An Empirical Evaluation of Frequency-Based Statistical Models for Estimating Killable Mutants

Konstantin Kuznetsov
Saarland University
Germany
konstantin.kuznetsov@cispa.de

Alessio Gambi
Austrian Institute of Technology
Austria
alessio.gambi@ait.ac.at

Saikrishna Dhiddi
University of Passau
Germany
dhiddi@ads.uni-passau.de

Julia Hess
Saarland University
Germany
s9jahess@stud.uni-saarland.de

Rahul Gopinath
University of Sydney
Australia
rahul.gopinath@sydney.edu.au

ABSTRACT

Background. Mutation analysis is the premier technique for evaluating test suite quality estimating residual software defects. However, the reliability of mutation analysis is hampered by *equivalent mutants* which are undetectable by test cases. Reliably detecting and eliminating killable mutants is difficult as it is highly program and location dependent. Statistical estimation of killable mutants seems to be a promising approach to tackle this problem.
Aims. Frequency-based species estimation methods have been proposed as a solution for several related problems in software testing. This paper investigates whether such frequency-based estimation methods can accurately estimate the number of killable mutants.
Method. We conducted a large-scale empirical study on the ability of twelve widely known frequency-based estimators to predict the number of killable mutants in ten mature software projects.
Result. Our investigation finds limited or no evidence that any of the statistical estimators are able to consistently predict the number of killable mutants in projects evaluated.
Conclusion. We found that the investigated estimators lack sufficient predictive power and cannot produce reliable and useful estimates of killable mutants.

CCS CONCEPTS

• **Software and its engineering → Software verification and validation**; **Software testing and debugging**; **Empirical software validation**.

ACM Reference Format:
Konstantin Kuznetsov, Alessio Gambi, Saikrishna Dhiddi, Julia Hess, and Rahul Gopinath. 2024. An Empirical Evaluation of Frequency-Based Statistical Models for Estimating Killable Mutants. In . ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

```
def determinant(a,b,c,d):
  ad = a * d
  bc = b * c
  return ad - bc
```

```
# Tests

⊢ determinant(1,2,1,2) = 0
⊢ determinant(1,2,3,4) = -2
```

Figure 1: A simple program (left) and two of its tests (right)

```
def determinant(a,b,c,d):
- ad = a * d
+ ad = a / d
  bc = b * c
  return ad - bc
```

```
def determinant(a,b,c,d):
  ad = a * d
- bc = b * c
+ bc = c * b
  return ad - bc
```

Figure 2: Killable (left) and equivalent (right) mutants

1 INTRODUCTION

Mutation analysis is the premier means of assessing the quality of software test suites in preventing defects [45], and estimating residual risk [30]. Mutation analysis involves generating mutants *exhaustively* and evaluating them against the test suite under consideration (Figure 1-right). Mutants are copies of the original code (Figure 1-left) in which artificial code *mutations* that share strong similarities with real faults are inserted (Figure 2) [4, 5, 21, 33].

A test case detects (or *kills*) a mutant if it induces and observes a change in behavior in the mutant when compared to the original program. The mutants undetected by any test case are called *surviving* mutants. The ratio of the number of mutants killed by a test suite to the number of *killable* mutants is called *mutation score* a good indicator for test suite effectiveness in preventing faults [32]. However, some mutants (Figure 2-right) are behaviorally *equivalent* to the original program [12]. The amount of such mutants is program specific [29, 40] and cannot be established *a-priori*. Consequently, the mutation score might be inaccurate and highly variable. Hence, an accurate estimate of the number of killable mutants is important in software testing, but has remained unsolved.

Recently, the *Software Testing as Species Discovery (STADS) framework* [9] was proposed for estimating *reachable but yet to be covered* elements remaining for coverage evaluation. The STADS framework relies on an estimator called Chao's estimator [17]. It is based on the idea that the relative frequency of coverage of software elements, such as statements or branches, by test cases contains information about the number of similar software elements that are reachable but yet to be covered. Frequency-based estimators such as Chao's [17] are designed to be robust toward strong biases in sampling making them attractive in areas such as reachable coverage estimation which can be strongly program dependent.

Given the closeness of mutation analysis and coverage analysis—covering a mutant is a prerequisite for killing it—we adapted the STADS framework [10] for predicting killable mutants. That is, we estimate the count of killable mutants from the frequency of mutants killed by test cases. We count the number of mutants killed by a single test case, those killed by two test cases, and so on, and use this data to predict the remaining mutants that can be killed.

> The idea in frequency-based mutant count estimation is that the ratio of number of mutants that are killed by a single test case vs those killed by two test cases, etc., contains the information for predicting those that are not yet killed.

This leads to the first research question.

(1) How accurate are frequency-based estimators in predicting killable mutants?

To ensure **ground truth** for comparison, we first conducted a large scale **manual evaluation of surviving mutants** from ten mature well-tested open-source projects. The mutants were generated by the state-of-the-art mutation testing framework PIT [20].

Chao's estimator preferred by STADS is one among the many frequency-based species estimators in the literature with different assumptions and prediction accuracies. Since it is not clear which of these estimators may be applicable to estimating killable mutants, we conducted a large scale empirical study of **twelve** different frequency-based statistical estimators from the literature.

As there is little information on robustness of frequency-based estimators to sampling biases, the second research question is:

(2) Are frequency-based estimators affected by sampling strategies?

We used three different test suites—*developer written*, *randomly generated*, and *feedback-driven* to evaluate this question. If the estimators are indeed robust to sampling bias, then the estimations should coincide or at least overlap significantly. We should also be able to identify best strategies by comparison to ground truth.

Are frequency-based estimators affected by the specific sampling unit used, such as using test class as a test unit instead of test method? This informs our next question.

(3) Are frequency-based estimators affected by sampling units?

To evaluate the impact of different sampling units, we reanalyzed the data from our first experiment but with test classes (which contain multiple test methods) as sampling units. We hypothesized that the estimates should overlap, with the difference in accuracy (i.e., variance of the estimation) being the only major difference. We compare the results of both aggregations with the ground truth.

> Our large study empirically investigates the hypothesis that (at least) one of the surveyed frequency-based estimators can be effectively used to estimate killable mutants.

This paper summarizes the main findings of our study and is complemented by the replication package [1].

Empirical Study Results Overview. For evaluating (1), after selecting ten well maintained, open-source projects and analyzing them using PIT, we manually classified 1016 surviving mutants and used this result for assessing the prediction quality of frequency-based statistical estimators (see Section 2.2). We found that the estimates from the selected estimators were significantly different from the manual estimates. Hence, we questioned our experimental settings and identified two major threats to the validity of our study: (1) the manual classification of live mutants could be in error, or (2) the tests themselves could have been biased, with more effective tests checking only specific parts of the code (e.g., business logic), and hence, not sufficiently random.

To mitigate the first threat, we extended our initial study with test suites created with EvoSuite [24], using two fundamentally different configurations (i.e., a random strategy and a strategy guided by coverage). We expected the estimations obtained from manual test suites to coincide with estimations from EvoSuite test suites as they predict the same quantity. We evaluate this in (2). To mitigate the second threat, we compared the estimates from EvoSuite test suites and from the manual classification. We expected the estimates from EvoSuite to match estimates from manually classified live mutants as EvoSuite test suites are free of manual bias.

We, again, found that manual classification was significantly different from estimations from automatically generated test suites, and the estimates from the automatically generated test suites were significantly different from each other. Therefore, we critically inspected our experiment and identified another possible error source–perhaps, different test objectives in generating test suites may have an impact. If so, and if estimation from at least one of the test suites is correct, then changing the sampling unit from test methods to test classes should not significantly impact the estimates from a given test suite. We investigated this in (3). As both test methods and test classes sample the same quantity, i.e., the killed mutants, we expected that the estimators to produce consistent estimates.

Once again, our results showed that the estimates from test methods and test classes were significantly different, which left us with the only option that the considered frequency-based statistical estimators cannot be reliably applied to mutation analysis.

Contributions. In summary, our main contributions are:

- The first empirical study on the application of twelve, widely used frequency-based statistical estimators on estimating killable mutants, identifying their limits.
- One of the largest mutation analysis datasets containing 1016 live mutants manually classified by three researchers on ten mature projects, multiple test suites generated manually and automatically, and their mutation analysis resulting in more than 2.5B test executions.

2 STATISTICAL FRAMEWORK

We wish to measure the number of killable mutants with help of statistical estimators from biometrics. These estimators have also been successful in counting systems in computer networks [2] and estimating reachable coverage [10].

We next introduce the foundational model underlying statistical estimation and the statistical estimators for our study.

2.1 The Urn Probabilistic Model

Consider the following probabilistic model: We have an urn with colored balls from which n balls are sampled with replacement. Let $S(n)$ be the colors observed. Furthermore, let us assume that each ball has multiple colors. We are interested in how many colors S this urn can contain. In application to our problem, this urn sampling, which is described with the Bernoulli product model, associates each test (a ball) with the killed mutants (the colors of that ball). We use the definitions from the Bernoulli product described in the STADS framework to formalize the intuition.

Let $\mathcal{P}$ be the program under test and $\mathcal{D}$ the set of all tests X that exercise $\mathcal{P}$. We model a testing campaign $\mathcal{T}$ as a stochastic process

$$\mathcal{T} = \{X_n | X_n \in \mathcal{D}\}_{n=1}^{T}$$

where T tests are sampled with replacement from $\mathcal{D}$. Let $\{M_i\}_{i=1}^{S}$ be a set of mutants. Mutant M_i can be detected with a probability π_i that might be affected by factors specific to M_i's definition (e.g., the mutation operators that generated it or the location in the code where it is applied). In the Bernoulli product model, a test can kill one or more mutants. For a testing campaign of size T, we let the *kill incidence* matrix, or simply killmatrix, $W_{S\times T}$ be defined as

$$W_{S\times T} = \left\{W_{ij} | i = 1, 2, \ldots, S \wedge j = 1, 2, \ldots, T\right\},$$

where $W_{ij} = 1$ if test X_j kills mutant M_i and $W_{ij} = 0$ otherwise.

This way, the i_{th}-row sum of W ($Y_i = \sum_{j=1}^{T} W_{i,j}$) denotes the incidence-based frequency of M_i being killed. We define the incidence frequency counts Q_k, where $0 \le k \le T$, as the number of mutants killed by exactly k tests. Consequently, the *unobservable* frequency count Q_0 denotes the number of undetected mutants.

We assume that the probability that a mutant M_i is detected by a test X_j is defined as $P(W_{i,j} = 1) = \pi_i \cdot \nu_j$, where variables $\{\nu_1, \nu_2, \cdots, \nu_T\}$ are responsible for test effects. Indeed, the ability of a test to kill mutants might be affected by various factors, such as coverage, input data, environment or flakiness. We model those test effects as a random variable from an unknown probability density function $h(\nu)$, whereas we assume fixed mutant detection rates π_i. Hence, we model probability distribution of each element W_{ij} of the killmatrix as a Bernoulli random variable conditioned on ν_j:

$$P(\forall(i,j) W_{ij} = w_{ij} | \nu_j) = (\pi_i \nu_j)^{w_{ij}} (1 - \pi_i \nu_j)^{1-w_{ij}}$$

The probability distribution for the incidence matrix can be expressed as the probability for all $i : 1 \le i \le S$ and $j : 1 \le j \le T$ that we have $W_{ij} = w_{ij}$.

$$P\left(\forall(i,j) W_{ij} = w_{ij} | \nu_i\right) = \prod_{j=1}^{T} \prod_{i=1}^{S} \pi_i \nu_j{}^{w_{ij}} (1 - p_i)^{1-w_{ij}}$$

Integrating all possible values of $\{\nu_1, \nu_2, \cdots, \nu_T\}$, we obtain the unconditional marginal distribution for the incidence-based frequency Y_i for the mutant M_i, which follows Binomial distribution:

$$\begin{aligned} P(Y_i = y_i) &= \binom{T}{y_i} \left[\pi_i \int \nu h(\nu) d\nu\right]^{y_i} \left[1 - \pi_i \int \nu h(\nu) d\nu\right]^{T-y_i} \\ &= \binom{T}{y_i} \lambda_i{}^{y_i} (1 - \lambda_i{}^{y_i})^{T-y_i}, \end{aligned}$$

where $\lambda_i = \pi_i \int \nu h(\nu) d\nu$. That is, the frequency Y_i is a binomial random variable with detection probability λ_i, and the incidence frequency counts Q_k can be derived as:

$$Q_k = E\left[\sum_{i=1}^{S} I(Y_i = k)\right] = \sum_{i=1}^{S} \binom{T}{k} \lambda_i^k (1 - \lambda_i)^{T-k}.$$

2.2 Frequency-Based Estimators

Estimators that can estimate the total number of colors under Bernoulli Product model belong to the class of frequency-based estimators. They have been used extensively in biometrics (Ecology) to estimate *unseen* species [17], and are also called *species richness* estimators. Estimating species richness in a given geographical area is challenging because the number of species is often very large, preventing an exhaustive survey. Hence, ecologists use *sampling*: (1) dividing the area into *sampling units*, and (2) randomly selecting sampling units to survey for number of species found.

Sampling data might show different distributions of species in sampling units and might be incomplete. Hence, species richness estimation estimates the *total number of species* that is present in the considered geographical area *including species not found* in any sampling unit. The idea is that we can estimate the number of species that did not show up in any sampling unit by considering the next rarest species, such as species detected only once (*singleton*), twice (*doubleton*), and the number of species found.

Chao [17] identified two kinds of species richness estimators based on whether they adopt *incidence data*, i.e., data about the presence of species across multiple sampling units, or *abundance data*, i.e., data about the number of individuals of different species found. As our model leverages kill incidence matrix, we primarily resort to incidence sampling estimators.

Incidence sampling considers T sampling units randomly selected among all the available ones and assumes these are independent [17]. In each sampling unit, the relevant (categorical) data is the presence of various species; hence, incidence sampling data do not consider the explored size of each species. After surveying all the T sampling units, incidence sampling reports the count of species that appear only once (Q_1), twice (Q_2), and so on. Using these values, the estimators predicts Q_0, i.e., the number of species that never appeared during sampling.

2.2.1 *Chao estimators [15].* The basic Chao estimator is:

$$\hat{S}_{Chao} = \begin{cases} S_{obs} + \frac{T-1}{T} Q_1^2/(2Q_2) & \text{if } Q_2 > 0. \\ S_{obs} + \frac{T-1}{T} Q_1(Q_1 - 1)/2 & \text{otherwise.} \end{cases}$$

S_{obs} is the observed species count, Q_1 and Q_2 the frequency of singleton and doubleton species, and T is the number of sampled units. This estimator is represented as `Chao` in the rest of the paper.

Chiu et al. [19] derived an improved version (referred to as `iChao`), which makes use of the additional information of tripletons Q_3 and quadrupletons Q_4 to estimate undetected species richness.

$$\hat{S}_{iChao} = S_{Chao} + \frac{T-3}{T} \frac{Q_3}{4Q_4} \times \max\left[Q_1 - \frac{T-3}{T-1} \frac{Q_2 Q_3}{2Q_4}, 0\right]$$

Chao estimators are known to provide *lower bounds* estimates.

2.2.2 Jackknife estimator [13, 14]. The first and second order Jackknife estimators are computed as follows:

$$S_{jack1} = S_{obs} + Q_1\left(\frac{T-1}{T}\right)$$
$$S_{jack2} = S_{obs} + \frac{2T-3}{T}Q_1 - \frac{(T-2)^2}{T(T-1)}Q_2.$$

The second order Jackknife [47] is more robust to sampling bias [31] but has larger standard error. Higher-order Jackknife estimators (S_{jackJ}) are constructed similarly. We consider up to J = 5, and automatically select the best performing one (referred as `Jackknife`). These estimators underestimate on small sample size and overestimate otherwise [17].

2.2.3 Incidence coverage estimator [18]. The Incidence Coverage Estimator (referred as `ICE`) is given by:

$$S_{ICE} = S_{freq} + \frac{S_{infreq}}{\hat{C}_{infreq}} + \frac{Q_1}{\hat{C}_{infreq}}\hat{\gamma}^2_{infreq}, \quad \hat{C}_{infreq} = 1 - Q_1 / \sum_{i=1}^{k} Q_i,$$

$$\hat{\gamma}^2_{infreq} = \max\left[\frac{S_{infreq}}{\hat{C}_{infreq}}\frac{T}{T-1}\frac{\sum_{i=1}^{k} i(i-1)Q_i}{\left(\sum_{i=1}^{k} iQ_i\right)\left(\sum_{i=1}^{k} iQ_i - 1\right)} - 1, 0\right]$$

S_{freq} is the number of species that occur more than k times, while S_{infreq} is those that do not. A $k = 10$ cut-off is recommended. We also considered the original coverage estimator as first proposed by Lee et al. [34], which is equivalent to the `ICE` with $k = 0$ (`ICE-k0`). Gotelli et al. [28] provided a version (referred as `ICE-1`) for highly heterogeneous communities, which underestimates less.

2.2.4 Bootstrap estimator [47]. The Bootstrap estimator is based on resampling T sampling units with replacement from a set of initially observed T sampling units a sufficient number of times m.

$$S_{bootstrap} = avg_m(S_{obs} + \sum_{j=1}^{S_{obs}}(1 - Y_j/T)^T)$$

Y_j is the number of sampling units where species j is detected.

2.2.5 Zelterman estimator [11]. Zelterman's estimator (referred as `Zelterman`) is frequently used in Social Sciences, in particular in illicit drug use research. It is defined as:

$$S_{zelterman} = S_{obs} + S_{obs}/[(1+\lambda)^T - 1], \hat{\lambda} = 2Q_2/(m-1)Q_1$$

2.2.6 Other estimators. So far we discussed incidence estimators that take as an input incidence data. However, we can consider the whole test suite as one single sample and count mutant kills.

With abundance data, we explore `Chao-Bunge`, UNPMLE, PNPMLE, and PCG estimators. Since they have complex formulation, we suggest the reader to refer to the corresponding papers for details. Some abundance estimators assume Poisson rather than Binomial distribution of counts, which can be considered as a limiting case when the number of tests tends to infinity (i.e., T is sufficiently large). The Chao-Bunge estimator, proposed by Chao and Bunge [16], utilizes the gamma-Poisson model in which species are detected in the sample according to a Poisson process and rates of the processes follow a gamma distribution. The Unconditional nonparametric maximum likelihood estimator (UNPMLE) was provided by Norris and Pollock [39]. The Penalized nonparametric maximum likelihood estimator (PNPMLE) was provided by Wang et al. [52]. Wang et al. [51] provided the Poisson-compound Gamma model with smooth nonparametric maximum likelihood estimation (PCG).

3 METHODOLOGY

Our methodology consisted of the following steps: (1) test subjects selection (Section 3.1); (2) test suite generation (Section 3.2); and, (3) killable mutants estimation (Section 3.3). After estimating the killable mutants for each test suite, sampling strategy, and selected project, we needed ground truth data about killable mutants to assess the estimation accuracy. Since such data was not available, we (4) manually classified a sample of surviving mutants as killable or equivalent (Section 3.4).

3.1 Test Subjects Selection

The choice of our test subjects was guided by the following considerations: (1) We consider manual mutant classification as the most important part of our study; hence, we wanted to ensure that our classifiers could understand easily what a program fragment does, thus closely matching real world settings, in which programmers work on well understood projects. (2) We wanted to reduce the work involved in setting up our experiments on different computing infrastructures, thus increasing the reproducibility of our results. Hence, we focused on large open-source *Java* projects that do not depend on external resources (e.g., databases), and built using standard tools such as Maven [7]. We found that the libraries published by *Apache Commons* [6] met our criteria. In particular, they are written to an exacting standard and follow common syntactic and semantic guidelines as in large enterprise projects. Thus, our classifiers could understand their functioning reasonably well after some initial training. At the time we conducted this study, Apache Commons contained the 41 projects listed in Table 1. Among those, for our study, we selected only projects that (1) could be built and tested successfully with minimal effort; (2) completed mutation analysis successfully; (3) are released more than once; and, (4) are packaged as a single Maven module.

For the sake of clarity, in Table 1 we group the Apache Commons projects into two groups: the first group (top) contains the 21 projects (name, release, and commit hash) that we could build, test, and analyze successfully; the second group (bottom) contains the projects (name, release, and rejection note) that we discarded. Specifically, we discarded 3 projects that had no official release at the time we conducted this study and 17 projects that have a complex structure, fail to build, do not pass the available tests, or break mutation analysis even after some effort into fixing them.

Since we required expensive manual mutant classification, we restricted our analysis to *ten largest projects* (in terms of project KLOC) fulfilling our criteria given in Table 1, and their test suite spectra are given in Table 2.

3.2 Test Suites Generation

The selected test subjects contain unit tests from developers for application- and domain- specific requirements. One may argue that the mutants killed by ORIGINAL test suites might be far from random as expected by a statistical process. For mitigation, we leveraged EVOSUITE [24] to generate RANDOM, a test suite that does

Table 1: List of considered Apache Commons projects

Project	Release	Commit Hash/Note
commons-beanutils	1.9.4-RC2	32ceb2c925
commons-cli	1.4	f7153c3c10
commons-codec	1.13	beafa49f88
commons-collections	4.4	cab58b3a80
commons-compress	1.18	b95d5cde4c
commons-configuration	2.5	dc00a04783
commons-csv	1.7	a227a1e2fb
commons-dbcp	2.6.0	3e7fca08d3
commons-dbutils	1.7	77faa3caef
commons-digester	3.2	ec75748096
commons-email	1.5	5516c487a5
commons-exec	1.3	0b1c1ff0cb
commons-fileupload	1.4	047f315764
commons-functor	1.0_RC1	62cd20998e
commons-imaging	1.0-alpha1-RC3	6f04ccc2cf
commons-io	2.6-RC3	2ae025fe5c
commons-lang	3.8_RC1	9801e2fb9f
commons-math	3.6_RC2	95a9d35e77
commons-net	3.6	163fe46c01
commons-pool	2.7.0	f4455dcb8a
commons-validator	1.6	c4b93a7275
commons-ognl	no releases	immature project
commons-geometry	no releases	immature project
commons-testing	no releases	immature project
commons-bcel	6.3.1	fail build
commons-vfs	2.4	complex structure
commons-text	1.7	fail build
commons-logging	1.2	fail mutation analysis
commons-jexl	v4.0-snapshot.4	failing tests
commons-crypto	1.0.0	failing tests
commons-jcs	2.2.1-RC4	failing tests
commons-chain	1.2	fail mutation analysis
commons-jxpath	1.3	fail mutation analysis
commons-scxml	2.0-M1	fail mutation analysis
commons-rng	1.2	complex structure
commons-rdf	0.5.0	complex structure
commons-proxy	2.0 RC1	fail build
commons-bsf	3.x-with-engines	complex structure
commons-weaver	2.0	complex structure
commons-jelly	1.0.1	fail build
commons-jci	1.1	complex structure

not target any specific testing goal, and DynaMOSA, a test suite that maximizes coverage using the Dynamic Many-Objective Sorting Algorithm [42]. Consequently, we argue that the former test suite has the lowest generation bias, whereas the latter might introduce some bias being not completely random, but still free from manual bias. For generating both test suites, we followed the best practice from SBST [23, 25, 43], with test generation for 60 seconds each, and test minimization for 300 seconds each.

3.3 Killable Mutants Estimation

We used PIT [20] (v 1.4.9) to generate the mutants and execute test suites. This resulted in 2.5B test method executions. We configured PIT to use its *default*[1] mutation operators and the *killmatrix* option enabled; however, we noticed that PIT (1) did not always identify all the mutants covered by the test cases; hence, it did not execute them; and (2) it did not always report the name of the test methods killing mutants. For the test execution, we used *JUGE* [22], which we extended to address PIT's limitations and parallelize the test executions across multiple computing nodes.

After executing the tests, we removed faulty samples related to invalid and timed-out mutants. To reduce the risk of misclassifying "slow" mutants as killed, we conservatively granted each unit test a timeout of 10 seconds. We considered those mutants that PIT marked as `SURVIVED` or `NOT_COVERED` as the surviving mutants.

Sampling units. We first estimated the killable mutants on the ten test subjects using the twelve estimators on each of the test suites. We first estimated the killable units using test methods as sampling units. That is, each mutant kill by a test method is counted separately. Next, we evaluated the same using test classes as sampling units. That is, we counted mutant kills by each test class.

3.4 Manual Mutants Classification

Running PIT on the Original test suites left thousands of surviving mutants. Since we could not exhaustively classify all of those in a reasonable time, we randomly sampled 100 survived mutants for each project and used those for the manual classification. We argue that classifying so many mutants is a good balance between classification effort (estimated to be between one and thirty minutes per mutant) and representativeness of the obtained results. Nevertheless, the whole manual classification took six months.[2]

Given the sampled mutants, we adopted a structured classification protocol that involved three classifiers. The lead classifier (R_A) has ten years of experience in programming with Java and is an expert in mutation analysis. The classifiers R_B and R_C have experience in programming (between three and five years), but were unfamiliar with mutation analysis before the start of this study. To cope with that, we organized a pilot study for training R_B and R_C using *commons-csv*, the smallest among the selected projects, and all the 116 mutants that survived its Original test suite.

After the initial training, the classification considered one project at a time as listed in Table 3 and required researchers R_B and R_C to classify *independently* all 100 sampled mutants. Killable mutants must be accompanied by a (hypothetical) unit test that could show the difference in behavior between that mutant and the original program. Similarly, equivalent mutants must be motivated with a convincing explanation. As a result, for each sampled mutant, we collected a label (killable or equivalent) along with how confident the classifier was in the classification (high, medium, low), and optional comments. Between R_B and R_C we achieved a Cohen's Kappa of 0.48 (moderate agreement).

Finally, we identified conflicting classifications (avg 4.9%) and let the three classifiers discuss and resolve them. During the discussion, R_A acted as the moderator while R_B and R_C could update their classifications in light of the discussed arguments. The discussion continued until the final classification was accepted unanimously.

Given the proportion of killable mutants in a sample, we obtained the population proportion—and thus an estimate of the total number of killable mutants—with population proportion confidence intervals [41]. The estimates from manual classification are the 95% confidence intervals based on random sampling.

[1] https://pitest.org/quickstart/mutators/

[2] The manual classification ran between July 2019 and December 2019.

Table 2: Apache Commons projects — Projects and test suites measures

Project			Total	ORIGINAL				RANDOM				DYNAMOSA			
Id	Name	KLOC	Mutants	Size	Stmt	Branch	Kill	Size	Stmt	Branch	Kill	Size	Stmt	Branch	Kill
1	commons-net	20	5764	254	33%	28%	28%	1988	48%	35%	32%	3499	51%	42%	35%
2	commons-math	100	47881	6377	92%	84%	76%	11956	74%	61%	77%	17450	79%	68%	18%
3	commons-lang	28	13061	4114	95%	91%	66%	4500	73%	60%	72%	9027	89%	86%	86%
4	commons-io	10	3273	1316	90%	88%	94%	1677	72%	65%	58%	2822	81%	79%	87%
5	commons-imaging	31	11597	563	73%	59%	47%	2974	63%	45%	44%	4935	67%	53%	45%
6	commons-dbcp	14	4230	1409	66%	66%	99%	1026	39%	42%	20%	2970	47%	58%	27%
7	commons-csv	2	635	312	89%	85%	81%	365	78%	63%	60%	647	89%	83%	76%
8	commons-configuration	28	6279	2803	87%	83%	99%	2991	75%	62%	39%	4956	80%	72%	40%
9	commons-compress	24	9566	1047	84%	76%	65%	2610	63%	45%	44%	4306	70%	57%	49%
10	commons-collections	29	8309	25011	86%	81%	75%	3954	67%	59%	57%	5659	74%	69%	93%

Table 3: Manual classification of live mutants.

Id	Project	Sampled	Misclass.		Equivalent
			R_B	R_C	
1	commons-net	100	0	4	1
2	commons-math	100	10	0	8
3	commons-lang	100	7	3	12
4	commons-io	100	3	2	5
5	commons-imaging	100	0	2	0
6	commons-dbcp	100	0	0	1
7	commons-csv*	116	3	23	9
8	commons-configuration	100	0	2	4
9	commons-compress	100	2	4	1
10	commons-collections	100	3	2	1

*We used commons-csv as a pilot to train researchers R_B and R_C.

Table 4: Comparison of *mean difference (MD)* between method estimators across test subjects—ORIGINAL test suites

Id	Estimator	Valid Est.	CI Overlaps	Mean (%)	Stdev (%)
1	ICE-k0	4	0	5.78	6.52
2	Zelterman	6	1	13.48	14.06
3	Chao-Bunge	6	1	9.77	11.03
4	Jackknife	6	0	15.03	9.21
5	Chao	8	1	18.19	16.47
6	iChao	7	1	18.59	15.05
7	ICE	8	1	19.22	16.7
8	ICE-1	8	2	15.16	12.67
9	UNPMLE	6	2	13.28	11.0
10	Bootstrap	9	0	19.76	20.51
11	PNPMLE	5	1	17.31	11.84
12	PCG	6	3	9.14	10.52

4 RESULTS

Our study investigated the ability of frequency-based statistical estimators to predict the number of killable mutants.

(1) What is the accuracy of frequency-based estimators in predicting killable mutants?

To answer this question, we compared the twelve statistical estimators against the corresponding manual classification estimates (see Section 3.4). We considered estimates computed for the ORIGINAL test suite using test methods as sampling unit. We first checked whether the statistical estimators are significantly different from the manual classification estimates, i.e., their 95% confidence intervals do not overlap. Next, we computed the mean absolute percentage difference (MD) between the point estimates of the statistical estimators (E) and the manual classification (E^M) as: $\frac{1}{m}\sum_i \frac{|E_i - E_i^M|}{S_i} \times 100\%$. We removed any invalid estimate before computing MD, gaining m valid estimates. We consider invalid any estimate that failed to compute, resulted in negative point estimate or a value higher than the total number of possible mutants. We report the counts of valid estimates (column *Valid Estimates*) and statistics about MD (*Mean* and *Stdev*) aggregated across test subjects in Table 4.

(2) Are frequency-based estimators affected by sampling strategies?

To answer this questions, we follow the same approach adopted to answer (1). However, for this research question we compared the estimates from the manual classification also against those obtained by the twelve statistical estimators for RANDOM (Table 5) and DYNAMOSA (Table 6).

Table 5: Comparison of *mean difference (MD)* between method estimators across subjects—RANDOM test suites

Id	Estimator	Valid Est.	CI Overlaps	Mean (%)	Stdev (%)
1	ICE-k0	2	1	19.51	21.08
2	Zelterman	8	1	36.33	19.32
3	Chao-Bunge	8	1	30.95	17.16
4	Jackknife	8	0	34.28	19.55
5	Chao	9	1	39.42	20.82
6	iChao	9	0	37.42	20.63
7	ICE	9	1	37.79	20.41
8	ICE-1	8	0	38.12	17.22
9	UNPMLE	8	2	39.9	17.42
10	Bootstrap	10	2	36.53	23.15
11	PNPMLE	9	1	35.45	20.76
12	PCG	9	2	35.17	22.03

(3) Are frequency-based estimators affected by sampling units?

To answer this question, we compared the estimations of the twelve statistical estimators using test methods as sampling unit against the corresponding estimations using test classes as sampling unit

Table 6: Comparison of *mean difference (MD)* between method estimators across subjects—DynaMOSA test suites

Id	Estimator	Valid Est.	CI Overlaps	Mean (%)	Stdev (%)
1	ICE-k0	1	1	5.54	-
2	Zelterman	6	0	49.86	12.08
3	Chao-Bunge	8	1	39.83	23.29
4	Jackknife	7	1	36.58	22.31
5	Chao	8	1	43.54	25.27
6	iChao	8	0	42.15	24.45
7	ICE	8	1	43.67	25.09
8	ICE-1	8	1	41.92	24.36
9	UNPMLE	6	2	39.16	29.03
10	Bootstrap	7	0	49.66	20.85
11	PNPMLE	7	1	37.72	19.16
12	PCG	6	0	32.21	26.21

for all test suites. The comparison between class and method estimators for Original is given in Table 7. That for Random is given in Table 8, and for DynaMOSA is given in Table 9.

Table 7: Comparison of *mean difference (MD)* between class and method estimators across subjects—between Original test suites

Id	Estimator	Valid Est.	CI Overlaps	Mean (%)	Stdev (%)
1	ICE-k0	4	2	2.87	3.17
2	Zelterman	5	4	8.4	16.32
3	Chao-Bunge	4	2	2.15	4.19
4	Jackknife	3	0	4.14	6.42
5	Chao	5	2	10.66	16.86
6	iChao	4	2	4.55	5.27
7	ICE	4	1	4.56	5.63
8	ICE-1	4	1	6.06	7.78
9	UNPMLE	3	0	4.19	5.23
10	Bootstrap	5	3	5.71	2.73
11	PNPMLE	0	-	-	-
12	PCG	1	0	2.03	-

Table 8: Comparison of *mean difference (MD)* between class and method estimators across subjects—between Random test suites

Id	Estimator	Valid Est.	CI Overlaps	Mean (%)	Stdev (%)
1	ICE-k0	2	2	3.29	2.74
2	Zelterman	5	1	33.82	19.44
3	Chao-Bunge	1	0	26.3	-
4	Jackknife	3	0	36.82	15.25
5	Chao	6	0	28.66	15.8
6	iChao	5	0	30.06	13.52
7	ICE	5	0	23.69	10.54
8	ICE-1	2	0	31.15	10.81
9	UNPMLE	2	0	36.67	27.71
10	Bootstrap	9	5	6.79	2.71
11	PNPMLE	1	0	56.16	-
12	PCG	2	1	28.42	37.91

Table 9: Comparison of *mean difference (MD)* between class and method estimators across subjects—between DynaMOSA test suites

Id	Estimator	Valid Est.	CI Overlaps	Mean (%)	Stdev (%)
1	ICE-k0	0	-	-	-
2	Zelterman	2	0	22.0	0.85
3	Chao-Bunge	2	0	23.83	21.74
4	Jackknife	1	0	32.77	-
5	Chao	4	0	22.16	13.47
6	iChao	4	0	26.47	15.11
7	ICE	4	1	22.19	16.63
8	ICE-1	3	0	23.85	19.12
9	UNPMLE	0	-	-	-
10	Bootstrap	6	1	7.11	2.76
11	PNPMLE	1	0	58.84	-
12	PCG	0	-	-	-

5 DISCUSSION

We next discuss each of the research questions and corresponding observations in depth.

(1) What is the accuracy of frequency-based estimators in predicting killable mutants?

Inspecting Table 4, we can observe that (i) none of the twelve frequency-based estimators generated a valid estimate (VE) for all the test subjects (VE $\in [4, 9]$); (ii) in most of the cases (11/12), the estimates were significantly different than manual estimates (CI Overlaps ≤ 3), and (iii) their mean difference showed extreme variability (MD $\in (5.78 \pm 6.52, 19.76 \pm 20.51)$).

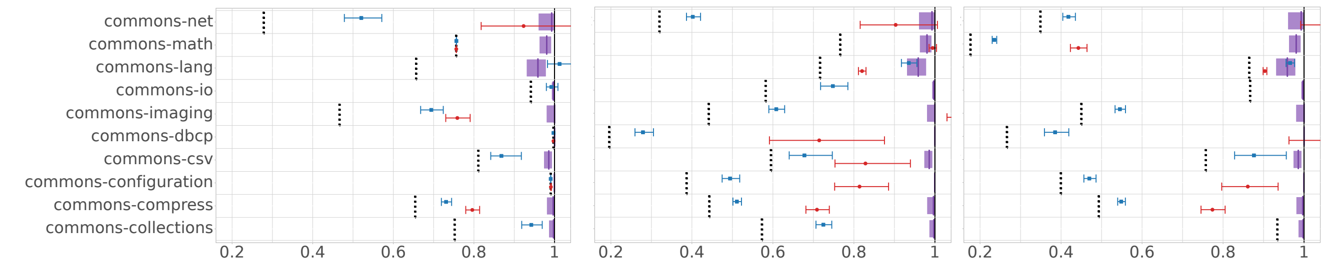
To better interpret the results in Table 4, we plot the results for Chao estimator (the estimator suggested by the STADS framework) in Figure 3a. In the figure, the x-axis represents the ratio of mutants, with 1.0 (bold solid line) indicating the complete set of generated mutants. The y-axis, instead, lists all the test subjects. For each test subject, the figure shows the ratio of killed mutants (black dotted line), the estimate and the 95% CI of the manual estimate of killable mutants (purple band), and the estimate and the 95% CI produced by Chao using the test methods (blue square) as sampling unit. For completeness, the figure also reports the estimates produced by Chao using test classes (red circle) as sampling unit. From the figure, it is evident that Chao's predictions are far from the manual estimates and their CIs almost never overlap.

Focusing on the best performing statistical estimators (Table 4), we can also observe that PCG achieved the maximum overlap with manual estimates; however, its CI overlapped E^M's only in three out of ten test subjects. Likewise, ICE-k0, which achieved the smallest mean difference of 5.78 from E^M, and Bootstrap, which achieved the maximum number (9) of valid estimates, never had any prediction's CI overlapping the manual estimates' CI.

In light of these observations, we cannot reliably say if any of the studied estimators might be useful to developers in practice.

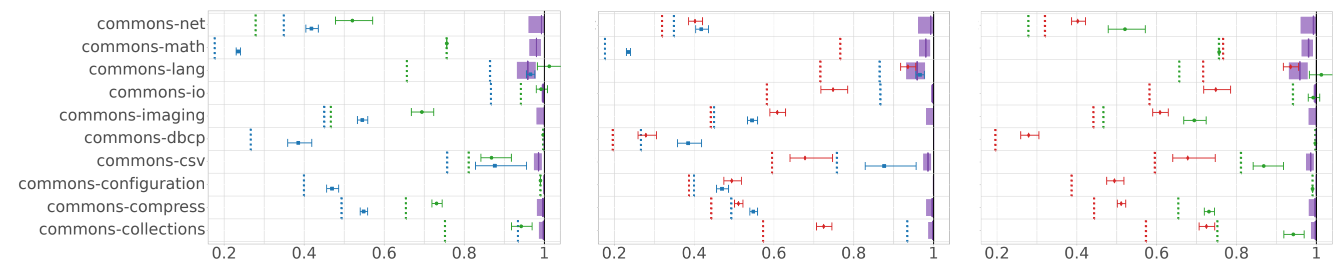
Consequently, we answer (1) as follows:

> Despite some positive results, in general the twelve frequency-based estimators we considered were neither accurate nor reliable in predicting killable mutants from the Original test suites using test method as sampling unit.

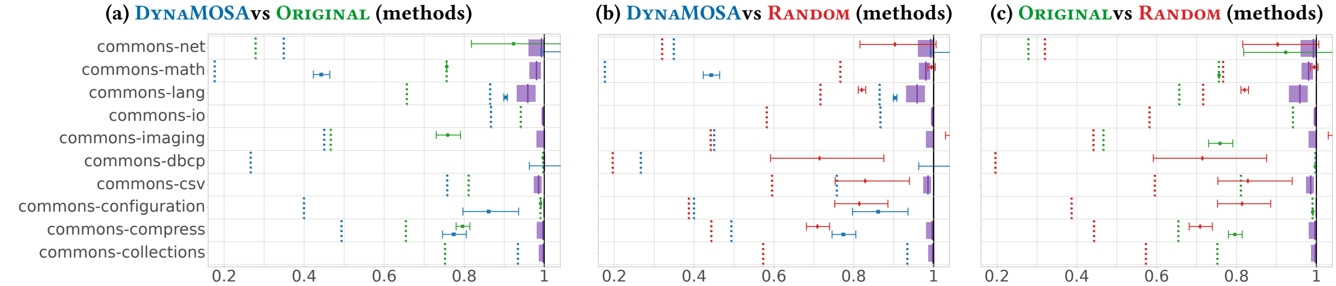


(a) Original (b) Random (c) DynaMOSA

Figure 3: Killable mutants estimated by Chao estimator and manual sampling (i.e., ground truth). Ratio of mutants in x-axis, with 1.0 indicating all generated mutants. The y-axis lists the projects. The method based estimators are in blue, while class based estimators are in red. The purple band is the manual sampling based estimate CI, with dark purple line the point estimate. The dotted black line indicates the total ratio of killed mutants, which is the absolute lower-bound. The figure shows how neither method nor class based estimators overlap with ground truth CI consistently.



(a) DynaMOSAvs Original (methods) (b) DynaMOSAvs Random (methods) (c) OriginalvsRandom (methods)



(d) DynaMOSAvs Original (classes) (e) DynaMOSAvs Random (classes) (f) OriginalvsRandom (classes)

Figure 4: Comparison of estimates from different test suites with methods (Figure 4a...Figure 4c) and classes (Figure 4d...Figure 4f) as sampling units for Chao estimator. The Original is green, DynaMOSA is blue, and Random is red. The figure shows that the estimate CI from none test suites overlaps each other or ground truth consistently.

2 Are frequency-based estimators affected by sampling strategies?

Frequency-based estimators rely on the data obtained by sampling campaigns; thus, it is possible that they may not be robust to bias in sampling, e.g., bias introduced by the developers. To study how different sampling strategies affect the statistical estimators, we generated two test suites automatically, i.e., without manual bias: Random (results in Table 5) and DynaMOSA (results in Table 6).

Comparing the data in tables 4 and 5 reveals that with Random (i) the estimators produced more valid estimates (97 vs. 79); (ii) the number of overlaps between CI from the estimators and manual classification was comparable (12 vs. 13), although not all the estimators behaved consistently; and, (iii) the mean difference degraded considerably (+20% on average).

Comparing the data in tables 4 and 6 reveals that the situation did not improve using DynaMOSA. Although the valid estimates with DynaMOSA are comparable to those with Original (80 vs. 79), the number of CI overlaps was smaller (9 vs. 13) and the mean difference was even larger than Random (+24% on average).

As before, we plot the results for Chao estimator with each sampling strategy in 3b and 3c. We also plot the results for Chao estimator test suite pairs across Figure 4a...Figure 4f along with the manual estimate.

We expected that using automated test generation strategies would mitigate manual bias and hence result in better estimates. Likewise, we expected that using unbiased sampling, i.e., random search, would produce better results than otherwise. Our expectations were met only partially.

We expected that if the estimators were robust to sampling strategies used, the estimates in each pairs would coincide. However, we found significant differences between each pair. Sampling killable mutants using Random, the estimators performed better than with DynaMOSA, confirming that unbiased sampling is beneficial. Moreover, the estimators produced more valid estimates with Random than any other sampling strategy. However, we also observed that with Random the mean difference, i.e., the precision of the estimations, drastically worsened compared to Original and that with DynaMOSA the situation worsened. One possible explanation of these results is that the automatically generated test suites did not adequately sample the population (see statement and branch coverage in Table 2). However, our counterargument is that if the estimators were actually performing as expected, they should have produced larger confidence interval, hence increasing the *accuracy* of the predictions, i.e., the number of CI overlaps, at the expense of their *precision*, i.e., mean difference. But this was not the case; therefore, we answer (2) as follows:

The estimation quality is affected by the sampling strategy. For instance, unbiased sampling improved the reliability of the estimators but reduced their precision. Additionally, no matter the chosen sampling strategy, none of the estimators showed satisfactory results.

(3) Are frequency-based estimators affected by sampling units?

(2) showed that the estimators are affected by the sampling strategy, i.e., the bias in sampling of killable mutants. (3), instead, focuses on the granularity at which sampling happens, i.e., the sampling unit. Using the same sampling strategy but different granularities, i.e., test method and class, we expect estimators to produce comparable results, as they predict the same quantity. Therefore, studying this aspect would let us draw conclusions on the estimators' consistency. Moreover, this study could let us discount the possibility that either the manual classification was incorrect or the (bad) results achieved in (1) were a statistical flukei—which would be the case if the estimators from other test generators coincide.

From Table 7, Table 8, and Table 9, we observe that the estimates from method and class level sampling units are significantly different. Even Bootstrap, which had the highest (50%) overlap for Random, did not produce a significant overlap of CI between method and class estimators for other test suites.

This situation is clearly exemplified in Figure 3 for Chao estimator, in which almost all the CIs for the same test subject and sampling strategy do not overlap.

In light of these results, we answer (3):

The difference between estimates from test method and test class level estimators is statistically significant for every estimator examined. Consequently, none of the twelve estimators was robust against changes in sampling unit.

5.1 Additional Observations

We observe that the plots vary considerably based on the subject in question. While there are several interesting patterns in the data, the data we have is insufficient to explore these fully. Therefore, we will refrain from interpreting these until a larger follow-up study can be conducted.

5.2 What Does This Mean in Practice?

For an estimator to be useful to the developers, it should be able to produce estimates that are accurate and precise. Additionally, the estimators must be reliable, i.e., its ability to produce valid estimates should not strongly depend on the project under analysis.

We note that examination of just 100 mutants is sufficient to produce an estimate within 2% of the true value. Assuming that examining 100 mutants is reasonable, we should expect any equivalent mutant estimator to do better than this value. However, the fact that none of the estimators could consistently produce estimates that are close to the manual estimates suggests that the frequency-based estimators are not yet ready for use by developers.

To verify that our manual analysis was not the cause of an error, we also investigated whether the estimators could produce consistent values when estimating the same quantities but using different sampling units—method and class. We observed that the estimates produced were inconsistent, i.e., only few CIs overlapped. This is a cause for concern and points to violated assumptions in the underlying model, which needs to be investigated further.

While overall, the result of our study is negative, we observe a glimmer of hope. We note that in Table 8 comparing method and class sampling units using Random, Bootstrap produced 50% CI overlaps. Furthermore, the mean difference was 6.79% with a similarly small standard deviation of 2.71%. It is possible that not all the mutants that can be killed manually may be killable by automatic test generators due to technical limitations or deficiencies in the test oracles. Hence, it is possible that Bootstrap estimation is true to the actual value, albeit with a large amount of uncertainty.

Furthermore, unlike DynaMOSA, Random is unguided in test generation, which may be a hint as to the better performance of Bootstrap on Random and points to the need for further study.

Empirical Strategy Used. This paper uses the quantitative research, including systematic collection of data about different mutants and comparing classifications by different experts, and the agreement is computed by Cohen's Kappa.

Data Availability. The replication package [1] contains data about manual classification of mutants, test suites, kill matrices, and scripts to compute and plot estimations.

6 THREATS TO VALIDITY

External Validity. Our study was conducted on a limited number of programs from a specific open-source repository and using a small number of test producers; hence, our findings may not generalize to other projects and test suites produced by other means. To reduce this risk, we selected multiple projects from Apache Commons and used EvoSuite in two exemplary configurations. Apache Commons projects are popular, implement different functionalities, and are comparable to well run industrial projects. EvoSuite, instead, implements standard baselines and well established and effective algorithms that generate test suites with remarkably different features. We use a single run of both the test generator (randomized) on our classes. While multiple runs are required for statistical confidence of the results, we note that our approach is similar to the one adopted by established biometrics studies that draw conclusions from single sampling campaigns.

Internal Validity. Automatically generated test suites did not always achieve high coverage; hence, they can lead to larger uncertainty in the final estimation of equivalent mutants. However, we note that this situation is similar to the one currently faced by software practitioner. Next, our analysis can be subject to bugs, sampling errors, and manual-misclassification of mutants as equivalents that might bias our results. We tried to mitigate this risk by reviewing the code of JUGE and our scripts, cross-checking the results, and use the largest possible subset of classes for which test generation succeeded. It is possible that our manual classification is biased. We tried to mitigate it by cross-checking between three classifiers.

Construct Validity. We are the first to apply frequency-based statistical models to the problem of killable mutants estimation; hence, our mapping of statistical estimators to the mutation testing domain might not to capture important variables. We tried to mitigate this threat by adopting different sampling strategies.

7 RELATED WORK

Mutation analysis is considered a primary way of evaluating test quality [45]; thus, mutation score is usually considered as a test suite adequacy metrics [4, 5, 21, 33]. Unfortunately, equivalent mutants have vexed practitioners from the very beginning [12] and remain an open issue [35] that affects also residual risk estimation.

Studies focusing on estimating killable or equivalent mutants. Papadakis et al. [44] conducted a study for estimating killable mutants with more numerous subjects. We note that this approach requires manual classification, however limited, for residual defect estimation. This may not be feasible in many cases where the testers may not be program experts. Furthermore, evaluating residual risk may be conducted by people who are not involved in either testing or development (such as the end-user). Hence, alternative means of estimating killable mutants is required.

In comparison to the mutant classification performed in Papadakis et al. [44], our study considers larger programs and different test suites, do not employ selective mutation, whose limits have been discussed empirically and theoretically [26, 27], and we employ mechanisms such as conflict identification and resolution, to reduce manual classification error proneness.

Vincenzi et al. [49] proposed estimating the (posterior) probability that specific mutation operators generate equivalent mutants. Marsit et al. [8, 36, 37] proposed using information theory to measure the intrinsic *redundancy* in programs as a proxy for mutants equivalence. Despite their potential benefits and promising initial results, none of those methods has been empirically evaluated yet.

Studies conducting mutant classification. A few previous studies also relied on the manual mutants classification. Acree's study [3] involved two competent software engineers experts in mutation analysis that classified live mutants from four COBOL programs. Similar to our classification procedure, Acree used manually written tests for eliminating a large chuck of mutants; however, differently from us, the classifiers had no exposure to the programs under analysis and focused on small COBOL programs. During his study, Acree documented various misclassifications (avg. 23%), suggesting that even manual classification has errors. We also found misclassifications (see Table 3, *Misclass.* column), but achieved better accuracy (less than 5% misclassifications on avg),[3] arguably because we trained the labelers and followed a structured and systematic classification protocol.

Other studies of note are by Yao et al. [53] on 18 C programs, and Grün et al. [29] (extended by Schuler et al. [46]) on 7 Java programs. Both studies involved a single researcher but classified a different amount of live mutants, $1,194$ Yao et al. and 140 Grün et al. Yao et al.'s study estimated that 77% of all mutants are killable, while Grün et al.'s reported that 45% of the *classified* live mutants are equivalent. Unfortunately, none of those studies reported the misclassification rate. Compared to those studies, our manual classification required (modulo the number of mutants) the same amount of time, but involved twice as many researchers. We also considered real-world, more complex, and arguably more difficult to evaluate, projects, and a representative set of mutation operators [26]. Finally, we studied manually written and automatically generated unit test suites, that covered more mutants and estimating a higher number (generally $> 90\%$) of killable mutants.

We used statistical estimators for predicting killable mutants, others, instead, used them for estimating residual faults. For instance, Böhme [10] argued to use the same species richness estimators we studied for estimating residual defect density, while Nayak [38] and Voas and McGraw [50] modeled faults and residual defects as members of unknown populations and estimated their number via *capture-recapture* methods. Tohma et al. [48], instead, modeled the distribution of observed faults as hyper-geometric distribution to estimate the number of residual defects.

8 CONCLUSIONS AND FUTURE WORK

Accurately estimating the number of killable mutants is crucial for estimating the residual risk and the effectiveness of test generators. While a sound and complete classifier for killable and equivalent mutants is impossible, recent advances in statistical estimation using frequency based estimators gave us hope that one could at least estimate the number of killable mutants. Consequently, we organized the first, large evaluation of these estimators on mutation analysis spanning several projects and multiple sampling strategies.

Unfortunately, the results we achieved show that the considered statistical estimators applied to the killable mutants estimation are not ready for prime time, as they did not produce consistent, accurate, or precise estimates.

Nonetheless, our observations pointed out that it may be possible, with more sophisticated models and more data, to successfully put statistical estimation in use. However, further study is required to investigate this aspect.

[3] This measure does not consider the results of `commons-csv` that we used for training the labelers.

REFERENCES

[1] [n.d.]. https://anonymous.4open.science/r/chaos-replication-848B.

[2] Nicola Accettura, Giovanni Neglia, and Luigi Alfredo Grieco. 2015. The Capture-Recapture approach for population estimation in computer networks. *Computer Networks* 89 (2015), 107–122.

[3] Allen Troy Acree Jr. 1980. *On Mutation*. Ph.D. Dissertation. Georgia Institute of Technology, Atlanta, Georgia. GIT-ICS-80/12.

[4] James H Andrews, Lionel C Briand, and Yvan Labiche. 2005. Is mutation an appropriate tool for testing experiments?. In *Proceedings of the 27th international conference on Software engineering*. 402–411.

[5] James H Andrews, Lionel C Briand, Yvan Labiche, and Akbar Siami Namin. 2006. Using mutation analysis for assessing and comparing testing coverage criteria. *IEEE Transactions on Software Engineering* 32, 8 (2006), 608–624.

[6] Apache Software Foundation. [n.d.]. Apache Commons. http://commons.apache.org/.

[7] Apache Software Foundation. [n.d.]. Apache Maven. https://maven.apache.org/.

[8] A. Ayad, I. Marsit, J. Loh, M. N. Omri, and A. Mili. 2019. Estimating the Number of Equivalent Mutants. In *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 112–121.

[9] Marcel Böhme. 2018. Assurances in Software Testing: A Roadmap. *CoRR* abs/1807.10255 (2018). arXiv:1807.10255 http://arxiv.org/abs/1807.10255

[10] Marcel Böhme. 2018. STADS: Software testing as species discovery. *ACM Transactions on Software Engineering and Methodology* 27, 2 (1 7 2018). https://doi.org/10.1145/3210309

[11] Dankmar Böhning. 2010. Some general comparative points on Chao's and Zelterman's estimators of the population size. *Scandinavian Journal of Statistics* 37, 2 (2010), 221–236.

[12] Timothy A Budd and Dana Angluin. 1982. Two notions of correctness and their relation to testing. *Acta informatica* 18, 1 (1982), 31–45.

[13] K. P. Burnham and W. S. Overton. 1978. Estimation of the Size of a Closed Population when Capture Probabilities vary Among Animals. *Biometrika* 65, 3 (1978), 625–633. http://www.jstor.org/stable/2335915

[14] K. P. Burnham and W. S. Overton. 1979. Robust Estimation of Population Size When Capture Probabilities Vary Among Animals. *Ecology* 60, 5 (1979), 927–936. http://www.jstor.org/stable/1936861

[15] Anne Chao. 1984. Nonparametric estimation of the number of classes in a population. *Scandinavian Journal of statistics* (1984), 265–270.

[16] Anne Chao and John Bunge. 2002. Estimating the number of species in a stochastic abundance model. *Biometrics* 58, 3 (2002), 531–539.

[17] Anne Chao and Chun-Huo Chiu. 2016. Species richness: estimation and comparison. *Wiley StatsRef: statistics reference online* 1 (2016), 26.

[18] Anne Chao, SM Lee, and SL Jeng. 1992. Estimating population size for capture-recapture data when capture probabilities vary by time and individual animal. *Biometrics* (1992), 201–216.

[19] Chun-Huo Chiu, Yi-Ting Wang, Bruno A Walther, and Anne Chao. 2014. An improved nonparametric lower bound of species richness via a modified good–turing frequency formula. *Biometrics* 70, 3 (2014), 671–682.

[20] Henry Coles. [n.d.]. PIT - Real world mutation testing. https://pitest.org.

[21] Murial Daran and Pascale Thévenod-Fosse. 1996. Software error analysis: A real case study involving real faults and mutations. *ACM SIGSOFT Software Engineering Notes* 21, 3 (1996), 158–171.

[22] Xavier Devroey, Alessio Gambi, Juan Pablo Galeotti, René Just, Fitsum Meshesha Kifetew, Annibale Panichella, and Sebastiano Panichella. 2021. JUGE: An Infrastructure for Benchmarking Java Unit Test Generators. *CoRR* abs/2106.07520 (2021). arXiv:2106.07520 https://arxiv.org/abs/2106.07520

[23] Xavier Devroey, Sebastiano Panichella, and Alessio Gambi. 2020. Java Unit Testing Tool Competition: Eighth Round. In *ICSE '20: 42nd International Conference on Software Engineering, Workshops, Seoul, Republic of Korea, 27 June - 19 July, 2020*. ACM, 545–548. https://doi.org/10.1145/3387940.3392265

[24] Gordon Fraser and Andrea Arcuri. 2011. Evosuite: automatic test suite generation for object-oriented software. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. 416–419.

[25] Alessio Gambi, Gunel Jahangirova, Vincenzo Riccio, and Fiorella Zampetti. 2022. SBST Tool Competition 2022. In *15th IEEE/ACM International Workshop on Search-Based Software Testing, SBST@ICSE 2022, Pittsburgh, PA, USA, May 9, 2022*. IEEE, 25–32. https://doi.org/10.1145/3526072.3527538

[26] Rahul Gopinath, Iftekhar Ahmed, Mohammad Amin Alipour, Carlos Jensen, and Alex Groce. 2017. Mutation reduction strategies considered harmful. *IEEE Transactions on Reliability* 66, 3 (2017), 854–874.

[27] Rahul Gopinath, Amin Alipour, Iftekhar Ahmed, Carlos Jensen, and Alex Groce. 2016. On the limits of mutation reduction strategies. In *Proceedings of the 38th International Conference on Software Engineering*. ACM.

[28] Nicholas J Gotelli and Anne Chao. 2013. Measuring and estimating species richness, species diversity, and biotic similarity from sampling data. (2013).

[29] Bernhard JM Grün, David Schuler, and Andreas Zeller. 2009. The impact of equivalent mutants. In *2009 International Conference on Software Testing, Verification, and Validation Workshops*. IEEE, 192–199.

[30] Joseph R Horgan and Aditya P Mathur. 1996. Software testing and reliability. In *Handbook of software reliability engineering*. 531–566.

[31] Joaquín Hortal, Paulo AV Borges, and Clara Gaspar. 2006. Evaluating the performance of species richness estimators: sensitivity to sample grain size. *Journal of animal ecology* 75, 1 (2006), 274–287.

[32] Yue Jia and Mark Harman. 2010. An analysis and survey of the development of mutation testing. *IEEE transactions on software engineering* 37, 5 (2010), 649–678.

[33] René Just, Darioush Jalali, Laura Inozemtseva, Michael D Ernst, Reid Holmes, and Gordon Fraser. 2014. Are mutants a valid substitute for real faults in software testing?. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 654–665.

[34] Shen-Ming Lee and Anne Chao. 1994. Estimating population size via sample coverage for closed capture-recapture models. *Biometrics* (1994), 88–97.

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

[36] Imen Marsit, Mohamed Nazih Omri, JiMing Loh, and Ali Mili. 2018. Impact of Mutation Operators on Mutant Equivalence.. In *ICSOFT*. 55–66.

[37] Imen Marsit, Mohamed Nazih Omri, and Ali Mili. 2017. Estimating the Survival Rate of Mutants. In *ICSOFT*.

[38] Tapan Nayak. 1988. Estimating Population Size by Recapture Sampling. *Biometrika* 75 (03 1988). https://doi.org/10.2307/2336441

[39] James L Norris and Kenneth H Pollock. 1998. Non-parametric MLE for Poisson species abundance models allowing for heterogeneity between species. *Environmental and Ecological Statistics* 5, 4 (1998), 391–402.

[40] A. Jefferson Offutt and W. Michael Craft. 1994. Using compiler optimization techniques to detect equivalent mutants. *Software Testing, Verification and Reliability* 4, 3 (1994), 131–154. https://doi.org/10.1002/stvr.4370040303 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/stvr.4370040303

[41] R Lyman Ott and Micheal T Longnecker. 2015. *An introduction to statistical methods and data analysis*. Cengage Learning.

[42] Annibale Panichella, Fitsum Meshesha Kifetew, and Paolo Tonella. 2015. Reformulating branch coverage as a many-objective optimization problem. In *IEEE International Conference on Software Testing, Verification and Validation*. IEEE, 1–10.

[43] Sebastiano Panichella, Alessio Gambi, Fiorella Zampetti, and Vincenzo Riccio. 2021. SBST Tool Competition 2021. In *14th IEEE/ACM International Workshop on Search-Based Software Testing, SBST 2021, Madrid, Spain, May 31, 2021*. IEEE, 20–27. https://doi.org/10.1109/SBST52555.2021.00011

[44] Mike Papadakis, Marcio Delamaro, and Yves Le Traon. 2014. Mitigating the effects of equivalent mutants with mutant classification strategies. *Science of Computer Programming* 95 (2014), 298–319.

[45] Mike Papadakis, Marinos Kintis, Jie Zhang, Yue Jia, Yves Le Traon, and Mark Harman. 2019. Mutation testing advances: an analysis and survey. In *Advances in Computers*. Vol. 112. Elsevier, 275–378.

[46] David Schuler and Andreas Zeller. 2010. (Un-) covering equivalent mutants. In *2010 Third International Conference on Software Testing, Verification and Validation*. IEEE, 45–54.

[47] Eric P. Smith and Gerald van Belle. 1984. Nonparametric Estimation of Species Richness. *Biometrics* 40, 1 (1984), 119–129. http://www.jstor.org/stable/2530750

[48] Yoshihiro Tohma, Kenshin Tokunaga, Shinji Nagase, and Yukihisa Murata. 1989. Structural approach to the estimation of the number of residual software faults based on the hyper-geometric distribution. *IEEE transactions on software engineering* 15, 3 (1989), 345–355.

[49] Auri Vincenzi, Elisa Nakagawa, José Maldonado, Márcio Delamaro, and Roseli Romero. 2002. Bayesian-Learning Based Guidelines to Determine Equivalent Mutants. *International Journal of Software Engineering and Knowledge Engineering* 12 (12 2002), 675–689. https://doi.org/10.1142/S021819400200113X

[50] Jeffrey M. Voas and Gary McGraw. 1997. *Software Fault Injection: Inoculating Programs against Errors*. John Wiley & Sons, Inc., USA.

[51] Ji-Ping Wang. 2010. Estimating species richness by a Poisson-compound gamma model. *Biometrika* 97, 3 (2010), 727–740.

[52] Ji-Ping Z Wang and Bruce G Lindsay. 2005. A penalized nonparametric maximum likelihood approach to species richness estimation. *J. Amer. Statist. Assoc.* 100, 471 (2005), 942–959.

[53] Xiangjuan Yao, Mark Harman, and Yue Jia. 2014. A study of equivalent and stubborn mutation operators using human analysis of equivalence. In *Proceedings of the 36th International Conference on Software Engineering*. 919–930.