants is controlled, there is little difference in the diversity of mutants produced by different tools.

### D. *Comparison of information carried with entropy*

Similar to diversity measures, we expect little correlation when number of mutants is not controlled for, ($Judy \times Major$ -0.066) to ($Pit \times Major$ 0.354). We expect a larger set of mutants to have more entropy, and smaller set of mutants to have less entropy. ($Judy \times Pit$ -2) and ($Pit \times Major$ 0.186).

Once the number of mutants are controlled, we see a larger correlation ($Judy \times Major$ 0.537) to ($Pit \times Major$ 0.688), but little difference in mean ($Judy \times Pit$ -2) and ($Pit \times Major$ 0.186).

For entropy, we again find Pit and Major better than Judy. Pit and Major have similar values, with Pit leading by a slight margin.

In terms of entropy, the leaders are Pit and Major, with Pit leading by a small margin.

## E. *Tool Assessment*

In this section, we provide an overall evaluation of the tools we used, considering all measures.

*1) Judy:* Judy is a tool oriented towards a research audience, and produces bytecode based mutants. We see that Judy produces the smallest number of mutants, compared to Major and Pit. In terms of raw mutation score, Judy has a slight advantage over other tools, with Judy producing the lowest mean mutation score. However, the difference is small, further reduced if non-detected mutants are removed first. In terms of mutant strength with either minimal mutants or surface mutants, the other two tools (Major and Pit) perform better than Judy. In terms of covariance between mutants, while Judy is better than Pit and Major on average when number of mutants is not considered, both Pit and Major are better than Judy when we restrict the number of mutants. In terms of multi information, Judy is better than Pit and Major when full sets of mutants are considered. However this is likely to be due to the small number of mutants produced by Judy, as shown by the sampled measure. That is, for a constant sized sample of mutants, Pit and Major produced more diverse mutants than Judy. The entropy measure also suggests that mutants from Pit and Major contain more information about the program than Judy.

*2) Major:* Major is one of the few tools in use that is source based. It is also oriented towards the research community. In terms of raw mutation score, Major produces a medium mean mutation score score compared to Pit (higher) and Judy (lower). However, the mean difference is marginal. The conclusions are similar for refined mutation scores, with a difference of a few percentage points between mean mutation score across projects with other tools. In terms of minimal and surface mutant sets, without controlling the number of mutants, Major produces the best mean mutant set. However, this advantage disappears once we control the number of mutants, with Pit producing better mean mutant set. In terms of diversity measures, sum of covariance and mutual information, Major occupies a middle rank between Pit and Judy both with and without control for number of mutants, with Judy better when the number of mutants are not controlled, and Pit better when the number of mutants is controlled. For entropy, Major is better than Judy, while Pit is better than Major. We also note that Mutants from Major and Pit are very close in most measures.

*3) Pit:* Pit is a tool firmly focused on an industrial audience. It is bytecode based, and in terms of ease of use, provides the best user experience. Pit produces a very large number of mutants compared to Major and Judy. In terms of mean raw mutation score, the mutants produced by Pit are marginally easier to detect than those produced by other tools (the difference decreases if refined mutants are used). In terms of size of minimal and surface mutant sets, Pit occupies a middle ground between Judy (smaller) and Major (larger). However, when the number of mutants is controlled, Pit produces the strongest mutant set. For diversity measures such as sum of covariance and mutual information, controlling the number of mutants, Pit produces mutants with the most diversity. In terms of information content, Pit produces mutants with the largest entropy both when number of mutants is controlled or otherwise.

## F. *Impact of* phase *of generation*

There is no evidence that phase of generation had any impact on the mutation score — either raw or refined. For strength of mutants — using minimal or surface sets — there was no evidence that phase mattered when the number of mutants was not controlled. While the variable phase could explain some of the variability in mutation score with statistical significance once the number of mutants was controlled, the actual effect was only a few percentage points, and was dwarfed by the variability introduced by the tool. For measurements of diversity of mutants — sum of covariance and mutual information — we found similar results. While statistically significant effect was observed once the number of mutants was controlled, the effect was less than a percentage point, and was dwarfed by the difference due to tool. For entropy, the effect of phase was statistically significant both with and without control for number of mutants. However, as for the measures of diversity, the variability explained was small, and dwarfed by the variability due to tool.

In summary, there is little evidence of a large impact of phase of generation on the variability of mutants.

## G. *Impact of target* audience

There is no evidence that target audience had any impact on the mutation score — either raw or refined. For strength of mutants — using minimal or surface sets — the variable audience is statistically significant. For both minimal and surface sets, the variance explained by audience is less than that explained by tool for both full set of mutants, and constant number of sampled mutants. Considering the measurements for diversity of mutants — sum of covariance and mutual information — we found that audience is indeed statistically significant, and the impact is larger than that of considering tool separately when considering the full set of mutants. However, when considering a constant sample of mutants, the impact of tool is larger for sum of covariance. For mutual information, the variation due to project dwarfs the variation due to tool or audience. For entropy, the impact of tool is again much larger than that due to audience.

In summary, there is some evidence of a practical impact of target audience on the variability of mutants using some of the measures. However, the variability due to tool is larger than the variability due to target audience, except for diversity measures, and for diversity measures, the effect disappears when the number of mutants is controlled.

The target audience has an impact on the variability of mutants. However, this may be an artifact of the particular tools used, and the number of mutants produced.

### *H. Impact of project characteristics*

In every measure tested, even after accounting for obvious factors such as number of mutants, and quality of test suites, the variation due to individual characteristics of the project was the single highest factor contributing to the variation of measurements for different tools. Note that since we control for number of mutants and test suite quality, this means that some underlying semantic property of each project is the driving factor, not mere size or test effort.

The characteristics of individual projects were the most important factor determining the effectiveness of different tools by a large margin.

That is, the interaction of *syntactic* and *semantic* characteristics of the project seems to determine whether a particular mutation tool will perform well with a given project or not. This is an area where further investigation is required to understand what these factors are both in generation and detection of mutants, and especially how they affect the quality of mutants produced. A more immediate implication is that, until we have an understanding of the factors involved, researchers should be wary of relying on a small number of projects for evaluation of their techniques. Finally, evolving a consensus on the standardization of mutants produced is important for the validity of mutation analysis in further research.

## VI. Threats to Validity

Our research makes use of multiple mutation tools, a variety of measures, and a large number of subjects. This means that our research is subject to the following threats to validity.

The single most important threat to validity is the applicability of our findings. Our subject programs were open source Java projects from Github. While our choice of subjects was driven by concerns about the size of the project (the larger the better), the size of the test suite (the larger the better), and the project's ability to complete mutation analysis successfully for the tools we selected, none of which have any direct influence on the measures, threats due to indirect unforeseen influences can't be ruled out. Further, our research results are only directly applicable only to Java programs. Though we expect our findings to be applicable to mutation tools in other programming languages, there is no statistical guarantee for such a belief other than the relative similarity between languages, between possible bugs, and hence between mutants.

While we tried very hard to use different kinds of tools, the fact remains that only three tools could be evaluated. This is not statistically adequate for any sort of guarantee about the behavior of these tools. We base our confidence on the observation that these tools are actively developed, used by real world practitioners of testing, and researchers, and also that the mutation operators are reasonably complete. However, it is possible that our tools may not be representative of the categories such as source based or bytecode mutation engines, or a typical representative of a tool aimed at research or industry. It is not clear if source and bytecode is a reasonable representation of the variation in mutation due to difference in phase of generation. More importantly, since we have only three tools, large deviance from any single tool is a threat to the validity of our research.

Finally, software bugs are a fact of life, and it can't be ruled out either in the tools used or in the analysis we performed.

While we have taken every care, the possibility of these threats remain. Hence it is important that our research be replicated on other languages with different tools, and on tools using different phases for mutant generation. To facilitate such a research, we place the data from our research and also the source code of our publication which can be regenerated from new data in the given format in public domain [93].

## VII. Conclusion

We evaluated mutants produced by different tools for Java mutation across a large number of projects using diverse measures of tool effectiveness. Using these measures, we find that the tool targeting industry — Pit — produced the best mutants, although the difference with other tools was often very small. We also find that the influence of project, even after controlling for factors such as test suite and number of mutants (which usually follows source code size of the project), is still the dominant contributor to the variation between the measurements from different tools.

That is, the syntactic characteristics of the projects and their interaction with the semantic characteristics of the particular project have the largest influence on how well a particular mutation analysis tool performs in terms of the quality of mutants produced.

We find that in terms of mutation score, there is very little mean difference between different tools. However, we have a more worrying result. Even though there was negligible mean difference, the standard deviation and different forms of correlation indicated that the mutant sets seldom agree on the mutation scores, and often even disagreed on how to rank two test suites in terms of effectiveness. This is especially worrying given that a number of research papers rely on small differences in correlation between mutation scores and other measures to show that a particular technique works well. This means that the research conducted so far is strongly tool dependent. Further, the relatively large spread of mutation scores suggests that some mutation tools may judge a test set to be effective by some benchmark, and others may not, which makes using any target mutation score (e.g., 80%) problematic as a guideline for testing practice. It is unsafe to rely on any single tool to measure adequacy of a test suite.

Our findings indicate a need to standardize mutation scores. We propose that we go back to the original definition. That is, standardize the mutation scores based on the actual *exhaustive* generation of all mutants as permitted by the grammar of the language in question. In fact, as Just et al. [7] shows, traditional "easy" operators are not sufficient, and we have to be serious about including *all possible* first order mutants including function call mistakes, argument swapping, casts etc. — all that are indicated by the language grammar. Since even first-order changes can be infeasibly large, we suggest that the changes to primitive types such as integers be restricted to the well known traditional boundary values such as *0, 1, -1*, and *(-)maxint*.

Since for Java, there is no tool that currently provides complete first-order mutation as we describe above, we suggest that practitioners use a much larger set of projects to evaluate mutation results using any of the tools we evaluated. As we show, a large number of projects tend to ameliorate the ill effects of an incomplete set of mutants. Once a standard mutation framework is available, for any new mutation tool or reduction technique that targets test quality *assessment* we require that the mutation score from the proposed technique be in high $R^2$ correlation, of at least 0.95 with the standard, and the coefficients $\beta_0, \beta_1$ of linear regression $\mu_{standard} = \beta_0 + \beta_1 \mu_{new}$ be available. On the other hand, for tools and reduction techniques that target *comparison* of testing testing techniques, we require that the new mutation scores be in high Kendall's $\tau_b$ correlation of at least 0.95 with the standard.

There is a reason for insisting on different correlation measures. For test assessment, it is only required that the standard mutation scores can be predicted from the new mutation score with the given accuracy. That is, it does not matter if the difference is not consistently positive or negative. However, for comparison between different testing techniques, it is important that if the new technique finds a tool to be better than another, it is in agreement with the standard mutation analysis, also. Using Kendall's $\tau_b$ also lets other tools be more discriminating in specific areas than the standard, but still be in overall agreement.

Obviously, in the long run there may be new standards (e.g., more complete sets of mutation operators) that replace the current standard; Such a tool needs to offer an argument for its superiority, however, and measure its statistical divergence from the standard to place results using an older standard in context.

## References

[1] R. J. Lipton, "Fault diagnosis of computer programs," Carnegie Mellon Univ., Tech. Rep., 1971.

[2] T. A. Budd, R. J. Lipton, R. A. DeMillo, and F. G. Sayward, *Mutation analysis*. Yale University, Department of Computer Science, 1979.

[3] M. Daran and P. Thévenod-Fosse, "Software error analysis: A real case study involving real faults and mutations," in *ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, 1996, pp. 158–171.

[4] J. H. Andrews, L. C. Briand, and Y. Labiche, "Is mutation an appropriate tool for testing experiments?" in *International Conference on Software Engineering*. IEEE, 2005, pp. 402–411.

[5] J. H. Andrews, L. C. Briand, Y. Labiche, and A. S. Namin, "Using mutation analysis for assessing and comparing testing coverage criteria," *IEEE Transactions on Software Engineering*, vol. 32, no. 8, pp. 608–624, 2006.

[6] H. Do and G. Rothermel, "On the use of mutation faults in empirical assessments of test case prioritization techniques," *Software Engineering, IEEE Transactions on*, vol. 32, no. 9, pp. 733–752, 2006.

[7] R. Just, D. Jalali, L. Inozemtseva, M. D. Ernst, R. Holmes, and G. Fraser, "Are mutants a valid substitute for real faults in software testing?" in *ACM SIGSOFT Symposium on The Foundations of Software Engineering*. Hong Kong, China: ACM, 2014, pp. 654–665.

[8] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Transactions on Software Engineering*, vol. 37, no. 5, pp. 649–678, 2011.

[9] M. Delahaye and L. Du Bousquet, "A comparison of mutation analysis tools for java," in *Quality Software (QSIC), 2013 13th International Conference on*. IEEE, 2013, pp. 187–195.

[10] T. A. Budd and A. S. Gopal, "Program testing by specification mutation," *Computer Languages*, vol. 10, no. 1, pp. 63–73, Jan. 1985.

[11] V. Okun, "Specification mutation for test generation and analysis," Ph.D. dissertation, University of Maryland Baltimore County, 2004.

[12] B. H. Smith and L. Williams, "An empirical evaluation of the mujava mutation operators," in *Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION, 2007. TAICPART-MUTATION 2007*. IEEE, 2007, pp. 193–202.

[13] Y. Jia and M. Harman, "Milu: A customizable, runtime-optimized higher order mutation testing tool for the full c language," in *Practice and Research Techniques, 2008. TAIC PART'08. Testing: Academic & Industrial Conference*. IEEE, 2008, pp. 94–98.

[14] A. Derezińska and K. Hałas, "Analysis of mutation operators for the python language," in *International Conference on Dependability and Complex Systems*, ser. Advances in Intelligent Systems and Computing. Springer International Publishing, 2014, vol. 286, pp. 155–164.

[15] D. Le, M. A. Alipour, R. Gopinath, and A. Groce, "Mucheck: An extensible tool for mutation testing of haskell programs," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*. ACM, 2014, pp. 429–432.

[16] R. Just, "The major mutation framework: Efficient and scalable mutation analysis for java," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ser. ISSTA 2014. New York, NY, USA: ACM, 2014, pp. 433–436.

[17] M. Kusano and C. Wang, "Ccmutator: A mutation generator for concurrency constructs in multithreaded c/c++ applications," in *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*. IEEE, 2013, pp. 722–725.

[18] H. Coles, "Pit mutation testing," http://pitest.org/.

[19] S. A. Irvine, T. Pavlinic, L. Trigg, J. G. Cleary, S. Inglis, and M. Utting, "Jumble java byte code to measure the effectiveness of unit tests," in *Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION, 2007. TAICPART-MUTATION 2007*. IEEE, 2007, pp. 169–175.

[20] J. Duraes and H. Madeira, "Emulation of software faults by educated mutations at machine-code level," in *International Symposium on Software Reliability Engineering*, 2002, pp. 329–340.

[21] H. Coles, "Pit mutators," http://pitest.org/quickstart/mutators/.

[22] M. Gligoric, V. Jagannath, and D. Marinov, "Mutmut: Efficient exploration for mutation testing of multithreaded code," in *Software Testing, Verification and Validation (ICST), 2010 Third International Conference on*. IEEE, 2010, pp. 55–64.

[23] J. Nanavati, F. Wu, M. Harman, Y. Jia, and J. Krinke, "Mutation testing of memory-related operators," in *Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on*. IEEE, 2015, pp. 1–10.

[24] Y.-S. Ma, Y.-R. Kwon, and J. Offutt, "Inter-class mutation operators for java," in *International Symposium on Software Reliability Engineering*. IEEE, 2002, pp. 352–363.

[25] Y.-S. Ma, J. Offutt, and Y. R. Kwon, "Mujava: an automated class mutation system," *Software Testing, Verification and Reliability*, vol. 15, no. 2, pp. 97–133, 2005.

[26] C. Zhou and P. Frankl, "Mutation testing for java database applications," in *Software Testing Verification and Validation, 2009. ICST'09. International Conference on*. IEEE, 2009, pp. 396–405.

[27] A. Siami Namin, J. H. Andrews, and D. J. Murdoch, "Sufficient mutation operators for measuring test effectiveness," in *International Conference on Software Engineering*. ACM, 2008, pp. 351–360.

[28] A. J. Offutt, A. Lee, G. Rothermel, R. H. Untch, and C. Zapf, "An experimental determination of sufficient mutant operators," *ACM Transactions on Software Engineering and Methodology*, vol. 5, no. 2, pp. 99–118, 1996.

[29] E. F. Barbosa, J. C. Maldonado, and A. M. R. Vincenzi, "Toward the determination of sufficient mutant operators for c," *Software Testing, Verification and Reliability*, vol. 11, no. 2, pp. 113–136, 2001.

[30] R. Just, G. M. Kapfhammer, and F. Schweiggert, "Do redundant mutants affect the effectiveness and efficiency of mutation analysis?" in *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*. IEEE, 2012, pp. 720–725.

[31] B. Kurtz, P. Ammann, M. E. Delamaro, J. Offutt, and L. Deng, "Mutant subsumption graphs," in *Software Testing, Verification and Validation Workshops (ICSTW), 2014 IEEE Seventh International Conference on*. IEEE, 2014, pp. 176–185.

[32] D. Schuler, V. Dallmeier, and A. Zeller, "Efficient mutation testing by checking invariant violations," in *ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, 2009, pp. 69–80.

[33] A. J. Offutt and W. M. Craft, "Using compiler optimization techniques to detect equivalent mutants," *Software Testing, Verification and Reliability*, vol. 4, no. 3, pp. 131–154, 1994.

[34] D. Schuler and A. Zeller, "Covering and uncovering equivalent mutants," *Software Testing, Verification and Reliability*, vol. 23, no. 5, pp. 353–374, 2013.

[35] S. Nica and F. Wotawa, "Using constraints for equivalent mutant detection," in *Workshop on Formal Methods in the Development of Software, WS-FMDS*, 2012, pp. 1–8.

[36] M. Papadakis, Y. Jia, M. Harman, and Y. L. Traon, "Trivial compiler equivalence: A large scale empirical study of a simple, fast and effective equivalent mutant detection technique," in *International Conference on Software Engineering*, 2015.

[37] X. Yao, M. Harman, and Y. Jia, "A study of equivalent and stubborn mutation operators using human analysis of equivalence," *International Conference on Software Engineering*, pp. 919–930, 2014.

[38] R. Gopinath, A. Alipour, A. Iftekhar, C. Jensen, and A. Groce, "How hard does mutation analysis have to be, anyway?" in *International Symposium on Software Reliability Engineering*. IEEE, 2015.

[39] R. Gopinath, A. Alipour, I. Ahmed, C. Jensen, and A. Groce, "Do mutation reduction strategies matter?" Oregon State University, Tech. Rep., Aug 2015, under review for Software Quality Journal. [Online]. Available: http://hdl.handle.net/1957/56917

[40] H. Coles, "Mutation testing systems for java compared," http://pitest.org/java_mutation_testing_systems/.

[41] L. Madeyski and N. Radyk, "Judy–a mutation testing tool for java," *IET software*, vol. 4, no. 1, pp. 32–42, 2010.

[42] P. K. Singh, O. P. Sangwan, and A. Sharma, "A study and review on the development of mutation testing tools for java and aspect-j programs," *International Journal of Modern Education and Computer Science (IJMECS)*, vol. 6, no. 11, p. 1, 2014.

[43] J. Offut, "Problems with jester," https://cs.gmu.edu/~offutt/documents/personal/jester-anal.html.

[44] P. Ammann, "Problems with jester," https://sites.google.com/site/mutationworkshop2015/program/MutationKeynote.pdf.

[45] J. Offut, "Problems with parasoft insure++," https://cs.gmu.edu/~offutt/documents/handouts/parasoft-anal.html.

[46] P. Ammann, M. E. Delamaro, and J. Offutt, "Establishing theoretical minimal sets of mutants," in *International Conference on Software Testing, Verification and Validation*. Washington, DC, USA: IEEE Computer Society, 2014, pp. 21–30.

[47] J. W. Nimmer and M. D. Ernst, "Automatic generation of program specifications," *ACM SIGSOFT Software Engineering Notes*, vol. 27, no. 4, pp. 229–239, 2002.

[48] M. Harder, B. Morse, and M. D. Ernst, "Specification coverage as a measure of test suite quality," MIT Lab for Computer Science, Tech. Rep., 2001.

[49] M. Harder, J. Mellen, and M. D. Ernst, "Improving test suites via operational abstraction," in *International Conference on Software Engineering*. IEEE Computer Society, 2003, pp. 60–71.

[50] L. Zhang, S.-S. Hou, J.-J. Hu, T. Xie, and H. Mei, "Is operator-based mutant selection superior to random mutant selection?" in *International Conference on Software Engineering*. New York, NY, USA: ACM, 2010, pp. 435–444.

[51] L. Zhang, M. Gligoric, D. Marinov, and S. Khurshid, "Operator-based and random mutant selection: Better together," in *IEEE/ACM Automated Software Engineering*. ACM, 2013.

[52] M. Gligoric, A. Groce, C. Zhang, R. Sharma, M. A. Alipour, and D. Marinov, "Comparing non-adequate test suites using coverage criteria," in *ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, 2013.

[53] R. Gopinath, C. Jensen, and A. Groce, "Code coverage for suite evaluation by developers," in *International Conference on Software Engineering*. IEEE, 2014.

[54] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on test data selection: Help for the practicing programmer," *Computer*, vol. 11, no. 4, pp. 34–41, 1978.

[55] T. A. Budd, R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Theoretical and empirical studies on using program mutation to test the functional correctness of programs," in *ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. ACM, 1980, pp. 220–233.

[56] T. A. Budd, "Mutation analysis of program test data," Ph.D. dissertation, Yale University, New Haven, CT, USA, 1980.

[57] A. P. Mathur and W. E. Wong, "An empirical comparison of data flow and mutation-based test adequacy criteria," *Software Testing, Verification and Reliability*, vol. 4, no. 1, pp. 9–31, 1994.

[58] A. J. Offutt and J. M. Voas, "Subsumption of condition coverage techniques by mutation testing," Technical Report ISSE-TR-96-01, Information and Software Systems Engineering, George Mason University, Tech. Rep., 1996.

[59] K. S. H. T. Wah, "A theoretical study of fault coupling," *Software Testing, Verification and Reliability*, vol. 10, no. 1, pp. 3–45, 2000.

[60] ——, "An analysis of the coupling effect i: single test data," *Science of Computer Programming*, vol. 48, no. 2, pp. 119–161, 2003.

[61] A. J. Offutt, "The Coupling Effect : Fact or Fiction?" *ACM SIGSOFT Software Engineering Notes*, vol. 14, no. 8, pp. 131–140, Nov. 1989.

[62] ——, "Investigations of the software testing coupling effect," *ACM Transactions on Software Engineering and Methodology*, vol. 1, no. 1, pp. 5–20, 1992.

[63] W. B. Langdon, M. Harman, and Y. Jia, "Efficient multi-objective higher order mutation testing with genetic programming," *Journal of systems and Software*, vol. 83, no. 12, pp. 2416–2430, 2010.

[64] R. Gopinath, C. Jensen, and A. Groce, "Mutations: How close are they to real faults?" in *Software Reliability Engineering (ISSRE), 2014 IEEE 25th International Symposium on*, Nov 2014, pp. 189–200.

[65] D. Baldwin and F. Sayward, "Heuristics for determining equivalence of program mutations." DTIC Document, Tech. Rep., 1979.

[66] A. J. Offutt and J. Pan, "Automatically detecting equivalent mutants and infeasible paths," *Software Testing, Verification and Reliability*, vol. 7, no. 3, pp. 165–192, 1997.

[67] A. J. Offutt and R. H. Untch, "Mutation 2000: Uniting the orthogonal," in *Mutation testing for the new century*. Springer, 2001, pp. 34–44.

[68] A. T. Acree, Jr., "On mutation," Ph.D. dissertation, Georgia Institute of Technology, Atlanta, GA, USA, 1980.

[69] A. Mathur, "Performance, effectiveness, and reliability issues in software testing," in *Annual International Computer Software and Applications Conference, COMPSAC*, 1991, pp. 604–605.

[70] W. E. Wong, "On mutation and data flow," Ph.D. dissertation, Purdue University, West Lafayette, IN, USA, 1993, uMI Order No. GAX94-20921.

[71] W. Wong and A. P. Mathur, "Reducing the cost of mutation testing: An empirical study," *Journal of Systems and Software*, vol. 31, no. 3, pp. 185 – 196, 1995.

[72] A. J. Offutt, G. Rothermel, and C. Zapf, "An experimental evaluation of selective mutation," in *International Conference on Software Engineering*. IEEE Computer Society Press, 1993, pp. 100–107.

[73] R. A. DeMillo, D. S. Guindi, W. McCracken, A. Offutt, and K. King, "An extended overview of the mothra software testing environment," in

*International Conference on Software Testing, Verification and Validation Workshops*. IEEE, 1988, pp. 142–151.

[74] J. Zhang, M. Zhu, D. Hao, and L. Zhang, "An empirical study on the scalability of selective mutation testing," in *International Symposium on Software Reliability Engineering*. ACM, 2014.

[75] X. Cai and M. R. Lyu, "The effect of code coverage on fault detection under different testing profiles," in *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4. ACM, 2005, pp. 1–7.

[76] A. S. Namin and J. H. Andrews, "The influence of size and coverage on test suite effectiveness," in *ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, 2009, pp. 57–68.

[77] P. Ammann, "Transforming mutation testing from the technology of the future into the technology of the present," in *International Conference on Software Testing, Verification and Validation Workshops*. IEEE, 2015.

[78] GitHub Inc., "Software repository," http://www.github.com.

[79] Apache Software Foundation, "Apache commons," http://commons.apache.org/.

[80] M. Sridharan and A. S. Namin, "Prioritizing mutation operators based on importance sampling," in *International Symposium on Software Reliability Engineering*. IEEE, 2010, pp. 378–387.

[81] R. H. Untch, "On reduced neighborhood mutation analysis using a single mutagenic operator," in *Annual Southeast Regional Conference*, ser. ACM-SE 47. New York, NY, USA: ACM, 2009, pp. 71:1–71:4.

[82] P. Chevalley and P. Thévenod-Fosse, "A mutation analysis tool for java programs," *International journal on software tools for technology transfer*, vol. 5, no. 1, pp. 90–103, 2003.

[83] Parasoft, "Insure++," www.parasoft.com/products/insure/papers/tech_mut.htm.

[84] J. Offutt, "Insure++ critique," https://cs.gmu.edu/~offutt/documents/handouts/parasoft-anal.html.

[85] Parasoft, "Insure++ mutation analysis," http://www.parasoft.com/jsp/products/article.jsp?articleId=291&product=Insure.

[86] D. Schuler and A. Zeller, "Javalanche: Efficient mutation testing for java," in *ACM SIGSOFT Symposium on The Foundations of Software Engineering*, Aug. 2009, pp. 297–298.

[87] M. P. Usaola and P. R. Mateo, "Bacterio: Java mutation testing tool: A framework to evaluate quality of tests cases," in *Proceedings of the 2012 IEEE International Conference on Software Maintenance (ICSM)*, ser. ICSM '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 646–649.

[88] Y.-S. Ma, J. Offutt, and Y.-R. Kwon, "Mujava: A mutation system for java," in *Proceedings of the 28th International Conference on Software Engineering*, ser. ICSE '06. New York, NY, USA: ACM, 2006, pp. 827–830.

[89] I. Moore, "Jester-a junit test tester," in *International Conference on Extreme Programming*, 2001, pp. 84–87.

[90] M. G. Macedo, "Mutator," http://ortask.com/mutator/.

[91] S. Watanabe, "Information theoretical analysis of multivariate correlation," *IBM J. Res. Dev.*, vol. 4, no. 1, pp. 66–82, Jan. 1960.

[92] C. E. Shannon, "A mathematical theory of communication," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 5, no. 1, pp. 3–55, 2001.

[93] R. Gopinath, "Replication data for: Does Choice of Mutation Tool Matter?" http://eecs.osuosl.org/rahul/sqj2015.