Carleton University technical report SCE-04-15

Is Mutation an Appropriate Tool for Testing Experiments?

J. H. Andrews
Computer Science Department
University of Western Ontario
London, Canada
andrews@csd.uwo.ca

L. C. Briand, Y. Labiche
Software Quality Engineering Laboratory
Systems and Computer Engineering Department
Carleton University, 1125 Colonel By Drive
Ottawa, ON K1S5B6, Canada
{briand, labiche}@sce.carleton.ca

Abstract

The empirical assessment of test techniques plays an important role in software testing research. One common practice is to instrument faults, either manually or by using mutation operators. The latter allows the systematic, repeatable seeding of large numbers of faults; however, we do not know whether empirical results obtained this way lead to valid, representative conclusions. This paper investigates this important question based on a number of programs with comprehensive pools of test cases and known faults. It is concluded that, based on the data available thus far, the use of mutation operators is yielding trustworthy results.

1. Introduction

Experimentation is an essential part of research in software testing. Typically, experiments are used to determine which of two or more methods is superior for performing some testing-related activity. For instance, one may be interested in comparing the fault detection effectiveness of several testing criteria used to derive test cases, and one resorts to experiments to that aim. Testing experiments often require a set of subject programs with known faults. These subject programs should be big enough to be realistic, but not so big as to make experimentation infeasible. As for the faults, the ability of a technique to deal with the given faults should be an accurate predictor of the performance of the technique on real programs.

One problem in the design of testing experiments is that real programs of appropriate size with real faults are hard to find, and hard to prepare appropriately (for instance, by preparing correct and faulty versions). Even when actual programs with actual faults are available, often these faults are not numerous enough to allow the experimental results to achieve statistical significance. Many researchers therefore have taken the approach of introducing faults into correct programs to produce faulty versions.

These faults can be introduced by hand (the experimenter can for instance ask experienced engineers to do that), or by automatically generating variants of the code. Generally we view an automatically-generated variant as the result of applying an operator to the code. The operators used in this way are called mutation operators, the resulting faulty versions are called mutants, and the general technique is called mutation or mutant generation.

The main potential advantage of mutant generation is that the mutation operators can be described precisely and thus provide a well-defined, fault-seeding process. This helps researchers replicate others' experiments, a necessary condition for good experimental science. While hand-introduced faults can be argued to be more realistic, ultimately it is a subjective judgment whether a given fault is realistic or not. Another important advantage of mutant generation is that a potentially large number of mutants can be generated, increasing the statistical significance of results obtained.

However, an important question remains. How do we know whether the ability to detect (or "kill") mutants is an accurate predictor of actual performance; that is, what is the external validity of mutants in assessing the fault detection effectiveness of testing techniques? An answer to this question would have important consequences on how we perform testing experiments and therefore on the validity of experimental results.

The main contribution of this paper is to compare the fault detection ability of test suites on hand-seeded, automatically-generated, and real-world faults. Our subject programs are a widely-used set of programs with hand-seeded faults, and a widely-used program with real faults. We generated mutants from the subject programs using a set of standard mutation operators from the literature on mutation testing. Our results suggest that the generated mutants were similar to the real faults but different from the hand-seeded faults, and that the hand-seeded faults are harder to detect than the real faults.

2. Related Work

The idea of using mutants to measure test suite adequacy was originally proposed by DeMillo et al. [4] and Hamlet [9], and explored extensively by Offutt and others [17]. Offutt [14] showed empirical support for one of the basic premises of mutation testing, that a test data set that detects simple faults (such as those introduced by mutation) will detect complex faults.

Experiments using faulty variants of programs have been carried out by Frankl and Weiss [7], Thévenod-Fosse et al. [20], and Hutchins et al. [11], and since then by many researchers. Frankl and Weiss used nine Pascal programs with one existing fault each, whereas Hutchins et al. hand-seeded 130 faults over the seven programs used. Thévenod-Fosse et al. automatically seeded faults in four small C programs using mutation operators. Generally, these experiments follow the pattern of generating a large "test pool" of test cases, running all the faulty versions on all the test cases in the test pool, observing which test cases detected which faults, and using that data to deduce the fault detection abilities of given test suites drawn from the pool (e.g., test suites that satisfy specific coverage criteria).

Although mutant generation was originally proposed as part of a testing strategy, note that Thévenod-Fosse et al. used it instead as a method for generating faulty versions for experiments. Other researchers who have done this include Kim et al. [12], Memon et al. [13], Andrews and Zhang [1] and Briand and Labiche [2].

Finally, Chen et al. [3] used both hand-seeded faults and generated mutants in their experiments. They point out that the hand-seeded faults are only a subset of the possible faults, and raise the issue of whether the faults were representative, but as this was not the focus of their research they do not explore it further.

To conclude, as far as we are aware there has been no empirical study that has directly assessed the use of mutants or hand-seeded faults by comparing them with results obtained on real faults.

3. Experiment Material and Procedure

In this section, we describe the experiment we performed, the subject programs and the artifacts (faulty versions and test cases) with which they are associated, and the mutation operators that we used to produce mutants.

In the remainder of this article, we use the terms hand-seeded faults (e.g., produced by an experienced engineer) and real faults (e.g., discovered during software development), or simply faults when there is no ambiguity, as opposed to faults generated from mutation operators that are referred to as mutants.

A summary of the properties of the subject programs is found in Table 1. "NLOC" here is the net lines of code, i.e., lines of code which are nonblank after comments have been removed. "# conditionals" is the number of C conditional constructs (if, while, case, etc.) and binary logical operators (&& and ||), included as a rough indicator of the complexity of the code.

3.1. Definition of the Experiment

To achieve the objectives stated in the Introduction, we analyze the detection rates of test suites. We use eight subject programs (Table 1) for which faults and large pools of test cases are available (Section 3.2). Test suites are then formed by random sampling of the large test pools for each subject program. We then create mutant versions of those programs and execute those test suites on available faulty versions and mutants, recording faults that are killed, in order to compute the fault detection ratios of the test suites. Those rates are compared between mutants and faults in order to determine whether they are similar. Because each test suite may yield a different detection ratio, we obtain two distributions for each subject program, for faults and mutants respectively. The central tendencies of these distributions are compared through statistical inference testing. Differences across fault and mutant distributions have to be explained based on the data available and possible differences of results across subject programs also need to be investigated.

3.2. Subject Programs

We used a set of eight well-known subject programs written in C (Table 1). The first is the program usually referred to as Space, developed at the European Space Agency and first used by Vokolos and Frankl [21] to evaluate a regression testing technique. The last seven are the so-called Siemens suite of programs with hand-seeded faults, first used by Hutchins et al. [11] to compare control flow-based and dataflow-based coverage criteria. The artifacts (including faulty versions and test cases) of all eight programs have subsequently been modified and extended by other researchers, notably Rothermel and

Table 1. Description of Subject Programs

	Subject Programs							
Criterion	Space	Printtokens	Printtokens2	Replace	Schedule	Schedule2	Tcas	Totinfo
NLOC	5905	343	355	513	296	263	137	281
# conditionals	635	94	81	99	37	51	25	46
Test Pool Size (# test cases)	13585	4130	4115	5542	2650	2710	1608	1052
Number of faults (Versions)	38	7	10	32	9	10	41	23
Number of compiled mutants	11379	582	375	666	253	299	291	516

Harrold [19] and Graves et al. [8]. We chose these programs because of the maturity of the associated artifacts, and because of their historical significance: Do et al. [6] report that over the last ten years, 17 high-profile experimental software engineering papers have used the Siemens suite and/or Space.

Space is associated with 38 faulty versions (real faults), 33 from the original Vokolos and Frankl research and 5 detected later by other researchers. Each faulty version corresponds to a fault that was corrected “during testing and the operative use of the program” [21]. The "gold" version with all faults corrected is taken as producing correct output. In contrast, the faulty versions of the Siemens suite programs were produced by hand, "by ten different people, mostly without knowledge of each other's work; their goal was to produce faults that were as realistic as possible" [11].

The test pool (set of all test cases) for each subject program was constructed by the original researchers and by subsequent researchers following various functional (black-box) testing techniques and structural test coverage criteria. Harder et al. [10] give a comprehensive history of the construction of these test pools.

3.3. Mutation Operators

To generate mutants of the subject programs, we used a mutant generation program first used by Andrews and Zhang [1] to generate mutants for code written in C. To generate mutants from a source file, each of four classes of "mutation operators" was applied to every line of code. Every valid application of a mutation operator to a line of code resulted in another mutant being generated. The four classes of mutation operators were:

- Replace an integer constant C by 0, 1, -1, ((C)+1), or ((C)-1).
- Replace an arithmetic, relational, logical, bitwise logical, increment/decrement, or arithmetic-assignment operator by another operator from the same class.
- Negate the decision in an `if` or `while` statement.
- Delete a statement.

The first three operator classes were taken from Offutt et al.'s research [15] on identifying a set of "sufficient" mutation operators, i.e., a set S of operators such that test suites that kill mutants formed by S tend to kill mutants formed by a very broad class of operators. They were adapted so that they would work on C programs rather than the Fortran of the original research. The fourth operator, which also appears in [15], was added because the subject programs of the original study [1], during which the mutant generation program was first used, contained a large number of pointer-manipulation and field-assignment statements that would not be vulnerable to any of the sufficient mutation operators.

About 8.4% of the resulting mutants did not compile. The numbers of mutants of each subject program that compiled appear in Table 1. For the Space program, there were so many mutants generated that it was infeasible to run them all on the test suite. We therefore ran the test suite on every 10th mutant generated. Because the number of mutants generated per line did not follow any pattern that would interact with the selection of every 10th mutant, this amounted to a random selection of 10% of the mutants, taken from a uniform distribution over all the possible mutants.

3.4. Analysis Procedure

We provide in this section a brief description and justification of the analysis procedure that we used. More details will be presented in the next section as we report on the results.

The first step was to generate and compile the mutants, and to run all mutants and faulty versions on the entire test pool. All mutants that were not killed by any test case were then deemed to be equivalent. Though this may not be the case for every mutant, it was thought to be a good enough approximation and it is in any case the only option when dealing with large numbers of mutants, since automatically identifying equivalent mutants is an undecidable problem [19].

For each subject program, 5000 test suites were randomly formed by randomly sampling (without duplication) the available test pool. We needed to generate a large number of test suites to obtain sample distributions of fault/mutant detection rates that would approximate well the underlying theoretical distributions. Large samples also increase the power of statistical tests and facilitate their usage as further described below. One decision to be made was related to the size of the test suites. We wanted this to be a constant in our analysis so as not to blur the trends we were analyzing. We performed our analysis with different sizes to determine how it would affect the results, ranging from 10 to 100 test cases. We then determined which test case in the pool detected which mutant and fault. Next we computed the fault detection ratios of all test suites, plotted the detection ratio distributions of mutants and faults for each subject program, and then compared their central tendencies.

To summarize, for each test suite S, the procedure yielded two pieces of summary data: Dm(S), the number of mutants detected by S, and Df(S), the number of original faults detected by S. Given the number Nm of non-equivalent mutants and Nf of non-equivalent original faults of the subject program, we calculated the *mutation adequacy ratio* Am(S) of each test suite S as Dm(S)/Nm, and the *fault adequacy ratio* Af(S) as Df(S)/Nf.

To compare the means of the *Am* and *Af* distributions, we employed a standard statistical test: the Matched Pairs *t*-test [18], for reasons that will be further detailed in Section 4[1]. If the difference in means between the fault and mutant distributions is statistically and practically[2] significant, then the next question that arises is why? Moreover, if the results are not consistent across programs we also need to identify what the most plausible explanations are. There are many possible reasons including differences in characteristics of the subject programs, the test suites, and the way faults were seeded. The only thing that is common to all subject programs is the way we generated mutants.

To explain our results, and for reasons further explained in Section 4, we have also looked at the percentage of test cases that killed each fault and mutant. For each mutant M and faulty version F we calculated K(M) (resp. K(F)), the number of test cases that killed it. We then calculated, for each mutant M, the ease E(M) of killing the mutant as the ratio E(M) = K(M)/T, where T is the total number of test cases for the program. (We calculated E(F) for each faulty version similarly.) For each of them we then obtained a distribution where each observation corresponds to a mutant or fault, and we could compare the means of those distributions across subject programs.

In terms of analysis, we first performed an Analysis of Variance (ANOVA) to assess the overall statistical significance of differences among program means. If significant, we then proceeded to compare the distribution means for each program pair using a *t*-test for independent samples. To do so we resort to the Bonferroni procedure to ensure a low risk of type I error when performing a large number of comparisons (28 in our case) [18]. Because we deal here with smaller samples (sample sizes are determined by the number of mutants and faults, as opposed to test suites above), we need to check non-significant results with an equivalent non-parametric test, the Mann-Whitney test [17, 18]. Indeed, on small samples, violations of the *t*-test assumptions can lead to increases in type II errors, that is to wrongly deduce that a difference in means is not significant.

In all our statistical tests we used as a level of significance $\alpha = 0.05$, thus implying a 5% chance of committing a type I error or, in other words, a 5% risk to identify a difference as statistically significant when it is not.

3.5. Threats to validity

The programs, test pools, and faults we are using were not selected based on any particular criteria, except that they were well-prepared and historically important. However, we have no guarantee that test pools have the same detection power and coverage, though the literature reporting on them seems to suggest they were generated in a similar way and are rather comprehensive since they cover some of the main structural and black-box coverage criteria, as reported in Section 3.2.

We have actual faults for Space, but for the others we have simply “realistic” faults hand-seeded by experienced engineers. We also cannot expect the programs to be of similar complexity, regardless of how we define and measure it. They actually widely vary in size (Table 1) and control flow complexity. It is also expected that the results of our study would vary depending on the mutation operators selected; the ones we used, as discussed in Section 3.3, were selected based on the literature available so as to be a minimal but sufficient set.

Our analysis and interpretations will have to account for all the abovementioned factors so as to ensure that all plausible explanations are considered.

4. Analysis Results

Following the procedure described in Section 3.4, we first compare the adequacy ratios of test suites in terms of mutant and fault (Section 4.1). In Section 4.2, we identify possible phenomena which could explain the trends observed in Section 4.1. To investigate the most plausible explanations, we then compare the percentages of test cases that kill faults and mutants across programs, thus looking at their detectability (Sections 4.3 and 4.4). We end the section with a discussion of the implications of our results (Section 4.5).

4.1. Comparing Detection Distributions of Mutant and Faults

The first step of our analysis is to compare, for each subject program, the distributions of mutant and fault adequacy ratios *Am* and *Af*. The objective is to determine whether there are statistically and practically significant differences among them. We provide in Table 2 descriptive statistics of those distributions for each subject program (with a two-decimal precision). Note that those are the results we obtain when the selected test suite size

---

[1] When dealing with large data sets (as a rule of thumb, more than 100 observations), this test is very robust to departures from the normality assumption underlying the matched-pairs *t*-test (differences between pairs are assumed to be normally distributed), even in situations showing extreme distribution skewness. This is important as such violations of assumptions are common in real-world datasets. Note also that in our case we will not have extreme outliers among our observations as we will be dealing with ratios between 0 and 1 and that we will have 5000 observations at our disposal. However, to be on the safe side, another equivalent, non-parametric test was used (Wilcoxon Matched-Pairs Signed Ranks Test) to double-check the results. In general, when using large samples, the matched pairs *t*-test is also particularly well suited since it takes into account the magnitude of the differences and is therefore more powerful than the equivalent non-parametric tests.

[2] When dealing with large samples, even small differences in means can lead to statistically significant results. However, it is also important to determine whether a statistically significant result would have any practical implication, that is if the difference in means is large enough to be worth considering. This is what is typically referred to as “practical” significance, sometimes also denoted as “clinical” significance.

Table 2. Descriptive Statistics – Adequacy Ratios for Mutants and Faults (test suite size 100)

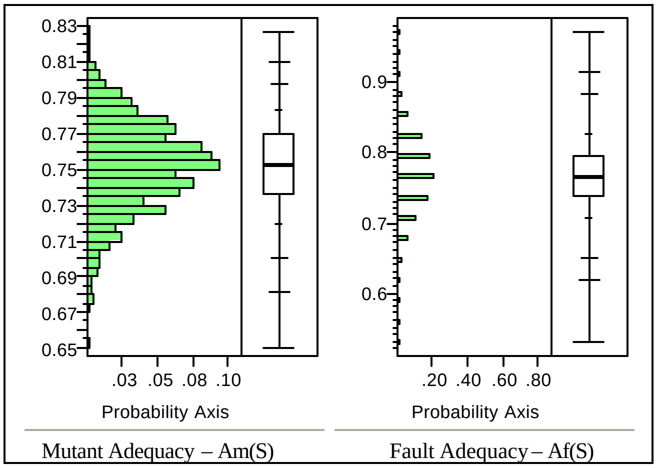
	Subject Programs							
Statistic	Space	Replace	Printtokens	Printtokens2	Schedule	Schedule2	Tcas	Totinfo
	Mutants – Am(S)							
Median	0.75	0.93	0.98	0.99	0.96	0.96	0.91	0.99
Mean	0.75	0.93	0.97	0.99	0.96	0.96	0.90	0.99
90%	0.78	0.95	0.99	0.99	0.98	0.97	0.94	0.99
75%	0.77	0.94	0.98	0.99	0.97	0.97	0.93	0.99
25%	0.74	0.93	0.97	0.98	0.94	0.95	0.89	0.98
10%	0.72	0.92	0.96	0.98	0.93	0.95	0.86	0.98
Min	0.65	0.88	0.94	0.93	0.91	0.91	0.77	0.94
Max	0.82	0.98	1	1	0.99	0.99	0.97	1
	Faults – Af(S)							
Median	0.76	0.68	0.57	0.90	0.67	0.67	0.76	0.65
Mean	0.77	0.67	0.63	0.93	0.66	0.63	0.79	0.89
90%	0.82	0.77	0.86	1	0.78	0.78	1	0.96
75%	0.79	0.74	0.71	1	0.78	0.78	0.95	0.91
25%	0.74	0.61	0.57	0.90	0.56	0.56	0.68	0.87
10%	0.71	0.55	0.29	0.80	0.44	0.44	0.61	0.83
Min	0.53	0.35	0.14	0.60	0.33	0	0.34	0.65
Max	0.97	0.93	1	1	1	1	1	1

is 100. Similar results were obtained with other test suite sizes (10, 20, 50) and are therefore not reported here.

From Table 2, we can easily observe that, in general, mutants tend to be easier to detect than faults. The only exception to this trend is Space where no practically significant difference can be observed between mutant and fault distributions (e.g., their medians are 0.75 and 0.76, respectively). Though due to space constraints not all distributions can be shown here, Figures 1a and 1b display the distribution histograms for the Space and Replace programs along with standard quantile box plot to visualize the distribution's main descriptive statistics. The box plot shows the median, average, various quantiles (1%, 10%, 25%, 75%, 90%, 99%), and the minimum and maximum.

We now have to check whether differences are statistically significant. To do that, we perform a Matched Pairs *t*-test [5, 18]. Recall that in these distributions, we compare mutant (Am) and fault (Af) adequacy ratios for 5000 test suites. In each distribution, observations are likely to correlate as they are paired, i.e., one observation in each of the *Am* and *Af* distributions corresponds to the same test suite. This is why such a matched pairs test is needed as opposed to an independent samples test.

Figure 1a. Subject Program Space: Distributions of Adequacy Ratios: test suite size = 100



Mutant Adequacy – Am(S)

Fault Adequacy – Af(S)

Figure 1b. Subject Program Replace: Distributions of Adequacy Ratios: test suite size = 100

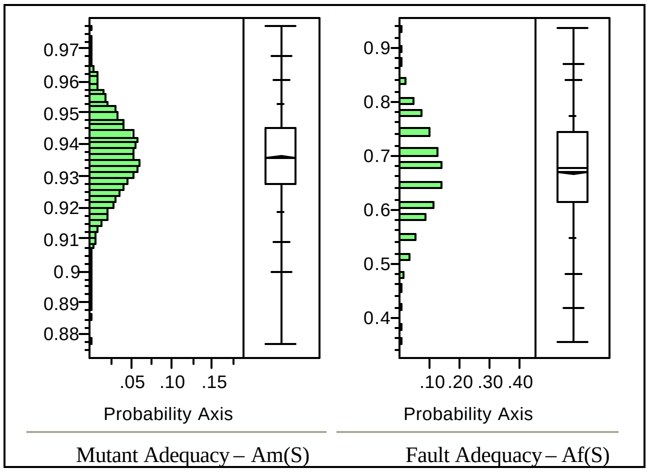


Table 3 shows, for each subject program, the results of the *t*-test, that is the average difference in fault/mutant adequacy ratios among pairs, the *t*-ratios, and *p*-values (this is a two-way test as we could not beforehand determine the direction of differences). Results show that all differences are statistically significant as all *p*-values are close or equal to 0. This is even the case of Space though with a mean difference of 1%, this is not practically significant as no test technique assessment would lead to different conclusions based on such small differences. For the remaining subject programs, average differences range from 6% to 34%, with an average of 22%. As opposed to Space, this is practically significant as, based on such ranges, if one has used mutants to assess a test technique, it will likely look more effective at detecting faults than if one has used the seeded faults.

Table 3. Matched Pairs *t*-test Results – test suite size = 100

	Matched Pairs Results		
Subject Programs	Mean Af(S) – Am(S)	*t*-ratio	*p*-value
Space	0.014	16.87	< 0.0001
Replace	-0.266	-233.96	0.0000
Printtokens	-0.344	-158.2	0.0000
Printtokens2	-0.061	-59.39	0.0000
Schedule	-0.298	-161.33	0.0000
Schedule2	-0.327	-152.19	0.0000
Tcas	-0.1128	-57.56	0.0000
Totinfo	-0.1037	-145.78	0.0000

However, it is important to remember that only the Space faults are real faults detected during testing and operation. So one plausible reason for the differences observed between Space and the other programs would be that the faults seeded in the latter are more difficult to detect than the real faults of Space. If this is deemed the most plausible reason, then the results of testing on the Space faults are therefore more credible and conclusions should be drawn based on this program's results only.

4.2. Alternative Explanations

Besides the relative detectability of real and seeded faults, several other factors could explain the data presented in the previous section: The test pool, the program inner characteristics (e.g., size), the test size suite compared to the size of the test pool and program, and the mutation process. It is important to consider each of them in turn to assess whether our first explanation is indeed the most plausible.

Test pools can contain test cases that have, on average, different detection abilities. The literature about these programs [10], however, suggests the pools were formed following a similar process and identical criteria. Furthermore, to explain our results this would mean that the average test case detection ability would vary depending on whether one consider mutants or faults as Space, when compared to the Siemens programs, has an intermediate-ranking adequacy ratio for faults and a low one for mutants.

Another possibility is that the difference between *Am* and *Af* increases as the test suite size increases compared to the size of the test pool or the size of the program. (Space is much bigger and has a larger test pool size than the other programs.) If this were the case, then we should see a smaller Am-Af for the Siemens programs as test suite sizes decrease. However, the results turned out to be similar for smaller test suite sizes of 10, 20, and 50: mean(Am-Af) remained similar for Space and tended to even increase for the other programs as smaller test suite sizes we used. The graphs in the Appendix compare *Am* and *Af* for the four test suite sizes, for Space and for Replace; the graphs for the other Siemens programs were similar.

The mutation process could somehow be biased so that mutants are harder to kill on Space than on the Siemens programs. However, we followed the exact same mutation process for Space and there is no clear reason why the result should be different. Therefore, the fact that Space is much larger than the Siemens programs is a more likely explanation for the lower mutation adequacy ratio as larger programs are known to be more difficult to test. This explanation is however neither confirmable nor falsifiable with the small number of subject programs we have, since all the Siemens programs are small and of similar size and we do not have observations in a continuous range of sizes up to that of Space.

A higher detectability of faults for Space than for the Siemens programs is the factor we already discussed above and the last to consider. To investigate further whether this is the most plausible explanation, we look next at the detectability of faults and mutants in terms of their likelihood of being detected by test cases randomly drawn from the test pools across programs.

Assuming that the Siemens hand-seeded faults are harder to detect than the real Space faults—and that this is not due to variations in the detection ability of test cases across test pools—we should observe the following trends: (1) Space faults should be easier for any individual test case to detect than the Siemens faults, (2) Space mutants should not be easier for any individual test case to kill than the Siemens program mutants. We explore these predictions in turn in the next two sections.

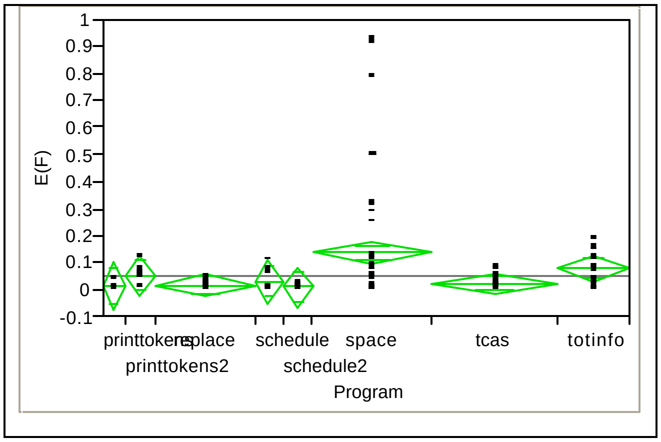
4.3. Comparing the Detectability of Faults across programs

Figure 2 shows the distributions of the ratios E(F) of test cases that detected each fault, for each subject program. The Y-Axis is the computed ratio for each mutant whereas the X-Axis is simply the subject program. The horizontal extent for each subject program on the X-Axis is proportional to its relative mutant sample size. We can see, for example, that the sample size is larger for Space and Replace, and rather small for Schedule and Schedule 2 (Table 1). These unequal sample sizes will need to be accounted for in our analysis. Moreover, the figure shows the observations for each program, the overall average (horizontal line across programs), and each program specific average (line across each diamond) as well as their 95% confidence interval for the average (the vertical span of each diamond).

Figure 2 clearly shows that Space ratios tend to be *higher* than for other programs, as we expected. A one-way Analysis of Variance (ANOVA) [5, 18] shows that, overall, there are statistically significant differences (F-test, p-value < 0.0001) across program means. The E(F) means of the Siemens programs and of Space faults are 0.035 and 0.14, respectively, thus clearly showing a significant difference.

Further statistical testing comparing each Siemens program with Space (using the Bonferroni procedure[3] for multiple comparisons of means using t-tests [18]) shows that Space is only significantly different from Tcas and Replace (at $\alpha = 0.05$), though differences are graphically visible between Space and all the other programs[4]. A likely cause for the lack of statistical significance with the other Siemens programs is that we deal with small samples, as the number of faults ranges from seven (Printtokens) to 23 (Totinfo). It may also be due to the Bonferroni procedure which tends to be conservative [5]. This increases the probability of a Type II error, and makes it likely that legitimately significant results will fail to be detected. In other words, it guarantees that the type I error rate is *at most* α (0.05 in our case) but it may be much less in reality.

Figure 2. Distribution of Ease of Detection of Faults – E(F)

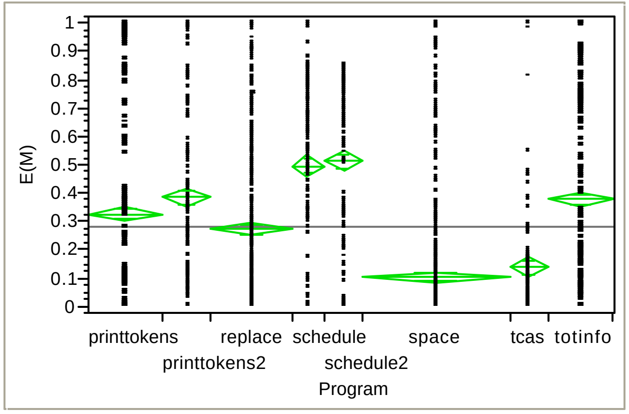


4.4. Comparing the Detectability of Mutants Across Programs

Figure 3 shows the distributions of E(M), the ratio of test cases that killed each mutant, for each subject program. It has the same structure as Figure 2, except for the fact that it is looking at mutants rather than faults.

As for faults in the previous section, ANOVA shows that the differences across programs are overall statistically significant (F-test, p-value < 0.0001).

Figure 3. Distribution of Ease of Detection of Mutants – E(M)



When looking more closely at each program pair using again the Bonferroni procedure and t-tests, we observe that the mean for Space is significantly lower than that of all programs but Tcas (at $\alpha = 0.05$), confirming what is visible in Figure 3. We can therefore conclude that, not only are mutants not easier to kill, but the individual test cases show more difficulty to detect mutants in Space and, to a lesser extent, Tcas than in Siemens programs. For Space, as discussed above, this is likely due to the much larger size of this program (see Table 1).

---

[3] This requires to use as significance threshold the ratio α divided by the number of mean comparisons, that is 28 in our case.

[4] An equivalent, non-parametric Mann-Whitney Test [5, 18] yields the same results. Other procedures for multiple mean comparisons, such as Tukey's method, can't be used as we have unequal sample sizes across programs [18].

These reverse trends for mutants and faults supports our interpretation that the differences across programs are not due to different test case detection abilities, as this would instead show consistent trends for both mutants *and* faults, i.e., Space test cases detecting more mutants and faults. Figures 2 and 3 as well their associated statistical test results therefore support our conjecture that the most plausible interpretation for the trends in Section 4.1 is that faults seeded in Space tend to be easier to detect because they are real faults, as opposed to faults being devised by humans as is the case for the Siemens programs.

4.5. Discussion

Based on the above results we can deduce that the fact that the test suite mutant adequacy ratios are comparable to fault adequacy ratios in Space, whereas they are very different for other programs, is due to the fact that the faults seeded in the other programs are probably not representative in terms of ease of detection. After all, they are not real faults and it is likely difficult for humans to gauge the detectability of seeded faults.

The paper in which these programs were reported for the first time [11] actually contains corroboration of this conclusion. Hutchins et al. clearly state that, from an initial large set of faults suggested by developers, faults that were detected by 350 or more test cases (from their original test pool) were discarded[6]. This supports our conjecture above that the results for Space are the ones that should be given more credibility. If this is the case, then we can conclude that mutants, based on the mutation operators presented here, do provide test effectiveness results that are representative of real faults.

It also implies that any research done on human-seeded faults should be carefully interpreted, since we see that by using the example programs used here—which have been widely used in previous research [6]—one would obtain conservative results when assessing the fault detection rates of test techniques. In other words, this would lead to underestimating the effectiveness of test techniques. This is especially important in situations in which one is measuring the cost-effectiveness of different testing techniques. If technique A detects a greater proportion of the hard-to-detect Siemens faults than does technique B, but A is more expensive than B, we may not actually have proven the relative cost-effectiveness of A.

5. Conclusions

The main contribution of the paper is a thorough investigation of a fundamental assumption that has been underlying much of experimental software testing research to date: Faults generated by hand (e.g., by experienced engineers) or from mutation operators are representative of real faults. Indeed, most experimental results rely on seeded mutants [1, 2, 12, 13, 20]—using mutation operators—or faults selected by experienced developers [7, 11]. Furthermore, this problem is not likely to go away in future research as real faults are usually too few to be amenable to experiments allowing statistical analysis. So even when real faults are available for a particular subject program, one is often forced to seed additional faults.

Therefore, a fundamental question is whether mutants or manually seeded faults are likely to yield results, for example in terms of detection ability, that are representative of what one would obtain on real faults. Our analysis suggests that mutants, when using carefully selected mutation operators and after removing equivalent mutants, can provide a good indication of the fault detection ability of a test suite. Furthermore, it shows the danger of using faults selected by humans as, in the systems studied in this article, it leads to underestimating the fault detection ability of test suites. Another issue with faults selected by humans is more related to the necessity of building over time an experimental body of results regarding test techniques' efficiency. For this to be possible, studies must be replicated by researchers. This is very difficult to achieve when faults have been selected based on a subjective, undefined process.

We should also note that, although our focus in this paper has been to show that the mutants we generated were not easier to detect than real faults, our data also suggest that they were not harder to detect than real faults to any practical degree. This fact lends support to the "competent programmer assumption" that underlies the theory of mutation testing; that is, the assumption that programmers tend to make relatively small mistakes.

Future research requires of course to replicate the study we report in this paper. We did so in such a way that it should facilitate the precise replication of our analysis and therefore the comparison of future results with ours. It would also be important to perform similar studies in the context of object-oriented systems, with suitable mutation operators, as those systems represent an increasing share of industrial systems. Finally, since the results in this paper report only on randomly-selected test suites, it will be important also to study whether they are the same for test suites selected according to various criteria, such as code coverage criteria.

---

[6] They report that about 38% of the faults were discarded for that reason, with 18% of the faults being discarded for the complementary reason that they were too hard to detect.

6. Acknowledgements

Many thanks to Gregg Rothermel and Hyunsook Do for valuable discussions and suggestions, and for sending us the Siemens and Space programs and all the associated artifacts. Thanks also to all the researchers who worked on and improved these subject programs and artifacts over the years. We would like to also thank Mike Sowka for reviewing drafts of this paper. This work was partly supported by a Canada Research Chair (CRC) grant. Jamie Andrews, Lionel Briand and Yvan Labiche were further supported by NSERC operational grants.

7. References

[1] Andrews, J.H. and Y. Zhang, *General Test Result Checking with Log File Analysis*, IEEE Transactions on Software Engineering, vol. 29 (7), pp. 634-648, 2003.

[2] Briand, L., Y. Labiche and Y. Wang, Using Simulation to Empirically Investigate Test Coverage Criteria, *Proc. IEEE/ACM International Conference on Software Engineering*, pp. 86-95, 2004.

[3] Chen, W., R.H. Untch, G. Rothermel, S. Elbaum and J. von Ronne, *Can fault-exposure-potential estimates improve the fault detection abilities of test suites?*, Software Testing, Verification and Reliability, vol. 12 (4), pp. 197-218, 2002.

[4] DeMillo, R.A., R. J. Lipton and F. G. Sayward. *Hints on Test Data Selection: Help for the Practicing Programmer*, in *IEEE Computer*. 1978. p. 34-41.

[5] Devore, J.L., *Probability and Statistics for Engineering and the Sciences*, 5th ed, Duxbury Press, 1999.

[6] Do, H., G. Rothermel and S. Elbaum, Infrastructure support for controlled experimentation with software testing and regression testing techniques, Oregon State University, Corvallis, OR, USA, Technical report 04-06-01, 2004.

[7] Frankl, P.G. and S.N. Weiss, An experimental comparison of the effectiveness of the all-uses and all-edges adequacy criteria, *Proc. 4th Symposium on Testing, Analysis, and Verification*, pp. 154-164, 1991.

[8] Graves, T.L., M.J. Harrold, J.-M. Kim, A. Porter and G. Rothermel, *An Empirical Study of Regression Test Selection Techniques*, ACM Transactions on Software Engineering and Methodology, vol. 10 (2), pp. 184-208, 2001.

[9] Hamlet, R.G., *Testing programs with the aid of a compiler*, IEEE Transactions on Software Engineering, vol. 3 (4), pp. 279-290, 1977.

[10] Harder, M., J. Mellen and M.D. Ernst, Improving Test Suites via Operational Abstraction, *Proc. 25th International Conference on Software Engineering*, pp. 60-71, 2003.

[11] Hutchins, M., H. Foster, T. Goradia and T. Ostrand, Experiments on the Effectiveness of Dataflow- and Controlflow-Based Test Adequacy Criteria, *Proc. 16th IEEE International Conference on Software Engineering*, pp. 191-200, 1994.

[12] Kim, S., J.A. Clark and J.A. McDermid, *Investigating the Effectiveness of Object-Oriented Testing Strategies with the Mutation Method*, Software Testing, Verification and Reliability, vol. 11 (3), pp. 207-225, 2001.

[13] Memon, A.M., I. Banerjee and A. Nagarajan, What Test Oracle Should I use for Effective GUI Testing?, *Proc. IEEE International Conference on Automated Software Engineering (ASE'03)*, pp. 164-173, 2003.

[14] Offutt, A.J., *Investigations of the Software Testing Coupling Effect*, ACM Transactions on Software Engineering and Methodology, vol. 1 (1), pp. 3-18, 1992.

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

[16] Offutt, A.J. and J. Pan, *Detecting Equivalent Mutants and the Feasible Path Problem*, Software Testing, Verification, and Reliability, vol. 7 (3), pp. 165-192, 1997.

[17] Offutt, A.J. and R.H. Untch, Mutation 2000: Uniting the Orthogonal, *Proc. Mutation*, pp. 45-55, 2000.

[18] Rice, J., *Mathematical Statistics and Data Analysis*, 2nd ed, Duxbury press, 1995.

[19] Rothermel, G. and M.J. Harrold, *Empirical Studies of a Safe Regression Test Selection Technique*, IEEE Trans. on Software Engineering, vol. 24 (6), pp. 401-419, 1998.

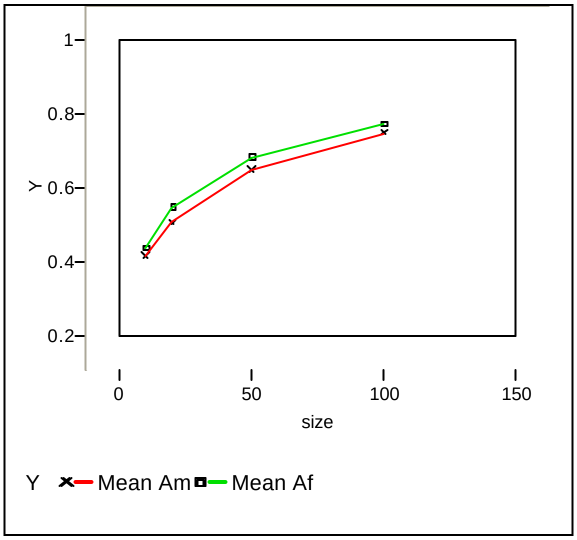
[20] Thévenod-Fosse, P., H. Waeselynck and Y. Crouzet, An experimental study on software structural testing: deterministic versus random input generation, *Proc. 21st International Symposium on Fault-Tolerant Computing*, pp. 410-417, 1991.

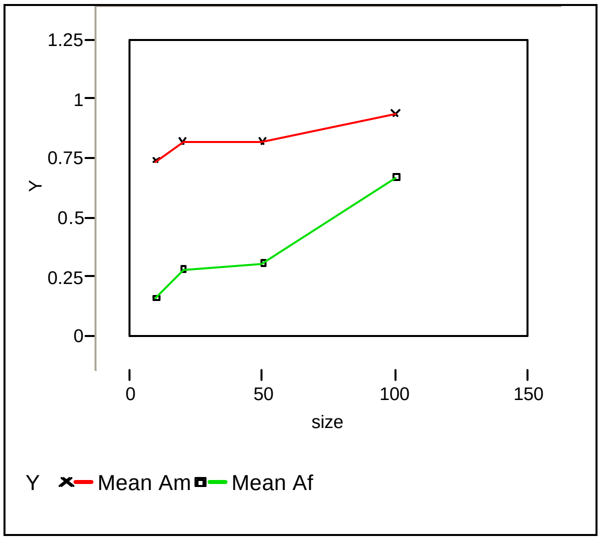
[21] Vokolos, F.I. and P.G. Frankl, Empirical evaluation of the textual differencing regression testing technique, *Proc. IEEE International Conference on Software Maintenance*, pp. 44-53, 1998.

8. Appendix

Figure 2 and Figure 3 show how the means of *Am* and *Af* changes as a function of the test suite size. We can observe that, regardless of size, *Af* and *Am* are very similar for Space. However, for Replace, as for the remaining programs, the difference is substantial and tends to grow as test suite size decreases.

Figure 2. Mean(Am) and Mean(Af) versus test suite size - Space



Figure 3. Mean(Am) and Mean(Af) versus test suite size - Replace