Software Qual J (2017) 25:871–920
DOI 10.1007/s11219-016-9317-7

Does choice of mutation tool matter?

Rahul Gopinath[1] · Iftekhar Ahmed[2] · Mohammad Amin Alipour[2] · Carlos Jensen[2] · Alex Groce[2]

Published online: 10 May 2016

Abstract Though mutation analysis is the primary means of evaluating the quality of test suites, 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 mostly implement at least a few domain-specific mutation operators. Thus different tools may not always agree on the mutant kills of a test suite. Few criteria exist to guide a practitioner in choosing the right tool for either evaluating effectiveness of a test suite or for comparing different testing techniques. We investigate an ensemble of measures for evaluating efficacy of mutants produced by different tools. These include the traditional difficulty of detection, strength of minimal sets, and the diversity of mutants, as well as the information carried by the mutants produced. We find that mutation tools rarely agree. The disagreement between scores can be large, and the variation due to characteristics of the project—even after accounting for difference due to test suites—is a significant factor. 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 experiments yielding small differences in mutation score, especially using a single tool, or a small number of projects may not be reliable. There is a clear need for greater standardization of mutation analysis. We propose one approach for such a standardization.

Keywords Mutation analysis · Empirical analysis · Software testing

✉ Rahul Gopinath
gopinatr@oregonstate.edu

[1] EECS Department, Oregon State University, Corvallis OR, USA

[2] Oregon State University, Corvallis OR, USA

1 Introduction

Mutation analysis (Lipton 1971; Budd et al. 1979) 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 that most complex faults can be found by tests able to detect simpler faults (the *coupling effect*). The ability of a test suite to detect the injected mutants is taken to represent its effectiveness in detecting real faults.

Mutation analysis has been validated many times in the past. Daran and Thévenod-Fosse (1996), Andrews et al. (2005, 2006), Do and Rothermel (2006), and more recently Just et al. (2014) 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 numerous mutation analysis tools (Jia and Harman 2011; Delahaye and Du Bousquet 2013), with different tools for different languages, virtual machines, and introducing mutations at various stages—including design level and specification (Budd and Gopal 1985; Okun 2004), directly from source code (Smith and Williams 2007; Jia and Harman 2008), abstract syntax tree (Derezińska and Hałas 2014; Le et al. 2014; Just 2014), intermediate representation (Kusano and Wang 2013), byte code of various virtual machines (Coles 2016; Irvine et al. 2007), and even machine code (Duraes and Madeira 2002). This also means that there is often no direct translation between modifications carried out at a latter phase to an earlier phase,[1] or a direct first-order translation between an earlier phase and a latter one.[2] Tools may also choose to implement uncommon domain-specific mutation operators such as those targeting multi-threaded (Gligoric et al. 2010) code, using memory-related (Nanavati et al. 2015) operators, higher-order mutations (Jia and Harman 2008), object-oriented operators (Ma et al. 2002, 2005), or database-targeting operators (Zhou and Frankl 2009).

Not all mutants are similar in their fault emulation capabilities (Siami Namin et al. 2008; Offutt et al. 1996; Barbosa et al. 2001). Redundant mutants tend to add noise (Just et al. 2012; Kurtz et al. 2014), and undetected equivalent mutants deflate the measured mutation score (Schuler et al. 2009; Offutt and Craft 1994; Schuler and Zeller 2013; Nica and Wotawa 2012; Papadakis et al. 2015). There may be numerous easy-to-detect mutants, and a few stubborn mutants that are hard to find (Yao et al. 2014).

This presents a predicament for the software practitioner. There is no guideline for choosing a tool that is best suited for evaluating quality of a test suite, or for comparing multiple testing techniques. This paper proposes multiple measures to evaluate different tools and also provide a comparative benchmark for existing tools.

One measure that is often used for comparing mutants produced by different tools is to consider the mean scores obtained by different tools on similar subjects. It is assumed that the mutation tool with the lowest mean score produced the hardest to find (and hence best) mutants. However, this criteria fails to account for problems due to equivalent mutants. Equivalent mutants are undetectable and hence deflate the mutation score. This causes

[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 (Coles 2016b) for an example which does not have a direct source equivalent.

[2] See Pit return values mutator (Coles 2016b) for an example where first-order source changes imply much larger bytecode changes.

skew in favor of tools that produce numerous equivalent mutants. It is also sometimes assumed that tools generating only a small number of mutants are better than those generating a larger number of mutants for a program. In fact, some tools apply selective mutation by default, resulting in a smaller number of mutants. However, we note that selective mutation has questionable benefits (Gopinath et al. 2016, 2015) and that any large set of mutants can be reduced to a much smaller number by random sampling without significant loss of accuracy (Gopinath et al. 2015) or effectiveness (Gopinath et al. 2016). Hence, generating a smaller number of mutants is in itself not necessarily a desirable attribute.

The traditional way to evaluate a selected set of mutants is to first compute the minimum adequate test suite required for that set of mutants and then evaluate the effectiveness of that minimum test suite against the larger set from which the mutants were selected. If the minimal adequate test suite for the selected set of mutants is still adequate for the larger set of mutants, the selected set is deemed equal in effectiveness to the larger set of mutants. This evaluation criteria can be extended for non-adequate test suites trivially by considering only the detected mutants. However, we note that this criteria cannot provide a standard measure for comparing mutants from different tools because it requires a superset of mutants to compare against. Since we are trying to come up with a standard score, we have to consider any mutant that can possibly be introduced by a mutation tool. That is, the full set of mutants that is required for a standard measure is the complete set of faults a program can have, which is infeasible to evaluate.

Previous comparisons of mutation tools (Coles 2016a; Madeyski and Radyk 2010; Singh et al. 2014; Delahaye and Du Bousquet 2013) have focused on syntactic features, the number of mutants produced and tool support, with few considering the actual *semantic*[3] characteristics of mutants. Mutant semantics assumes new importance in the wake of recent questions regarding the efficacy of various mutation tools (Offut 2016a; Ammann 2015a; Offut 2016b).

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[4] of mutants produced, using the minimal set of mutants (Kintis et al. 2010; Ammann et al. 2014) originally formulated by Kintis et al. (2010) as disjoint mutant set as a measure of the utility of a mutant set, and also a slightly relaxed criterion—using non-subsumed (*surface*) mutants rather than a minimal set for comparison. We then measure the diversity[5] of a set of mutants using statistical measures such as sum of covariance [which we have shown

[3] By *semantics*, we mean the actual behavior (in contrast to the static syntax) of the mutants. That is, some mutants, while syntactically different, are actually indistinguishable in their behavior. Similarly mutants may be hard or easy to detect, and a set of mutants may encode more difference in behavior than another set. We use measures such as *mutual information* and *entropy* to measure the ability of a set of mutants to provide a diverse a behavior set.

[4] 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.

[5] Diversity of a set of mutants refers to how different one can expect any two mutants from the set to be, in terms of the tests that kill them. For example, say we have mutant set A, and killing tests given by $\{(m_1, t_1), (m_2, t_2)\}$, and mutant set B and killing tests given by $\{(m_1, t_1), (m_2, t_2), (m_3, t_3)\}$, both have similar diversity, while another set C given by $\{(m_1, t_1), (m_2, t_1)\}$ has a different diversity.

previously (Gopinath et al. 2015) 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 program behavior (Nimmer and Ernst 2002). The quality of the test suite is defined by how much of the specification it is able to accurately provide and verify (Harder et al. 2001, 2003). Hence, a set of mutants of a program may be considered to represent the program behavior with respect to the possibilities of deviation, and the information carried by a set of mutants is a reasonable measure of its 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 (Ammann 2015a).

> 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 τ_b). However, there is very little difference in mean across multiple projects.

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 small. We also note that *project* is a significant factor 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 individual project characteristics have a larger impact on the mutants produced than the tool used.

The rest of this paper is organized as follows. The previous research that is related to ours is given in Sect. 2. Our methodology is given in Sects. 3 and 3.4 details the different measures we evaluated. The results from the empirical analysis are given in Sect. 4, and its implications are discussed in Sect. 5. Threats to validity of our research are discussed in Sect. 6, and we summarize our research in Sect. 7.

2 Related work

The idea of mutation analysis was first proposed by Lipton (1971), and its main concepts were formalized by DeMillo et al. in the "Hints" (DeMillo et al. 1978) paper. The first implementation of mutation analysis was provided in the PhD thesis of Budd et al. (1980).

Previous research on mutation analysis suggests that it subsumes different coverage measures, including *statement*, *branch*, and *all-defs* dataflow coverage (Budd 1980; Mathur and Wong 1994; Offutt and Voas 1996). There is also some evidence that the faults produced by mutation analysis are similar to real faults in terms of error trace produced (Daran and Thévenod-Fosse 1996) and the ease of detection (Andrews et al. 2005, 2006). Recent research by Just et al. (2014) 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 that can detect *all* small

faults will, with high probability, detect a large portion of complex faults composed of these small faults (DeMillo et al. 1978). Evidence for the coupling effect comes from theoretical analysis by Wah (2000), Wah (2003) and empirical studies by Offutt (1989), Offutt (1992) and Langdon et al. (2010). The competent programmer hypothesis is harder to verify; however, the mean syntactic difference between faults was quantified in our previous work (Gopinath et al. 2014).

One problem in mutation analysis is the existence of equivalent mutants—mutants that are syntactically different, but semantically indistinguishable from the original program, leading to incorrect mutation score, because in general, identifying equivalent mutants is undecidable. The work on identifying equivalent mutants is generally divided into prevention and detection (Papadakis et al. 2015), with prevention focusing on reducing the incidence of equivalent mutants (Yao et al. 2014) and detection focusing on identifying the equivalent mutants by examining their static and dynamic properties. Measures for detection include efforts to identify them using compiler equivalence (Baldwin and Sayward 1979; Offutt and Craft 1994; Papadakis et al. 2015) dynamic analysis of constraint violations (Offutt and Pan 1997; Nica and Wotawa 2012), and coverage (Schuler and Zeller 2013).

A similar problem is that of redundant mutants (Just et al. 2012), 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. (2014) compared the behavior of each mutant under all tests and found a large number of redundant mutants. More recently, Papadakis et al. (2015) 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 important area of research has been reducing the cost of mutation analysis, broadly categorized as *do smarter*, *do faster*, and *do fewer* by Offutt and Untch (2000). 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 (1980) and Acree Jr (1980) 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 (1991), Wong (1993); Wong and Mathur 1995), and Offutt et al. (1993) using the DeMillo et al. (1988) mutation operators for FORTRAN.

Barbosa et al. (2001) provide guidelines for operator selection, such as considering at least one operator in each mutation class and evaluating empirical inclusion among the operators. Zhang et al. (2010) compared operator-based mutant selection techniques to random mutant sampling and found that random sampling performs as well as the operator selection methods. Zhang et al. (2013) compared various forms of sampling such as stratified random sampling based on operator strata, 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 were a sufficient sample for a 99 % correlation with the actual mutation score. The number of samples required for larger projects was found to be still smaller (Zhang et al. 2014), and recently, it was found (Gopinath et al. 2015) that 9604 mutants were sufficient

Table 1 Mutation data computed from Delahaye and Du Bousquet (2013)

Project	Test suite	Judy	Major	Pit	Jumble	Javalanche
Subject programs, test suites and mutation scores						
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
json-simple	3	51.85	21.37	58.52	53.90	68

Project	Judy	Major	Pit	Jumble	Javalanche
Number of mutants					
codec1.5	5302	5809	1826	1082	
codec1.7	7206	6922	2651	1525	
codec1.6		19,472	9544	4657	
jdom2		6699	4978	1958	
jopt-simple	1060	674	539	229	100
json-simple	677	1783	393	141	100

for obtaining 99 % accuracy for 99 % of the projects, irrespective of the independence of mutants or the size of the program.

A number of researchers have tried to approximate mutation score. Gligoric et al. (2013) found that branch coverage is highly correlated with mutation score. Cai and Lyu (2005) found that decision coverage was closely correlated with mutation coverage. Namin and Andrews (2009) found that fault detection ratio was well correlated with block coverage, decision coverage, and two different data-flow criteria. Our analysis (Gopinath et al. 2014) 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 highest correlation with mutation score.

Researchers have evaluated different mutation tools in the past. Delahaye and Du Bousquet (2013) 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 1,[6] and the correlations found (computed by us using the reported data in the paper) are given in Table 2.

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

[6] 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 2 Correlation for the mutation scores—data from Delahaye and Du Bousquet (2013)

	R^2	τ_b	% Difference μ	σ
Jumble × Judy	0.15	−0.33		
Jumble × Major	0.16	−0.33	−0.70	26.10
Jumble × Pit	0.26	0.07	−19.34	19.80
Judy × Major	1	1		
Judy × Pit	0.98	0.67		
Major × Pit	0.96	0.60	−18.64	9.70

3 Methodology

Mutation tools vary along different dimensions. As Ammann suggests in his keynote (Ammann 2015b), tools targeting different communities tend to have different priorities, with *theoretical completeness* a bone of contention between researchers and industry. Further, mutants generated in different phases of program compilation often do not have first-order equivalents in other phases. For example, changes in bytecode may not have a representation in the source code. Similarly changes in source may have a larger impact in the byte code. 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 projects and the tool 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.

3.1 Project selection

Keeping the goal of real-world projects that best represent real-world software, we looked for large Java projects in Github. We searched for projects that had at least 100 test cases and had at least 1000 lines of code. Since the number of projects thus obtained was small, we added *annotation-cli* from Github because it had close to 1000 lines of code and could be compiled and tested successfully. To ensure that our projects were representative, we also relied on projects from Apache foundation which are known to be of high quality.

We also selected only those projects that could be compiled and tested successfully using multiple mutation analysis tools. Thus we found 25 large Java projects from Github (2016) and Apache (2016), that had large test suites (Table 3).

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. (2014), Sridharan and Namin (2010), Siami Namin et al. (2008), Zhang et al. (2010), all of which use the Siemens test suites and programs (7 projects with mean 312 LOC), Zhang et al. (2013) (7 projects with mean 15,083 LOC), and Zhang et al. (2014) (12 projects with mean 6209 LOC). While our test suites are small (mean = 569, sd = 931) in comparison with most previous studies using the Siemens test suites[7]—Ammann et al. (2014) (mean = 3294, sd = 1588), Sridharan and

[7] The Siemens test suite is a test suite curated by researchers (Untch 2009), and this is at best a questionable representative for real-world test suites.

Table 3 Subject programs, test suite size, and mutation scores

Project	Test suite	Judy	Major	Pit
annotation-cli	126	42.42	43.27	59.38
asterisk-java	214	13.54	21.54	20.64
beanutils	1185	50.71	42.69	56.78
beanutils2	680	59.47	52.49	61.85
clazz	205	24.46	39.45	30.20
cli	373	71.17	76.61	86.14
collections	4407	76.99	58.63	34.69
commons-codec	605	92.72	73.52	82.66
commons-io	964	88.38	70.65	77.34
config-magic	111	55.19	29.80	60.69
csv	173	53.01	68.08	79.68
dbutils	239	44.23	65.20	47.34
events	206	77.14	70.03	59.95
faunus	172	2.55	58.65	49.07
java-api-wrapper	125	14.95	84.91	76.03
java-classmate	219	66.17	77.23	90.26
jopt-simple	566	84.50	79.32	94.50
mgwt	103	40.72	6.61	8.85
mirror	303	58.73	74.73	75.47
mp3agic	206	72.46	51.70	54.51
ognl	113	13.96	6.46	56.32
pipes	138	65.99	62.64	67.66
primitives	2276	93.35	71.33	35.71
validator	382	50.27	59.06	68.21
webbit	146	73.95	67.17	52.41
μ	569.48	55.48	56.47	59.45
σ	930.91	26.03	21.78	21.68

Namin (2010) (mean = 3115, sd = 1572), Siami Namin et al. (2008) (mean = 3115, sd = 1572), Zhang et al. (2010) (mean = 3115, sd = 1572), Zhang et al. (2013) (mean = 3115, sd = 1572), and Zhang et al. (2014) (mean = 81, sd = 29), we assume that the number and size of projects, and the extent of comparison more than makes up for it.

3.2 Tool selection

We started our evaluation with the list of all known tools for Java which were available (the first mutation system, JavaMut (Chevalley and Thévenod-Fosse 2003) is no longer available). We also discarded Insure++ (Parasoft 2014) which did not actually implement mutation testing (Offutt 2016; Parasoft 2015). The tools we investigated were Pit (Coles 2016), Major (Just 2014), Judy (Madeyski and Radyk 2010), Javalanche (Schuler and Zeller 2009), Bacterio (Usaola and Mateo 2012), MuJava (Ma et al. 2006), Jumble (Irvine et al. 2007), Jester (Moore 2001), and Mutator (Macedo 2016). Our choice of mutation tools for assessment was 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 (kill matrix). This

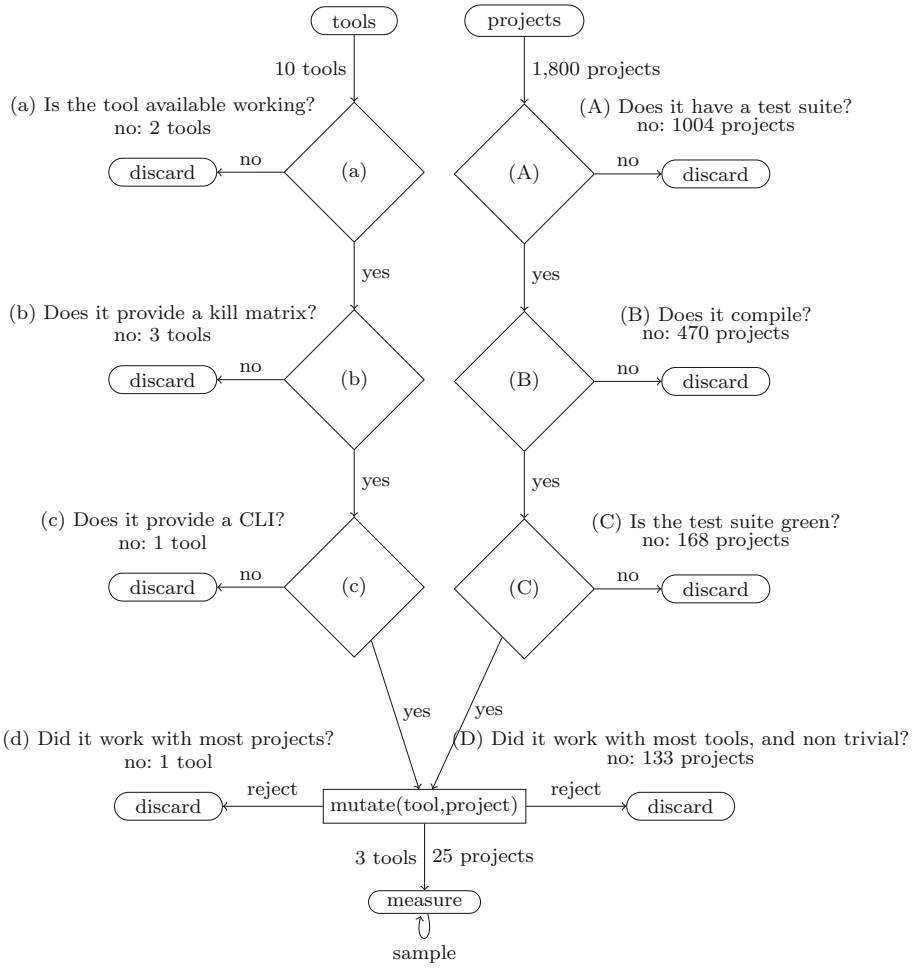

Fig. 1 The process of selection

eliminated Mutator, Jester, and Jumble. 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.

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 environment.[8]

Third, and most importantly, the tools had to work with a majority of the projects and test suites we had. MuJava could not handle package hierarchies. Examination of the source suggested that fixing this shortcoming was non-trivial. We discarded Javalanche for several issues: (1) Javalanche had 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; eliminating these,

[8] 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.

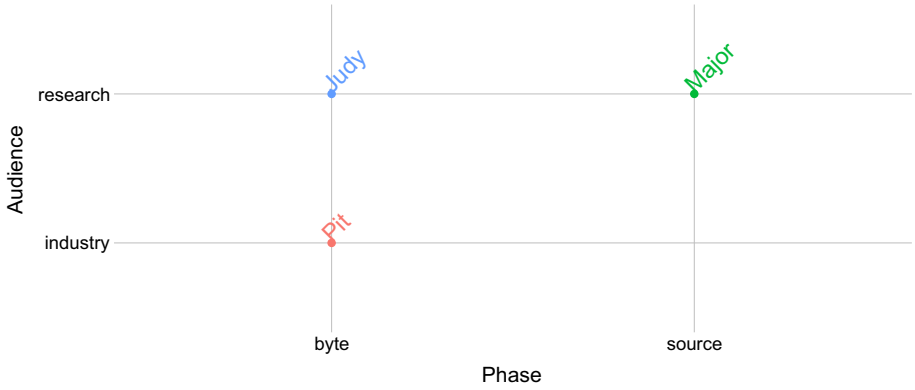

Fig. 2 Tools used for benchmark

the tests that remained were a small fraction of the original test suites. (2) Javalanche was last updated in 2012. (3) Javalanche uses only selective mutation, while other tools examined leave that choice to the tester. (4) We note that Javalanche could not complete successfully for a majority of the projects in the previous comparative study by Delahaye and Du Bousquet (2013). Hence, we removed both MuJava and Javalanche from the benchmark. Our process of selection is detailed in Fig. 1.

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 and 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 Fig. 2 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, Pit 1.0[9]) (Fig. 2).

3.3 Analysis

For each tool, we used the settings for the maximum number of operators to mutate. 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 4. A boxplot of the number of mutants by each tool is given in Fig. 3. Unfortunately, this also means that the mutation scores do not necessarily agree as we see in Table 3. 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, we removed the mutants that were not killed by any of the test cases as done in similar studies (Siami Namin et al.

[9] In the case of Pit, we extended Pit to provide a more complete set of mutants, a modification which was latter accepted to the main line (Pit 1.0).

Table 4 Number of mutants by tools in subject programs

Project	LOC	Judy	Major	Pit
annotation-cli	870	777	512	981
asterisk-java	29,477	12,658	5812	15,476
beanutils	11,640	6529	4382	9665
beanutils2	2251	990	615	2069
clazz	5681	2784	2022	5165
cli	2667	2308	1411	2677
collections	25,400	1006	10,301	24,141
commons-codec	6603	44	7362	9953
commons-io	9472	164	6486	9799
config-magic	1251	527	650	1181
csv	1384	1154	991	1798
dbutils	2596	1159	677	1922
events	1256	2353	615	1155
faunus	9000	3723	3771	9668
java-api-wrapper	1760	929	611	1711
java-classmate	2402	1423	952	2543
jopt-simple	1617	497	695	1790
mgwt	16,250	1394	6654	12,030
mirror	2590	1316	449	1876
mp3agic	4842	1272	4822	7182
ognl	13,139	8243	5616	21,227
pipes	3513	590	1171	3001
primitives	11,965	14	4916	11,312
validator	5807	3320	3655	5846
webbit	5018	144	1327	3707
μ	7138.04	2212.72	3059	6715
σ	7471.65	2931.64	2786.07	6369.23

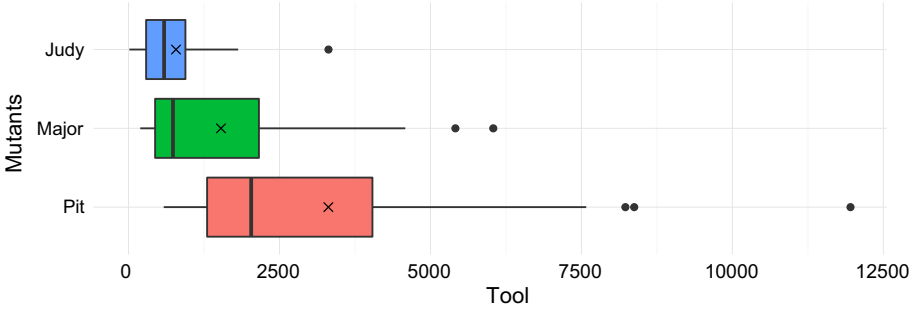

Fig. 3 Number of mutants produced by different tools across all projects in our sample. The *cross in the center* is the mean, while the *central black line* is the median

2008; Zhang et al. 2010, 2013, 2014). We call the original set the *raw mutant set* (vs. *refined mutant set*).

3.3.1 Sampling

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 the complete set of 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. 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. This sample size is of sufficient accuracy as recommended in our previous study on mutant sampling (Gopinath et al. 2015).

3.4 Measures

We considered multiple measures that can lead to insights about the characteristics of mutants. For each, we rely on two different measures of correlation—R^2 and Kendall's τ_b. For a detailed discussion on why both were used, see "Appendix".

We also provide the mean difference between the scores (denoted by *Difference* μ in the table) and the standard deviation (denoted by σ in the table) for this measurement. The mean difference is important as it provides the effect size—the consistent difference between 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,[10] they may not be practically different if the mean difference is in small 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.

For each measure, one common question was whether the phase of generation or target audience has an impact. To answer this question, we rely on *analysis of variance*. By running *ANOVA*[11] 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 Eq. 1, where we compare the ability of each model to predict the variability in the measure being analyzed.

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

[10] 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.

[11] Analysis of variance—*ANOVA*—is a statistical procedure used to compare the goodness of fit of statistical models. It can tell us whether 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.

Another question is the impact of tools after controlling for the number of mutants produced. To answer this question, we sampled 100 mutants at a time from each project 100 times.

We use the following measures for evaluating the quality of mutants.

3.4.1 Raw mutation score

The 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. There is little gain in randomly sampling test suites both for raw and refined mutation analysis. Hence, we chose to sample the test suites only for computing *refined mutation score*.

3.4.2 Refined mutation score

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—cannot be fulfilled. As done in previous studies (Siami Namin et al. 2008; Zhang et al. 2010, 2013, 2014), we remove the mutants that were not detected from our pool, leaving mutants that were detected by at least one test case. Next, we randomly sample progressively smaller fractions of test suites, 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. This is repeated 100 times, and the mutation scores of each sample are taken.

3.4.3 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. (2014) 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 are guaranteed to detect the latter—which in effect means that the latter is weaker than the former. A minimal test suite for a set of mutants is any test suite from which removing even a single test case causes the mutation score to drop. 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.

An adequate test suite is called a *minimum test suite* when it is the smallest test suite that is sufficient to kill *all* mutants produced from a program. A set of mutants is called a *minimal set of mutants* or a *disjoint set of mutants* when such a set is the smallest set of mutants that requires all test cases in the minimum test suite for complete detection.

3.4.4 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 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. (2014)

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, which we call *surface mutants*.

A set of test cases is called *unique* or *distinct* when no two tests in that set have the same mutant kills. A set of mutants is called a *surface set of mutants* when such a set is the smallest set of mutants that requires all test cases in a unique test suite for complete detection.

3.4.5 Covariance between mutants

We have shown previously (Gopinath et al. 2015) that for mutation analysis, the maximum number of mutants to be sampled for a given tolerance has an upper bound provided by the binomial distribution. 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. For a detailed discussion on *covariance*, see "Appendix". The most useful set of mutants only includes mutants that are completely independent of each other, and the least useful set of mutants only includes mutants that are completely dependent (redundant).

3.4.6 Mutual information 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*. See "Appendix" for further information on *mutual information*.

3.4.7 Entropy carried by mutants

The measures we introduced (minimal set, surface set, covariance) evaluated the redundancy in a set of mutants. Another way to think about a set of mutants is to think of mutants as expressing all possible divergence from the specifications of a program, and a test suite can be thought of as carrying information about the specification of the program. This suggests that a measure of information contained in the mutant×test-case matrix can be reasonable measure of goodness of a set of mutants. See "Appendix" for more information on *entropy*.

4 Results

4.1 Raw mutation score

We use different visualizations to inspect the distribution of mutation scores. Figures 4 and 5 show the distribution of mutation score, and the mean values, respectively. Note that this

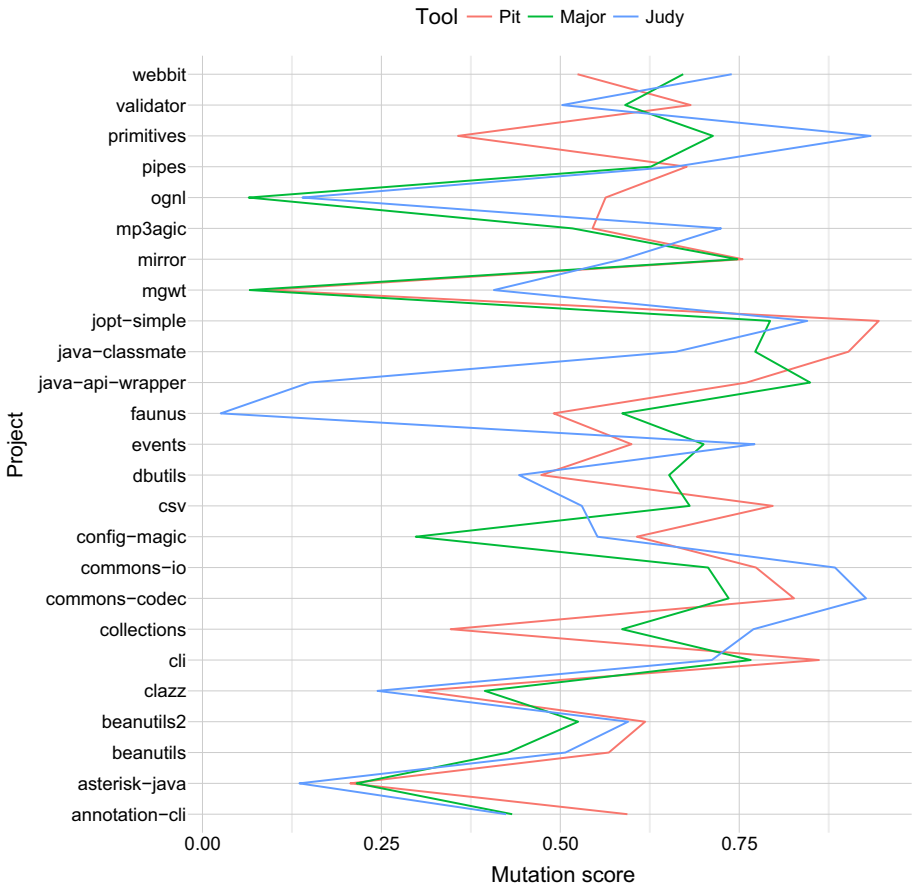

Fig. 4 Distribution of raw mutation score by different tools: mutation scores from different tools rarely agree, and none of the tools produce a consistently larger or smaller score compared to the other tools

is for raw mutants (without removing equivalent mutants and sampling). The correlations between mutation tools are given in Table 5.

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?*

As Table 6 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 for *raw mutation scores*.

4.2 Refined mutation scores

The mutation scores using randomly sampled fractions of the original test cases are given in Table 7.

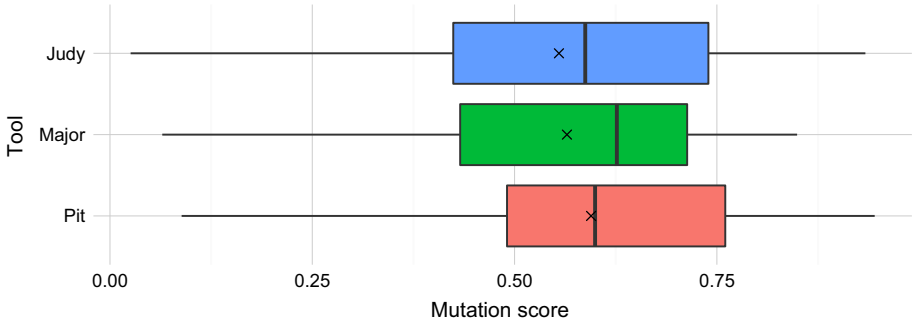

Fig. 5 Mean raw mutation score produced by different tools. Mutation scores produced by different tools are on average not consistently larger or smaller compared to other tools

Table 5 Correlation between mutation tools

	R^2	τ_b	% Difference μ	σ
Judy × Pit	0.37	0.27	−3.97	26.93
Judy × Major	0.52	0.41	−0.99	23.72
Pit × Major	0.67	0.54	2.98	17.53

Table 6 Model fit—mutation score for raw mutants

	df	Sum sq	Mean sq	*F* value	$Pr(>F)$
Model fit with phase $R^2 = 0.5$					
Project	24	2.62	0.11	4.12	0.0000
Phase	1	0	0	0.06	0.8034
Residuals	49	1.30	0.03		
Model fit with audience $R^2 = 0.51$					
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 tool $R^2 = 0.5$					
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		
Base model fit $R^2 = 0.51$					
Project	24	2.62	0.11	4.20	0.0000
Residuals	50	1.30	0.03		

Figure 6 visualizes the relationship of mutation scores by different tools. In the figure, the fraction of test suite determines the darkness of the point. Larger fractions have darker colors. We can see that the light colors cluster near the origin, while darker colors cluster around unity as expected (larger fractions of test suite will have higher mutation scores, and hence darker colors).

The refined mutation scores produced after removing the undetected mutants also tend to follow the same pattern. In Table 7, we see that the maximum R^2 is 0.69, with high

Table 7 Correlation between mutation tools for refined mutants ($\frac{1}{2^x}$ test suite sample)

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

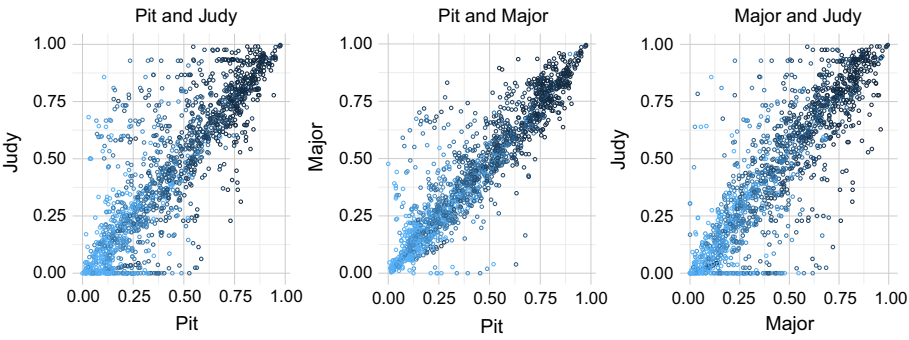

Fig. 6 The mutation scores of different tools plotted against each other for different fractions of the full test suite. The mutation score at larger fractions of the full test suites are *colored darker*

spread. Further, maximum Kendall's τ_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 tools produce mutants with consistently higher or lower mutation scores.

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

The *ANOVA* in Table 8 shows no evidence to suggest that *phase* and *audience* contribute toward model fit for the *refined mutation score*.

Table 8 Model fit with refined mutation score after removing undetected mutants

	df	Sum sq	Mean sq	*F* value	$Pr(>F)$
Model fit with phase $R^2 = 0.098$					
Project	24	34.15	1.42	21.33	0.0000
Phase	1	0	0	0.01	0.9084
Residuals	4474	298.41	0.07		
Model fit with audience $R^2 = 0.098$					
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 tool $R^2 = 0.098$					
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		
Base model fit $R^2 = 0.098$					
Project	24	34.15	1.42	21.34	0.0000
Residuals	4475	298.42	0.07		

4.3 Minimal set of mutants

Figure 7a provides the mean and variance for the size of the minimum mutant set for all projects. The size of minimal mutant set is given in Table 9, and a comparison between the tools is given in Table 10. 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 produce larger-sized minimal sets are better.

Table 10 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 τ_b is strong (maximum 0.85 between Pit and Major), suggesting a stronger similarity between Pit and Major.

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

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

The *ANOVA* (Table 11) suggests that *audience* is a statistically significant ($p < 0.05$) factor. When comparing R^2 of models containing both project and the factors investigated, the variation explained by models incorporating a given factor in addition to *project* is as follows: *phase* 46 %, *audience* 49 %, *tool* 58 %. Investigating the increase in explanatory power of complex models over a model containing only *project*, the increase in R^2 (explanatory power) was as follows: *phase* 1.1 %, *audience* 4.4 %, *tool* 13 %.

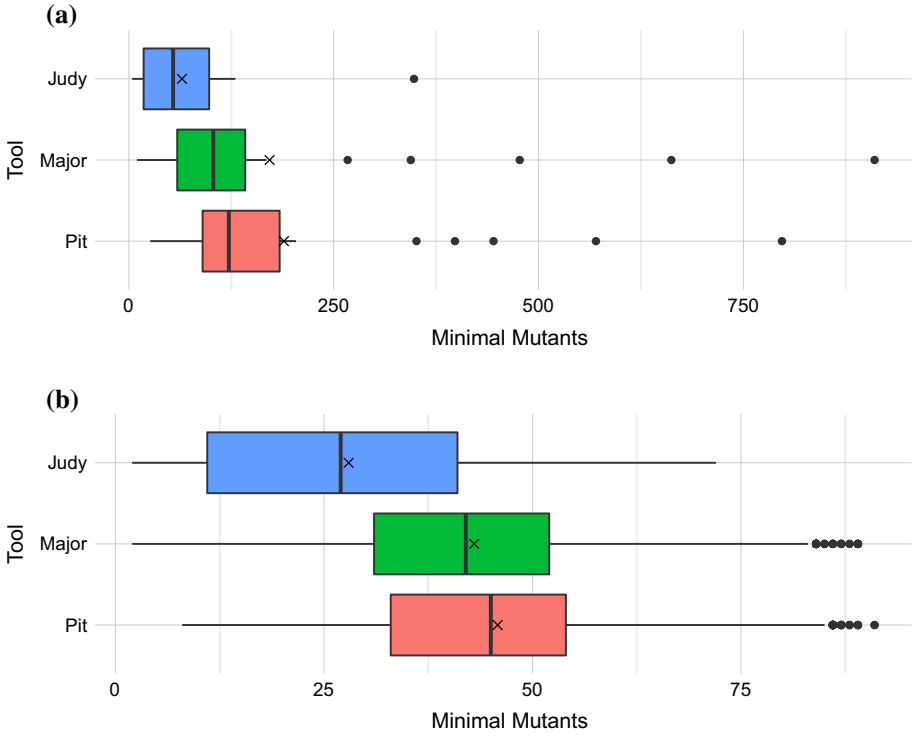

Fig. 7 Minimal mutant set sizes of tools—larger is better. **a** Full mutant set. **b** 100 mutant samples

4.3.1 What is the impact of tools after controlling for the number of mutants produced?

Figure 7b provides the mean and variance for the size of minimum mutant set for all projects when controlling for the number of mutants.

Table 12 suggests that the correlation between Pit and Major is very strong (0.93) and that the correlation between Judy and Major improved. We find the same with Kendall's τ_b, with strong correlation between Pit and Major (0.73). Finally, the 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 *ANOVA* to answer this question, for which models are given in Eq. 1. The measure is the size of the minimum mutant set, after controlling for the number of mutants.

The *ANOVA* (Table 13) suggests that *phase* and *audience* are statistically significant ($p < 0.05$) factors. When comparing R^2 of models containing both project and the factors investigated, the variation explained by models incorporating a given factor in addition to *project* is as follows: *phase* 65 %, *audience* 69 %, *tool* 79 %. Investigating the increase in explanatory power of complex models over a model containing only *project*, the increase in R^2 (explanatory power) was as follows: *phase* 2.2 %, *audience* 6.2 %, *tool* 16 %.

Table 9 Minimal set of mutants from different mutation tools

Project	Judy	Major	Pit
annotation-cli	20	20	26
asterisk-java	121	142	171
beanutils	348	344	398
beanutils2	67	105	145
clazz	18	59	49
cli	106	130	136
collections	130	910	797
commons-codec	4	267	351
commons-io	33	477	570
config-magic	33	45	49
csv	64	91	99
dbutils	73	60	104
events	21	10	30
faunus	7	103	122
java-api-wrapper	10	42	90
java-classmate	98	108	184
jopt-simple	67	95	131
mgwt	55	70	74
mirror	127	112	173
mp3agic	54	108	116
ognl	14	27	81
pipes	29	81	95
primitives	9	662	445
validator	102	168	204
webbit	9	59	90
μ	64.76	171.80	189.20
σ	71.94	215.33	185.83

Table 10 Correlation between minimal set of mutants from different mutation tools

	R^2	τ_b	Difference μ	σ
Judy × Pit	0.34	0.34	−124.44	175.05
Judy × Major	0.26	0.35	−107.04	208.59
Pit × Major	0.96	0.85	17.40	62.86

4.4 Surface mutants

Surface mutants are given in Table 14, and the relation between different tools is given in Table 15. The mean surface mutant sizes are plotted in Fig. 8a.

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 11 Model fit with minimum mutant set

	df	Sum sq	Mean sq	*F* value	$Pr(>F)$
Model fit with phase $R^2 = 0.46$					
Project	24	1,440,910.19	60,037.92	3.59	0.0001
Phase	1	33,480.54	33,480.54	2	0.1631
Residuals	49	818,380.79	16,701.65		
Model fit with audience $R^2 = 0.49$					
Project	24	1,440,910.19	60,037.92	3.83	0.0000
Audience	1	83,827.44	83,827.44	5.35	0.0250
Residuals	49	768,033.89	15,674.16		
Model fit with tool $R^2 = 0.58$					
Project	24	1,440,910.19	60,037.92	4.61	0.0000
Tool	2	227,046.96	113,523.48	8.72	0.0006
Residuals	48	624,814.37	13,016.97		
Base model fit $R^2 = 0.45$					
Project	24	1,440,910.19	60,037.92	3.52	0.0001
Residuals	50	851,861.33	17,037.23		

Table 12 Minimal set of mutants from different mutation tools (100 samples)

	R^2	τ_b	Difference μ	σ
Judy × Pit	0.49	0.35	−17.86	17.97
Judy × Major	0.52	0.40	−15.05	17.49
Pit × Major	0.93	0.73	2.82	7.01

Table 15 suggests that the R^2 correlation between Pit and Major is strong (0.94), while that between Judy and Pit is low (0.27) and that between Judy and Major is low (0.25). Similarly with Kendall's τ_b, it ranges from 0.76 to 0.24. 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?*

We use *ANOVA* to answer this question, for which models are given in Eq. 1. The measure is the size of surface mutant set.

The *ANOVA* (Table 16) suggests that *audience* is a statistically significant ($p < 0.05$) factor. When comparing R^2 of models containing both project and the factors investigated, the variation explained by models incorporating a given factor in addition to *project* is as follows: *phase* 41 %, *audience* 50 %, *tool* 58 %. Investigating the increase in explanatory power of complex models over a model containing only *project*, the increase in R^2 (explanatory power) was as follows: *phase* −0.25 %, *audience* 8.6 %, *tool* 16 %.

Table 13 Model fit with minimal mutant set (100 samples)

	df	Sum sq	Mean sq	*F* value	$Pr(>F)$
Model fit with phase $R^2 = 0.65$					
Project	24	1,812,892.77	75,537.20	569.70	0.0000
Phase	1	62,358.74	62,358.74	470.31	0.0000
Residuals	7474	990,978.33	132.59		
Model fit with audience $R^2 = 0.69$					
Project	24	1,812,892.77	75,537.20	645.12	0.0000
Audience	1	178,199.56	178,199.56	1521.89	0.0000
Residuals	7474	875,137.50	117.09		
Model fit with tool $R^2 = 0.79$					
Project	24	1,812,892.77	75,537.20	953.47	0.0000
Tool	2	461,297.59	230,648.79	2911.36	0.0000
Residuals	7473	592,039.48	79.22		
Base model fit $R^2 = 0.63$					
Project	24	1,812,892.77	75,537.20	536.05	0.0000
Residuals	7475	1,053,337.06	140.91		

4.4.1 *What is the impact of tools after controlling for the number of mutants produced?*

The correlation between different tools is given in Table 17. The mean surface mutant sizes after controlling number of mutants are plotted in Fig. 8b.

As we expect from the values for minimal mutants, controlling for the number of mutants has a large impact. Table 17 suggests that the correlation between Pit and Major is very strong (0.92), while that between Judy and Major improved, as did the correlation between Judy and Pit. Similarly for Kendall's τ_b it ranges from 0.72 to 0.34.

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

We use *ANOVA* to answer this question, for which models are given in Eq. 1. The measure is the size of the surface mutant set, after controlling for the number of mutants.

The *ANOVA* (Table 18) suggests that *phase* and *audience* are statistically significant ($p < 0.05$) factors. When comparing R^2 of models containing both project and the factors investigated, the variation explained by models incorporating a given factor in addition to *project* is as follows: *phase* 64 %, *audience* 68 %, *tool* 78 %. Investigating the increase in explanatory power of complex models over a model containing only *project*, the increase in R^2 (explanatory power) was as follows: *phase* 2.2 %, *audience* 6 %, *tool* 16 %.

4.5 Covariance between mutants

Figure 9a 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 a smaller sum of covariance are better.

Table 14 Surface set of mutants from different mutation tools

Project	Judy	Major	Pit
annotation-cli	29	20	31
asterisk-java	148	170	211
beanutils	478	428	548
beanutils2	80	105	149
clazz	24	73	64
cli	157	174	207
collections	148	995	898
commons-codec	6	364	488
commons-io	30	552	692
config-magic	42	50	60
csv	67	104	139
dbutils	78	69	124
events	25	22	25
faunus	8	126	179
java-api-wrapper	12	72	137
java-classmate	116	136	217
jopt-simple	90	118	177
mgwt	58	77	85
mirror	148	124	205
mp3agic	88	149	160
ognl	20	39	379
pipes	41	110	130
primitives	10	723	685
validator	140	218	273
webbit	15	69	114
μ	82.32	203.48	255.08
σ	97.04	237.65	230.22

Table 15 Correlation between surface set of mutants from different mutation tools

	R^2	τ_b	Difference μ	σ
Judy × Pit	0.27	0.24	−172.76	223.98
Judy × Major	0.25	0.31	−121.16	233.17
Pit × Major	0.94	0.76	51.60	78.48

Table 19 provides the sum of covariance for different projects, and Table 20 provides the correlation between the sum of covariance from different tools. Table 20 suggests that in terms of sum of covariance, Judy and Pit are in closest agreement (0.45), with medium correlation, while the correlation between other tools is low. However, Kendall's τ_b indicates a medium correlation between all tools (0.43, 0.43, 0.57).

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

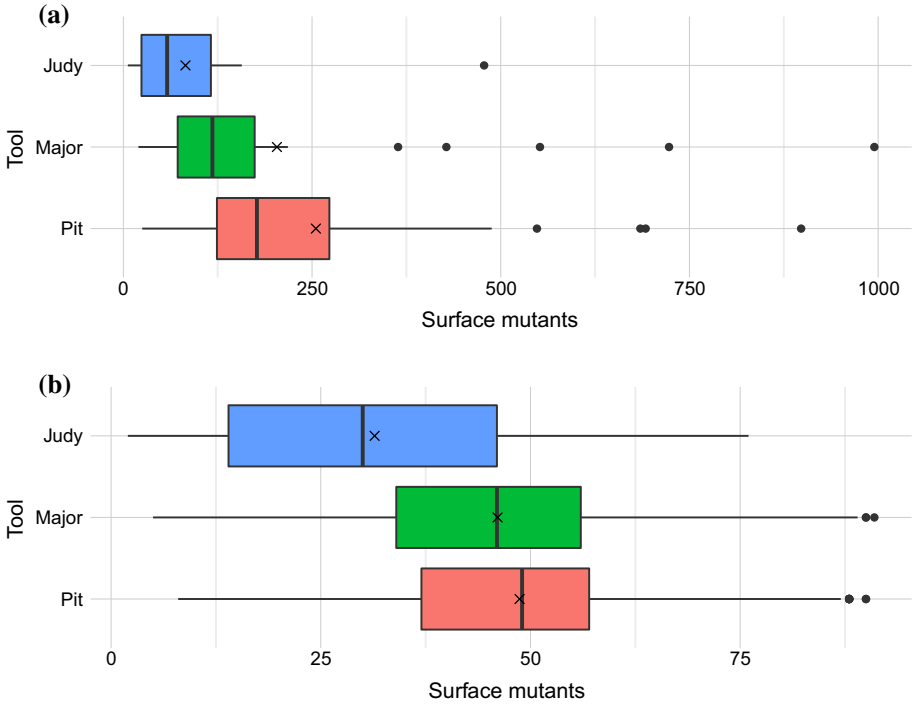

Fig. 8 Surface mutant sizes between tools—larger is better. **a** Full mutant set. **b** 100 mutant samples

Table 16 Model fit for surface mutants

	df	Sum sq	Mean sq	*F* value	$Pr(>F)$
Model fit with phase $R^2 = 0.41$					
Project	24	1,967,892.21	81,995.51	3.19	0.0003
Phase	1	20,160.81	20,160.81	0.78	0.3800
Residuals	49	1,258,614.53	25,686.01		
Model fit with audience $R^2 = 0.5$					
Project	24	1,967,892.21	81,995.51	3.76	0.0000
Audience	1	209,739.21	209,739.21	9.61	0.0032
Residuals	49	1,069,036.13	21,817.06		
Model fit with tool $R^2 = 0.58$					
Project	24	1,967,892.21	81,995.51	4.44	0.0000
Tool	2	393,236.03	196,618.01	10.66	0.0001
Residuals	48	885,539.31	18,448.74		
Base model fit $R^2 = 0.42$					
Project	24	1,967,892.21	81,995.51	3.21	0.0003
Residuals	50	1,278,775.33	25,575.51		

Table 17 Correlation between surface set of mutants from different mutation tools (100 samples)

	R^2	τ_b	Difference μ	σ
Judy × Pit	0.47	0.34	−17.28	18.12
Judy × Major	0.51	0.39	−14.66	17.69
Pit × Major	0.92	0.72	2.62	7.08

Table 18 Model fit for surface mutants (100 samples)

	df	Sum sq	Mean sq	*F* value	$Pr(>F)$
Model fit with phase $R^2 = 0.64$					
Project	24	1,708,095.67	71,170.65	539.73	0.0000
Phase	1	60,356.53	60,356.53	457.72	0.0000
Residuals	7474	985,549.60	131.86		
Model fit with audience $R^2 = 0.68$					
Project	24	1,708,095.67	71,170.65	603.93	0.0000
Audience	1	165,130.22	165,130.22	1,401.25	0.0000
Residuals	7474	880,775.91	117.85		
Model fit with tool $R^2 = 0.78$					
Project	24	1,708,095.67	71,170.65	868.84	0.0000
Tool	2	433,760.06	216,880.03	2,647.64	0.0000
Residuals	7473	612,146.07	81.91		
Base model fit $R^2 = 0.62$					
Project	24	1,708,095.67	71,170.65	508.65	0.0000
Residuals	7475	1,045,906.13	139.92		

We use *ANOVA* to answer this question, for which models are given in Eq. 1. The measure is the *sum of covariance* of mutant kill matrix.

The *ANOVA* (Table 21) suggests that *audience* is a statistically significant ($p < 0.05$) factor. When comparing R^2 of models containing both project and the factors investigated, the variation explained by models incorporating a given factor in addition to *project* is as follows: *phase* 14 %, *audience* 18 %, *tool* 17 %. Investigating the increase in explanatory power of complex models over a model containing only *project*, the increase in R^2 (explanatory power) was as follows: *phase* 0.62 %, *audience* 5 % *tool* 3.4 %. We note that project is not a statistically significant factor.

4.5.1 *What is the impact of tools after controlling for the number of mutants produced?*

Figure 9b shows the mean covariance across the different tools after controlling for the number of mutants.

Table 22 provides the correlation between sum of covariance from different tools after controlling for the number of mutants. We see that both R^2 and τ_b increase across all the

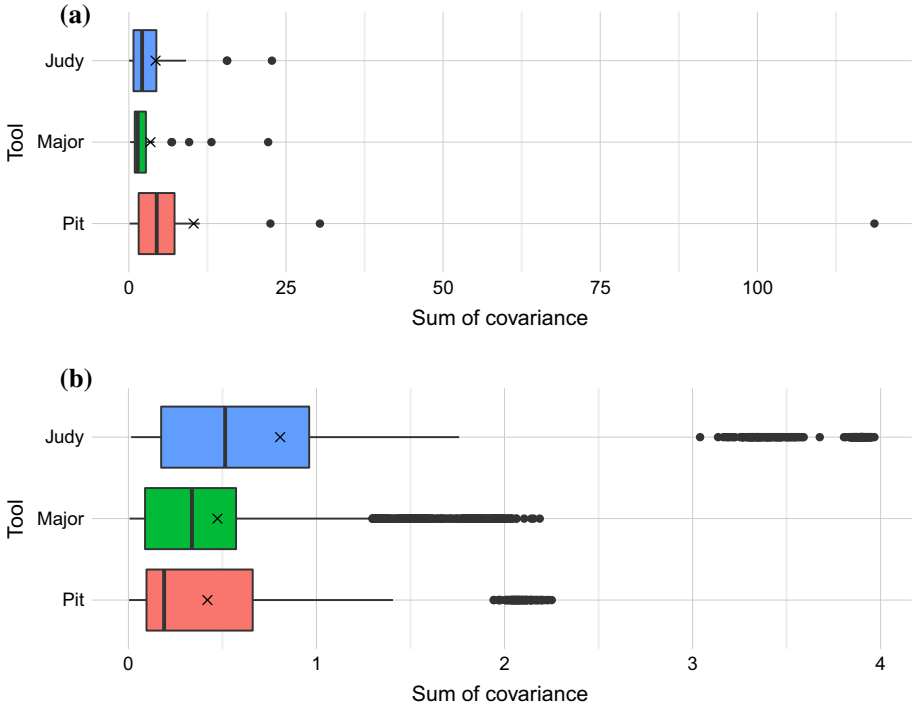

Fig. 9 Covariance between mutants produced—larger is better. **a** Full mutant set. **b** 100 samples

tools, with the Pit and Major correlation being highest (0.68), with medium correlation. Similar improvement is also seen over Kendall's τ_b—highest is between Judy and Major (0.66).

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

We use *ANOVA* to answer this question, for which models are given in Eq. 1. The measure is the *sum of covariance* of mutant kill matrix, after controlling for the number of mutants.

The *ANOVA* (Table 23) suggests that *phase* and *audience* are statistically significant ($p < 0.05$) factors. When comparing R^2 of models containing both project and the factors investigated, the variation explained by models incorporating a given factor in addition to *project* is as follows: *phase* 65 %, *audience* 66 %, *tool* 68 %. Investigating the increase in explanatory power of complex models over a model containing only *project*, the increase in R^2 (explanatory power) was as follows: *phase* 0.46 %, *audience* 1.5 %, *tool* 4.1 %. We note that project is statistically significant once number of mutants is controlled.

4.6 Mutual information (total correlation) between mutants

Figure 10a shows the mean mutual information between mutants produced.

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

Table 19 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
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	1.57
primitives	0.03	0.20	0.15
validator	1.02	0.94	5.90
webbit	4.16	13.15	22.53
μ	4.25	3.44	10.31
σ	5.73	5.05	23.64

Table 20 Correlation of sum of covariance between different mutation tools

	R^2	τ_b	Difference μ	σ
Judy × Pit	0.45	0.43	−6.06	21.65
Judy × Major	0.29	0.43	0.81	6.45
Pit × Major	0.19	0.57	6.87	23.23

Table 24 shows the mutual information of different tools across projects, and Table 25 provides the correlation of mutual information by different tools.

Table 25 suggests that the correlation between different tools is rather weak in terms of both R^2 (0.29 to −0.063) and τ_b (0.43 to −0.14).

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

We use *ANOVA* to answer this question, for which models are given in Eq. 1. The measure is the *mutual information* of mutant kill matrix.

Table 21 Model fit for covariance

	df	Sum sq	Mean sq	*F* value	$Pr(>F)$
Model fit with phase $R^2 = 0.14$					
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 audience $R^2 = 0.18$					
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 tool $R^2 = 0.17$					
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		
Base model fit $R^2 = 0.13$					
Project	24	6407.53	266.98	1.47	0.1261
Residuals	50	9103.91	182.08		

Table 22 Correlation of sum of covariance between different mutation tools (100 samples)

	R^2	τ_b	Difference μ	σ
Judy × Pit	0.50	0.61	0.34	0.86
Judy × Major	0.60	0.66	0.29	0.80
Pit × Major	0.68	0.61	−0.05	0.41

Table 23 Model fit for covariance (100 samples)

	df	Sum sq	Mean sq	*F* value	$Pr(>F)$
Model fit with phase $R^2 = 0.65$					
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 audience $R^2 = 0.66$					
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 tool $R^2 = 0.68$					
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		
Base model fit $R^2 = 0.64$					
Project	24	2320.14	96.67	540.43	0.0000
Residuals	7175	1283.46	0.18		

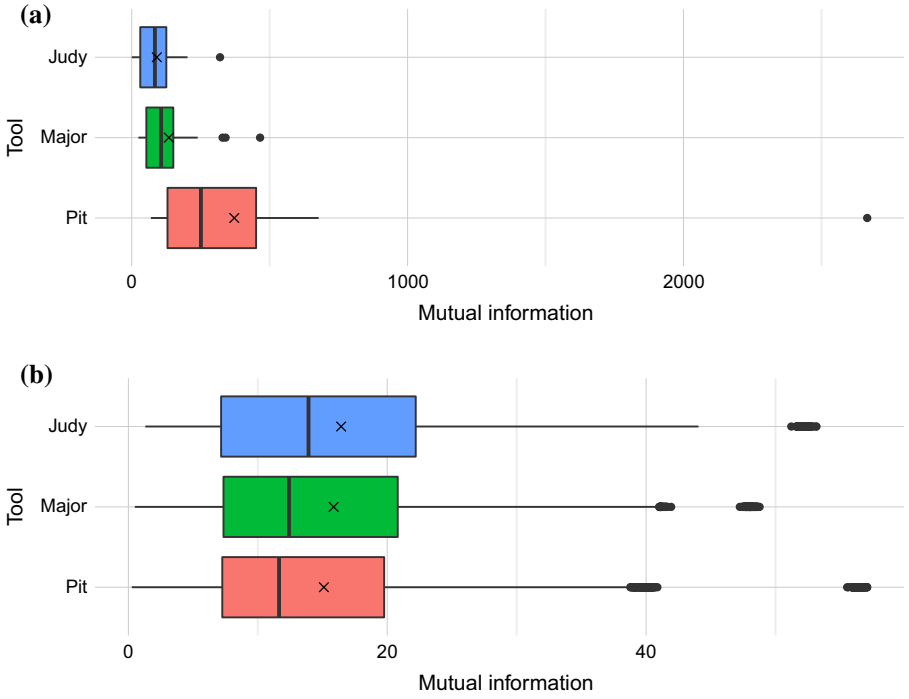

Fig. 10 Mutual Information of the mutant set produced—larger is better. **a** Full mutant set. **b** 100 samples

The *ANOVA* (Table 26) suggests that *audience* is a statistically significant ($p < 0.05$) factor. When comparing R^2 of models containing both project and the factors investigated, the variation explained by models incorporating a given factor in addition to *project* is as follows: *phase* −0.015 %, *audience* 19 %, *tool* 18 %. Investigating the increase in explanatory power of complex models over a model containing only *project*, the increase in R^2 (explanatory power) was as follows: *phase* −64 %, *audience* −46 %, *tool* −47 %.

4.6.1 What is the impact of tools after controlling for the number of mutants produced?

Figure 10b shows the mean mutual information between mutants produced, after controlling for the number of mutants.

Table 27 provides the correlation of mutual information by different tools after controlling number of mutants. Table 27 suggests that the correlation between different tools improves across all tools in terms of both R^2 (0.86–0.78) and τ_b (0.59–0.53).

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

We use *ANOVA* to answer this question, for which models are given in Eq. 1. The measure is the *mutual information* of the mutant kill matrix, after controlling for the number of mutants.

The *ANOVA* (Table 28) suggests that *phase* and *audience* are statistically significant ($p < 0.05$) factors. When comparing R^2 of models containing both project and the factors investigated, the variation explained by models incorporating a given factor in addition to *project* is as follows: *phase* 90 %, *audience* 90 %, *tool* 90 %. Investigating the increase in

Table 24 Mutual information different mutation tools

Project	Judy	Major	Pit
annotation-cli	90.94	58	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
μ	91.07	134.59	371.77
σ	75.57	110.12	507.05

Table 25 Mutual information correlation of different mutation tools

	R^2	τ_b	Difference μ	σ
Judy × Pit	0.29	0.19	−280.70	490.79
Judy × Major	−0.06	−0.14	−43.52	137.45
Pit × Major	0.10	0.43	237.19	507.78

explanatory power of complex models over a model containing only *project*, the increase in R^2 (explanatory power) was as follows: *phase* 0.06 %, *audience* 0.039 %, *tool* 0.065 %.

4.7 Entropy carried by mutants

The entropy of the set of mutants produced by each tool is given in Table 29, and the comparison between different tools in terms of entropy is given in Table 30. Figure 11a shows the entropy carried by the mutants.

Table 26 Model fit for covariance

	df	Sum sq	Mean sq	*F* value	$Pr(>F)$
Model fit with phase $R^2 = -0.00015$					
Project	24	2,457,752.40	102,406.35	0.98	0.5078
Phase	1	156,285.44	156,285.44	1.49	0.2274
Residuals	49	5,125,806.30	104,608.29		
Model fit with audience $R^2 = 0.19$					
Project	24	2,457,752.40	102,406.35	1.20	0.2840
Audience	1	1,117,537.88	1,117,537.88	13.15	0.0007
Residuals	49	4,164,553.86	84,990.90		
Model fit with tool $R^2 = 0.18$					
Project	24	2,457,752.40	102,406.35	1.19	0.2997
Tool	2	1,141,207.98	570,603.99	6.61	0.0029
Residuals	48	4,140,883.76	86,268.41		
Base model fit $R^2 = 0.64$					
Project	24	2320.14	96.67	540.43	0.0000
Residuals	7175	1283.46	0.18		

Table 27 Mutual information correlation of different mutation tools (100 samples)

	R^2	τ_b	Difference μ	σ
Judy × Pit	0.86	0.53	0.14	6.48
Judy × Major	0.85	0.59	−0.42	6.73
Pit × Major	0.78	0.58	−0.56	8.13

Impact of entropy: Mutant sets with higher entropy carry more information. Hence, tools that produce mutant sets with higher entropy are better.

We compare the number of mutants produced by each tool (Fig. 3) and entropy from mutants by each (Fig. 11a). As expected, a larger number of mutants can carry more information.

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

We use *ANOVA* to answer this question, for which models are given in Eq. 1. The measure is the *entropy* of the mutant kill matrix.

The *ANOVA* (Table 31) suggests that *audience* is a statistically significant ($p < 0.05$) factor. When comparing R^2 of models containing both project and the factors investigated, the variation explained by models incorporating a given factor in addition to *project* is as follows: *phase* 0.66 %, *audience* 8 %, *tool* 41 %. Investigating the increase in explanatory power of complex models over a model containing only *project*, the increase in R^2 (explanatory power) was as follows: *phase* 6.9 %, *audience* 14 %, *tool* 47 %.

Table 28 Model fit for mutual information (100 samples)

	df	Sum sq	Mean sq	*F* value	$Pr(>F)$
Model fit with phase $R^2 = 0.9$					
Project	24	980,113.02	40,838.04	2575.91	0.0000
Phase	1	672.77	672.77	42.44	0.0000
Residuals	7174	113,735.56	15.85		
Model fit with audience $R^2 = 0.9$					
Project	24	980,113.02	40,838.04	2570.60	0.0000
Audience	1	437.78	437.78	27.56	0.0000
Residuals	7174	113,970.54	15.89		
Model fit with tool $R^2 = 0.9$					
Project	24	980,113.02	40,838.04	2577.02	0.0000
Tool	2	737.74	368.87	23.28	0.0000
Residuals	7173	113,670.58	15.85		
Base model fit $R^2 = 0.9$					
Project	24	980,113.02	40,838.04	2561.12	0.0000
Residuals	7175	114,408.33	15.95		

4.7.1 What is the impact of tools after controlling for the number of mutants produced?

Figure 11b shows the entropy carried by the mutants, after controlling the number of mutants.

Table 32 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 (0.69–0.53) and τ_b (0.5–0.35).

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

We use *ANOVA* to answer this question, for which models are given in Eq. 1. The measure is the *entropy* of the mutant kill matrix, after controlling for the number of mutants.

The *ANOVA* (Table 33) suggests that *phase* and *audience* are statistically significant ($p < 0.05$) factors. When comparing R^2 of models containing both project and the factors investigated, the variation explained by models incorporating a given factor in addition to *project* is as follows: *phase* 62 %, *audience* 63 %, *tool* 73 %. Investigating the increase in explanatory power of complex models over a model containing only *project*, the increase in R^2 (explanatory power) was as follows: *phase* 2.9 %, *audience* 3.4 %, *tool* 14 %.

5 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.

Table 29 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
μ	2.90	4.71	4.90
σ	1.66	1.23	0.92

Table 30 Entropy correlation of different mutation tools

	R^2	τ_b	Difference μ	σ
Judy × Pit	0.28	0.25	−2	1.66
Judy × Major	−0.07	0.01	−1.81	2.13
Pit × Major	0.35	0.41	0.19	1.25

5.1 Comparison of difficulty of detection with mutation scores

We include two measures in the traditional comparison; the raw mutation score (Sect. 4.1) and the refined mutation score (Sect. 4.2). Considering the raw mutation scores that were reported by different tools, we find that Pit produces a much larger number of mutants than other tools (Fig. 3) and that the mean mutation scores across projects produced by different tools are quite close (Fig. 5). Surprisingly, the correlation between the mutation scores produced ranges from 0.37 (Judy × Pit) to 0.67 (Pit × Major). Similarly Kendall's τ_b

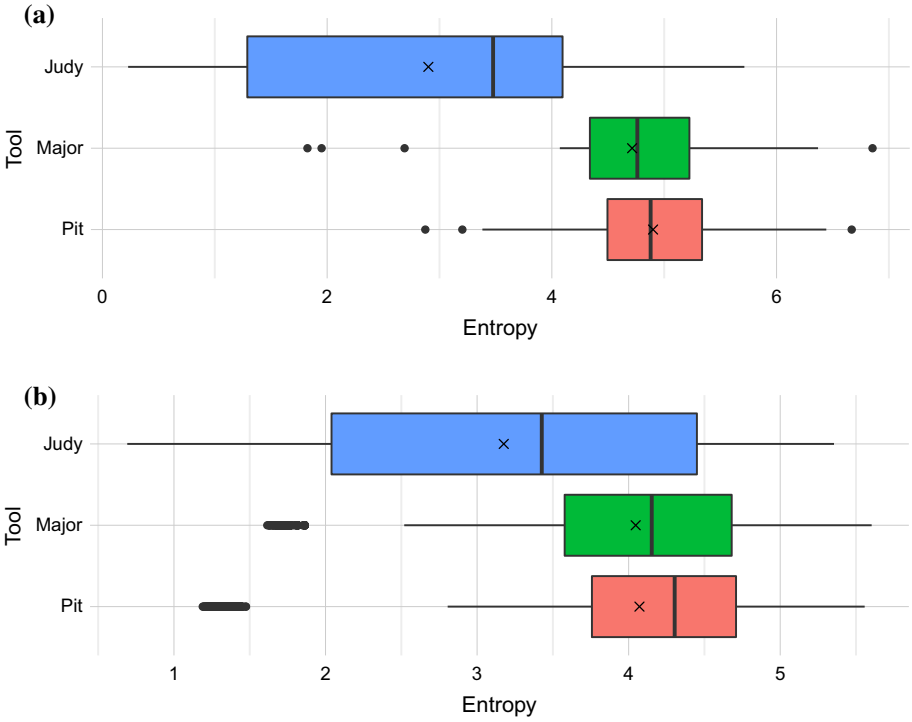

Fig. 11 Entropy of the mutant set produced—larger is better. **a** Full set of mutants. **b** 100 samples

Table 31 Model fit for entropy

	df	Sum sq	Mean sq	*F* value	$Pr(>F)$
Model fit with phase $R^2 = 0.0066$					
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 audience $R^2 = 0.08$					
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 tool $R^2 = 0.41$					
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		
Base model fit $R^2 = -0.062$					
Project	24	51.89	2.16	0.82	0.6971
Residuals	50	131.99	2.64		

Table 32 Entropy correlation of different mutation tools (100 samples)

	R^2	τ_b	Difference μ	σ
Judy × Pit	0.53	0.40	−1.03	1.17
Judy × Major	0.54	0.35	−0.92	1.18
Pit × Major	0.69	0.50	0.11	0.74

Table 33 Model fit for entropy (100 samples)

	df	Sum sq	Mean sq	*F* value	$Pr(>F)$
Model fit with phase $R^2 = 0.62$					
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 audience $R^2 = 0.63$					
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 tool $R^2 = 0.73$					
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		
Base model fit $R^2 = 0.59$					
Project	24	5963.01	248.46	434.55	0.0000
Residuals	7175	4102.34	0.57		

ranges from 0.27 (Judy × Pit) to 0.54 (Pit × Major). We also see a large standard deviation—up to 27 (Judy × Pit).

Our measures of correlation are markedly different from those obtained from mutation scores reported by Delahaye et al. (Table 2) 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. are incomplete (only four 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 and Radyk (2010) report that Judy and Jumble had a high correlation (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 suggest that tools rarely agree with each other on mutation score and often differ by a large amount. However, we find no tool consistently under or over reporting the mutation scores.

> This suggests two things: (1) the 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 the impact of equivalent mutants. For our second measure (Sect. 4.2), we removed all undetected mutants, and for the remaining mutants, we evaluated mutation scores produced by partial test suites containing a fixed fraction of the original test suite. Our results for *refined mutants* corroborate our findings with raw mutation scores. As Table 7 shows, the differences between Judy and Pit reduced in terms of both R^2 and τ_b, while measurements of other pairings remain very close. The spread, while slightly smaller than raw mutation scores, still remains high.

Note that mutation score is often the only measure from a set of mutants that is obtained by testers and researchers 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 when compared with Judy. This suggests that at least as far as mutation score is concerned, the impact of phase of generation and target audience is minimal.

5.2 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. We analyzed (Sect. 4.3) 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.

Our results from Sect. 4.3 suggest that the minimal set of mutants produced by different tools varies widely, from 0.26 (Judy × Major) to 0.96 (Pit × Major). Kendall's τ_b ranges from 0.35 (Judy × Major) to 0.85 (Pit × Major). We also see the maximum standard deviation of 209 (Judy × Major). Further, the mean difference between minimum mutants produced by tools ranges from −124 mutants (σ =175) (Judy × Pit) to 17 mutants (σ =63) (Pit × Major). Note that we compare *count* of minimal mutants rather than the fraction of reduction. The reason is that the *strength* of a set of mutants for the same program is determined by the size of the minimal mutant set irrespective of the size of the full mutant set. Unfortunately, this also means that one cannot normalize the size of minimal mutant set. The most interesting information here is the ordering between the tools, with Pit producing the strongest set of mutants, closely followed by Major.

Interestingly, the situation is different when the number of mutants is controlled for. We see a correlation between the minimum set from different tools from 0.49 (Judy × Pit) to 0.93 (Pit × Major). Similarly Kendall's τ_b ranges from 0.35 (Judy × Pit) to 0.73 (Pit × Major).

The mean difference between minimum mutants produced by tools ranges from −18 mutants (σ =18) (Judy × Pit) to 2.8 mutants (σ =7) (Pit × Major). 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 × Pit.

We find a similar result from our analysis of surface mutants (Sect. 4.4). 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 0.25 (Judy × Major) to 0.94 (Pit × Major). Kendall's τ_b ranges from 0.24 (Judy × Pit) to 0.76 (Pit × Major). We also see a maximum standard deviation of 233 (Judy × Major). Further, the mean

difference between minimum mutants produced by tools ranges from −173 (Judy × Pit) to 52 (Pit × Major).

We find a similar result as that of minimal mutants when the number of mutants is controlled for. We see a correlation between the minimum set from different tools from 0.47 (Judy × Pit) to 0.92 (Pit × Major). Similarly Kendall's τ_b ranges from 0.34 (Judy × Pit) to 0.72 (Pit × Major). As before, the mean difference between minimum mutants produced by tools ranges from −17 (Judy × Pit) to 2.6 (Pit × Major). 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.

Finally, note that Pit produces the strongest set of mutants, as measured by size of minimal and surface sets, irrespective of whether the size of mutant set is controlled for. However, we note that characteristics of projects are a significant factor and have a strong impact on the size of minimal or surface mutant sets.

> Using strength of mutants measured as by minimal and surface mutant sets, Pit and Major produce high strength mutants and only differ by a small amount. However, the characteristics of the project have a significant impact even after accounting for the number of mutants considered.

5.3 Comparison of mutant diversity with statistical measures

We had shown before (Gopinath et al. 2015) 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 (Sect. 4.5) shows that the correlation between tools for covariance ranges from 0.19 (Pit × Major) to 0.45 (Judy × Pit). The Kendall τ_b correlation ranges from 0.43 (Judy × Pit) to 0.43 (Pit × Major). These values are small, as expected. Remember that a larger covariance essentially means a smaller fraction of mutants can represent the full mutant set. Here, the number of mutants is 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, where the mean differences are 6.9 (Pit × Major), −6.1 (Judy × Pit) and 0.81 (Judy × Major).

Controlling the number of mutants should on the other hand lead to a higher correlation and smaller difference. Our results show that this is as expected, with mean difference ranging from −0.053 (Pit × Major) to 0.34 (Judy × Pit).

Our *mutual information* observations (Sect. 4.6) 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, with mean differences of −281 (Judy × Pit) 237 (Pit × Major) and −44 (Judy × Major).

As before, if the number of mutants is controlled, we have higher correlation ranging from 0.86 (Judy × Pit) to 0.78 (Pit × Major) and small mean differences ranging from 0.14 (Judy × Pit) to −0.56 (Pit × Major).

These measures 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.

5.4 Comparison of information carried with entropy

Similar to diversity measures, we expect little correlation when number of mutants is not controlled for. The correlation ranges from -0.066 (Judy $\times$ Major) to 0.35 (Pit $\times$ Major). We expect a larger set of mutants to have more entropy, and smaller set of mutants to have less entropy. The mean difference ranges from -2 (Judy $\times$ Pit) to 0.19 (Pit $\times$ Major).

Once the number of mutants is controlled for, we see a larger correlation ranging from 0.54 (Judy $\times$ Major) to 0.69 (Pit $\times$ Major), but little difference in mean, ranging from -2 (Judy $\times$ Pit) to 0.19 (Pit $\times$ Major).

For entropy, we again find Pit and Major better than Judy. Pit and Major have similar values, with Pit leading by a slight margin. However, note that once the size of the mutant set is controlled for, the characteristics of the project assume significance.

In terms of entropy, the leaders are Pit and Major, with Pit leading by a small margin. However, the characteristics of the project are a significant factor, even after accounting for the number of mutants considered.

5.5 Tool assessment

In general, we found that tool is a significant factor in almost all measures we examined. Further, the impact of tool remains significant even when the number of mutants is controlled for. This points to the fact that the particular tool used is an important factor in the quality of mutations produced. Next, provide assessments of each tool examined.

5.5.1 Judy

Judy is a tool oriented toward 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.

5.5.2 Major

Major is one of the few tools in use that is source based. It is also oriented toward the research community. In terms of raw mutation score, Major produces a medium mean

mutation 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 for 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 is 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.

5.5.3 Pit

Pit is a tool firmly focused on an industrial audience. It is bytecode based, and in terms of ease of use, it provides the best user experience. Pit produces a 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 for 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.

5.6 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.

5.7 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 a statistically significant factor. 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.

5.8 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.

5.9 Need for standardization

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 disagree 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 (Zhang et al. 2010, 2013; Gligoric et al. 2013; Gopinath et al. 2014). 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, we 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. (2014) show, 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*.

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 β_0, β_1 of linear regression $\mu_{\text{standard}} = \beta_0 + \beta_1 \mu_{\text{new}}$ be available. On the other hand, for tools and reduction techniques that target *comparison* of testing techniques, we require that the new mutation scores be in high Kendall's τ_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 τ_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 and measure its statistical divergence from the standard to place results using an older standard in context.

6 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 cannot 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 cannot 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 remains. 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 (Gopinath 2015).

7 Conclusion

We evaluated mutants produced by different mutation tools for Java 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.

We find that in terms of mutation score, while there is very little mean difference between different tools, there was often large standard deviation suggesting that tools seldom agree. Our findings indicate that in order to be useful as a measure of software quality, there is a need for standardization of mutation scores. We propose an approach for such a standardization.

In the meantime, we make a recommendation for languages such as Java where there are multiple tools, with no single tool consistently better than others. We suggest that researchers use a much larger set of projects, and multiple tools to evaluate mutation scores, since a large number of projects tend to reduce the ill effects of an incomplete set of mutants. For practicing testers, for whom a large number of projects is not appropriate, we recommend using multiple tools to compute the mutation score. A low mutation score reported by any tool used should be a cause for concern, and be manually reviewed and verified.

Appendix

Measures of correlation

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 τ_b is a measure of monotonicity between variables compared, measuring the difference between concordant and discordant pairs. Kendall's τ_b rank correlation coefficient is a nonparametric measure of association between two variables. It requires only that the dependent and independent variables (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 τ_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 τ_b, since the difference between concordant and discordant pairs would be small. On the other hand, say the mutation scores of one tool are 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 τ_b would be high. Hence both measures provide useful comparative information. Note that low τ_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.

While what can be considered high and low correlation is subjective, for the purpose of this paper, we consider $R^2 \leq 0.40$ to be low correlation, and $R^2 \geq 0.60$ to be high correlation.

Covariance

We showed (Gopinath et al. 2015) that for mutation analysis, the maximum number of mutants to be sampled for given tolerance has an upper bound provided by the binomial distribution, and the actual number is determined by the covariance.

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 $X_{1\ldots n}$. 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 $X_{1\ldots n}$ 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

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

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.

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 (1960)—*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:

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

Entropy

In information theory, *Shannon entropy* (Shannon 2001) 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 and their joint entropy. Formally,

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

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 information. 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:

$$I(p) = -p \times \log_2(p).$$

References

Acree, Jr. A. T. (1980). *On mutation*. Ph.D. dissertation, Georgia Institute of Technology, Atlanta, GA, USA.

Ammann, P. (2015a). *Problems with jester*. https://sites.google.com/site/mutationworkshop2015/program/MutationKeynote.

Ammann, P. (2015b). 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.

Ammann, P., Delamaro, M. E., & Offutt, J. (2014). Establishing theoretical minimal sets of mutants. In *International conference on software testing, verification and validation* (pp. 21–30). Washington, DC, USA: IEEE Computer Society.

Andrews, J. H., Briand, L. C., & Labiche, Y. (2005). Is mutation an appropriate tool for testing experiments? In *International conference on software engineering* (pp. 402–411). IEEE.

Andrews, J. H., Briand, L. C., Labiche, Y., & Namin, A. S. (2006). Using mutation analysis for assessing and comparing testing coverage criteria. *IEEE Transactions on Software Engineering*, *32*(8), 608–624.

Apache Software Foundation. (2016). *Apache commons*. http://commons.apache.org/.

Baldwin, D., & Sayward, F. (1979). *Heuristics for determining equivalence of program mutations*. DTIC Document: Tech. rep.

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

Budd, T. A. (1980). *Mutation analysis of program test data*. Ph.D. dissertation, Yale University, New Haven, CT, USA.

Budd, T. A., DeMillo, R. A., Lipton, R. J., & Sayward, F. G. (1980). 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* (pp. 220–233). ACM.

Budd, T. A., Lipton, R. J., DeMillo, R. A., & Sayward, F. G. (1979). *Mutation analysis*. Yale University, Department of Computer Science.

Budd, T. A., & Gopal, A. S. (1985). Program testing by specification mutation. *Computer Languages*, *10*(1), 63–73.

Cai, X., & Lyu, M. R. (2005). The effect of code coverage on fault detection under different testing profiles. In *ACM SIGSOFT software engineering notes* (Vol. 30, no. 4, pp. 1–7). ACM.

Chevalley, P., & Thévenod-Fosse, P. (2003). A mutation analysis tool for java programs. *International Journal on Software Tools for Technology Transfer*, *5*(1), 90–103.

Coles, H. (2016). *Pit mutation testing*. http://pitest.org/.

Coles, H. (2016a). *Mutation testing systems for java compared*. http://pitest.org/java_mutation_testing_systems/.

Coles, H. (2016b). *Pit mutators*. http://pitest.org/quickstart/mutators/.

Daran, M., & Thévenod-Fosse, P. (1996). Software error analysis: A real case study involving real faults and mutations. In *ACM SIGSOFT international symposium on software testing and analysis* (pp. 158–171). ACM.

Delahaye, M., & Du Bousquet, L. (2013). A comparison of mutation analysis tools for java. In *Quality software (QSIC), 2013 13th international conference on* (pp. 187–195). IEEE.

DeMillo, R. A., Guindi, D. S., McCracken, W., Offutt, A., & King, K. (1988). An extended overview of the mothra software testing environment. In *International conference on software testing, verification and validation workshops* (pp. 142–151). IEEE.

DeMillo, R. A., Lipton, R. J., & Sayward, F. G. (1978). Hints on test data selection: Help for the practicing programmer. *Computer*, *11*(4), 34–41.

Derezińska, A., & Hałas, K. (2014). Analysis of mutation operators for the python language. In *International conference on dependability and complex systems*, ser. Advances in Intelligent Systems and Computing (Vol. 286, pp. 155–164). Springer.

Do, H., & Rothermel, G. (2006). On the use of mutation faults in empirical assessments of test case prioritization techniques. *IEEE Transactions on Software Engineering*, *32*(9), 733–752.

Duraes, J., & Madeira, H. (2002). Emulation of software faults by educated mutations at machine-code level. *International Symposium on Software Reliability Engineering*, *2002*, 329–340.

GitHub Inc. (2016). *Software repository*. http://www.github.com.

Gligoric, M., Groce, A., Zhang, C., Sharma, R., Alipour, M. A., & Marinov, D. (2013). Comparing non-adequate test suites using coverage criteria. In *ACM SIGSOFT international symposium on software testing and analysis*. ACM.

Gligoric, M., Jagannath, V., & Marinov, D. (2010). Mutmut: Efficient exploration for mutation testing of multithreaded code. In *Software testing, verification and validation (ICST), 2010 third international conference on* (pp. 55–64). IEEE.

Gopinath, R. (2015). *Replication data for: Does choice of mutation tool matter?*. http://eecs.osuosl.org/rahul/sqj2015.

Gopinath, R., Alipour, A., Ahmed, I., Jensen, C., & Groce, A. (2015). Do mutation reduction strategies matter? Oregon State University, tech. rep., August 2015, under review for Software Quality Journal. http://hdl.handle.net/1957/56917.

Gopinath, R., Alipour, A., Ahmed, I., Jensen, C., & Groce, A. (2016). On the limits of mutation reduction strategies. In *Proceedings of the 38th international conference on software engineering*. ACM.

Gopinath, R., Alipour, A., Iftekhar, A., Jensen, C., & Groce, A. (2015). How hard does mutation analysis have to be, anyway? In *International symposium on software reliability engineering*. IEEE.

Gopinath, R., Jensen, C., & Groce, A. (2014). Code coverage for suite evaluation by developers. In *International conference on software engineering*. IEEE.

Gopinath, R., Jensen, C., & Groce, A. (2014). Mutations: How close are they to real faults? In *Software reliability engineering (ISSRE), 2014 IEEE 25th international symposium on* (pp. 189–200), November 2014.

Harder, M., Mellen, J., & Ernst, M.D. (2003). Improving test suites via operational abstraction. In *International conference on software engineering* (pp. 60–71). IEEE Computer Society.

Harder, M., Morse, B., & Ernst, M. D. (2001). *Specification coverage as a measure of test suite quality*. MIT Lab for Computer Science: tech. rep.

Irvine, S. A., Pavlinic, T., Trigg, L., Cleary, J. G., Inglis, S., & Utting, M. (2007). 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* (pp. 169–175). IEEE, 2007.

Jia, Y., & Harman, M. (2008). 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* (pp. 94–98). IEEE, 2008.

Jia, Y., & Harman, M. (2011). An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*, *37*(5), 649–678.

Just, R. (2014). 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 (pp. 433–436). New York, NY: ACM.

Just, R., Kapfhammer, G. M., & Schweiggert, F. (2012). Do redundant mutants affect the effectiveness and efficiency of mutation analysis? In *Software testing, verification and validation (ICST), 2012 IEEE fifth international conference on* (pp. 720–725). IEEE.

Just, R., Jalali, D., Inozemtseva, L., Ernst, M. D., Holmes, R., & Fraser, G. (2014). Are mutants a valid substitute for real faults in software testing? *ACM SIGSOFT symposium on the foundations of software engineering* (pp. 654–665). Hong Kong: ACM.

Kintis, M., Papadakis, M., & Malevris, N. (2010). Evaluating mutation testing alternatives: A collateral experiment. In *Asia Pacific software engineering conference (APSEC)* (pp. 300–309). IEEE.

Kurtz, B., Ammann, P., Delamaro, M. E., Offutt, J., & Deng, L. (2014). Mutant subsumption graphs. In *Software testing, verification and validation workshops (ICSTW), 2014 IEEE seventh international conference on* (pp. 176–185). IEEE, 2014.

Kusano, M., & Wang, C. (2013). Ccmutator: A mutation generator for concurrency constructs in multi-threaded c/c++ applications. In *Automated software engineering (ASE), 2013 IEEE/ACM 28th international conference on* (pp. 722–725). IEEE.

Langdon, W. B., Harman, M., & Jia, Y. (2010). Efficient multi-objective higher order mutation testing with genetic programming. *Journal of systems and Software*, *83*(12), 2416–2430.

Le, D., Alipour, M. A., Gopinath, R., & Groce, A. (2014). Mucheck: An extensible tool for mutation testing of haskell programs. In *Proceedings of the 2014 international symposium on software testing and analysis* (pp. 429–432). ACM.

Lipton, R. J. (1971). Fault diagnosis of computer programs. Carnegie Mellon University, Tech. rep.

Ma, Y.-S., Kwon, Y.-R., & Offutt, J. (2002). Inter-class mutation operators for java. In *International symposium on software reliability engineering* (pp. 352–363). IEEE.

Ma, Y.-S., Offutt, J., & Kwon, Y.-R. (2006). Mujava: A mutation system for java. In *Proceedings of the 28th international conference on software engineering*, ser. ICSE'06 (pp. 827–830). New York, NY: ACM, 2006.

Macedo, M. G. (2016). *Mutator*. http://ortask.com/mutator/.

Madeyski, L., & Radyk, N. (2010). Judy—A mutation testing tool for java. *IET software*, *4*(1), 32–42.

Ma, Y.-S., Offutt, J., & Kwon, Y. R. (2005). Mujava: An automated class mutation system. *Software Testing, Verification and Reliability*, *15*(2), 97–133.

Mathur, A. (1991). Performance, effectiveness, and reliability issues in software testing. In *Annual international computer software and applications conference, COMPSAC* (pp. 604–605), 1991.

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

Moore, I. (2001). Jester—a junit test tester. In *International conference on extreme programming* (pp. 84–87).

Namin, A. S., & Andrews, J. H. (2009). The influence of size and coverage on test suite effectiveness. In *ACM SIGSOFT international symposium on software testing and analysis* (pp. 57–68). ACM.

Namin, A. S., Andrews, J. H., & Murdoch, D. J. (2008). Sufficient mutation operators for measuring test effectiveness. In *International conference on software engineering* (pp. 351–360). ACM.

Nanavati, J., Wu, F., Harman, M., Jia, Y., & Krinke, J. (2015). Mutation testing of memory-related operators. In *Software testing, verification and validation workshops (ICSTW), 2015 IEEE eighth international conference on* (pp. 1–10). IEEE.

Nica, S., & Wotawa, F. (2012). Using constraints for equivalent mutant detection. In *Workshop on formal methods in the development of software, WS-FMDS* (pp. 1–8).

Nimmer, J. W., & Ernst, M. D. (2002). Automatic generation of program specifications. *ACM SIGSOFT Software Engineering Notes*, *27*(4), 229–239.

Offut, J. (2016a). *Problems with jester*. https://cs.gmu.edu/offutt/documents/personal/jester-anal.html.

Offut, J. (2016b). *Problems with parasoft insure++*. https://cs.gmu.edu/offutt/documents/handouts/parasoft-anal.html.

Offutt, J. (2016). *Insure++ critique*. https://cs.gmu.edu/offutt/documents/handouts/parasoft-anal.html.

Offutt, A. J., & Untch, R. H. (2000). Mutation, uniting the orthogonal. In *Mutation testing for the new century* (pp. 34–44). Springer, 2001.

Offutt, A. J., & Voas, J. M. (1996). *'Subsumption of condition coverage techniques by mutation testing*. Technical report ISSE-TR-96-01. Information and Software Systems Engineering. Tech. rep.: George Mason University.

Offutt, A. J., Rothermel, G., & Zapf, C. (1993). An experimental evaluation of selective mutation. In *International conference on software engineering* (pp. 100–107). IEEE Computer Society Press.

Offutt, A. J. (1989). The coupling effect: Fact or fiction? *ACM SIGSOFT Software Engineering Notes*, *14*(8), 131–140.

Offutt, A. J. (1992). Investigations of the software testing coupling effect. *ACM Transactions on Software Engineering and Methodology*, *1*(1), 5–20.

Offutt, A. J., & Craft, W. M. (1994). Using compiler optimization techniques to detect equivalent mutants. *Software Testing, Verification and Reliability*, *4*(3), 131–154.

Offutt, A. J., Lee, A., Rothermel, G., Untch, R. H., & Zapf, C. (1996). An experimental determination of sufficient mutant operators. *ACM Transactions on Software Engineering and Methodology*, *5*(2), 99–118.

Offutt, A. J., & Pan, J. (1997). Automatically detecting equivalent mutants and infeasible paths. *Software Testing, Verification and Reliability*, *7*(3), 165–192.

Okun, V. (2004). *Specification mutation for test generation and analysis*. Ph.D. dissertation, University of Maryland Baltimore County.

Papadakis, M., Jia, Y., Harman, M., & Traon, Y. L. (2015). Trivial compiler equivalence: A large scale empirical study of a simple, fast and effective equivalent mutant detection technique. In *International conference on software engineering*.

Parasoft. (2014). *Insure++*. www.parasoft.com/products/insure/papers/tech_mut.htm.

Parasoft. (2015). *Insure++ mutation analysis*. http://www.parasoft.com/jsp/products/article.jsp?articleId=291&product=Insure.

Schuler, D., & Zeller, A. (2009). Javalanche: Efficient mutation testing for java. In *ACM SIGSOFT symposium on the foundations of software engineering* (pp. 297–298). August, 2009.

Schuler, D., Dallmeier, V., & Zeller, A. (2009). Efficient mutation testing by checking invariant violations. In *ACM SIGSOFT international symposium on software testing and analysis* (pp. 69–80). ACM.

Schuler, D., & Zeller, A. (2013). Covering and uncovering equivalent mutants. *Software Testing, Verification and Reliability*, *23*(5), 353–374.

Shannon, C. E. (2001). A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, *5*(1), 3–55.

Singh, P. K., Sangwan, O. P., & Sharma, A. (2014). 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)*, *6*(11), 1.

Smith, B. H., & Williams, L. (2007). An empirical evaluation of the mujava mutation operators. In *Testing: academic and industrial conference practice and research techniques-MUTATION, 2007. TAICPART-MUTATION 2007* (pp. 193–202). IEEE.

Sridharan, M., & Namin, A. S. (2010). Prioritizing mutation operators based on importance sampling. In *International symposium on software reliability engineering* (pp. 378–387). IEEE.

Untch, R. H. (2009). On reduced neighborhood mutation analysis using a single mutagenic operator. In *Annual southeast regional conference*, ser. ACM-SE 47 (pp. 71:1–71:4). New York, NY: ACM.

Usaola, M. P., & Mateo, P. R. (2012). 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 (pp. 646–649). Washington, DC: IEEE Computer Society.

Wah, K. S. H. T. (2000). A theoretical study of fault coupling. *Software Testing, Verification and Reliability*, *10*(1), 3–45.

Wah, K. S. H. T. (2003). An analysis of the coupling effect i: Single test data. *Science of Computer Programming*, *48*(2), 119–161.

Watanabe, S. (1960). Information theoretical analysis of multivariate correlation. *IBM Journal of Research and Development*, *4*(1), 66–82.

Wong, W. E. (1993). *On mutation and data flow*. Ph.D. dissertation, Purdue University, West Lafayette, IN, USA, uMI Order No. GAX94-20921.

Wong, W., & Mathur, A. P. (1995). Reducing the cost of mutation testing: An empirical study. *Journal of Systems and Software*, *31*(3), 185–196.

Yao, X., Harman, M., & Jia, Y. (2014). A study of equivalent and stubborn mutation operators using human analysis of equivalence. In *International conference on software engineering* (pp. 919–930).

Zhang, L., Gligoric, M., Marinov, D., & Khurshid, S. (2013). Operator-based and random mutant selection: Better together. In *IEEE/ACM automated software engineering*. ACM.

Zhang, L., Hou, S.-S., Hu, J.-J., Xie, T., & Mei, H. (2010). Is operator-based mutant selection superior to random mutant selection? In *International conference on software engineering* (pp. 435–444). New York, NY: ACM.

Zhang, J., Zhu, M., Hao, D., & Zhang, L. (2014). An empirical study on the scalability of selective mutation testing. In *International symposium on software reliability engineering*. ACM.

Zhou, C., & Frankl, P. (2009). Mutation testing for java database applications. In *Software testing verification and validation, ICST'09. International conference on* (pp. 396–405). IEEE, 2009.

Rahul Gopinath is a Ph.D. candidate in the School of Electrical Engineering and Computer Science (EECS) at Oregon State University (OSU). He received his B.Tech from Kerala University, India, and MCS from Illinois Institute of Technology in 2010. His primary area of research is mutation analysis of programs, and especially how to make mutation analysis a workable technique for real-world developers and testers.

Iftekhar Ahmed is working toward a Ph.D. in Computer Science at the Oregon State University (OSU). He got his B.Sc. in Computer Science and Engineering from Shahjalal University of Science and Technology, Bangladesh. His research interest is in software engineering, with an emphasis on testing, developing and understanding critical software systems and ways to combine testing, static analysis and machine learning approaches for coming up with better tools and techniques. His goal is to develop a combined technique of mutant prioritization and automatic mutation analysis, which application developers can use with minimum effort and can integrate in their development tool-chain.

Mohammad Amin Alipour is a Ph.D. student of computer science at Oregon State University. He has B.S. in computer engineering and M.S. in computer science from Petroleum University of Technology, Iran, and Michigan Technological University. His research area is software testing and analysis. He has published papers in journals such as ACM Transactions on Software Engineering and Methodology (TOSEM), Journal of Software Testing, Verification and Reliability, and conferences such as ISSTA, ISSRE, and ICST. He has been reviewer for ISSTA, ASE, SPLASH and SPIN conferences.

Carlos Jensen is an associate professor in the School of EECS at Oregon State University. He is the Director of the Center for Applies Systems and Software (CASS), which includes the OSU Open Source Lab, home to over 160 medium and large open source projects like the Linux Foundation, Apache Foundation, and Drupal. His research is in the intersection of usability and software engineering; he and his students study how open source developers to produce great software, where they fail to, and what tools and techniques we can develop to help more projects be successful. Carlos Jensen received his Ph.D. in Computer Science from the Georgia Institute of Technology (2005).

Alex Groce received his Ph.D. in computer science from Carnegie Mellon University in 2005, and B.S. degrees in computer science and multi-disciplinary studies (with a focus on English literature) from North Carolina State University in 1999. Before joining the Oregon State University faculty in 2009, he was a core member of the Laboratory for Reliable Software at NASA's Jet Propulsion Laboratory, and taught lasses on software testing at the California Institute of Technology. His activities at JPL included a role as lead developer and designer for test automation for the Mars Science Laboratory mission's internal flight software test team, and lead roles in testing file systems for space missions. His research interests are in software engineering, particularly testing, model checking, code analysis, debugging and error explanation. He focuses on software engineering from an "investigative" viewpoint, with an emphasis on the execution traces that programs produce—software engineering as the art and science of building programs with a desired set of executions.