On adaptive sampling-based testing for software reliability assessment

Roberto Pietrantuono, Stefano Russo
DIETI, Università degli Studi di Napoli Federico II, Via Claudio 21, 80125, Napoli, Italy
{roberto.pietrantuono, stefano.russo}@unina.it

***Abstract*—Assessing reliability of software programs during validation is a challenging task for engineers. The assessment is not only required to be unbiased, but it needs to provide tight variance (hence, tight confidence interval) with as few test cases as possible. Statistical sampling is a theoretically sound approach for reliability testing, but it is often impractical in its current form, because of too many test cases required to achieve desired confidence levels, especially when the software has few residual faults inside.**

We claim that the potential of statistical sampling methods is largely underestimated. This paper presents an adaptive sampling- based testing (AST) strategy for reliability assessment. A two- stage conceptual framework is defined, where adaptiveness is included to uncover residual faults earlier, while various sampling-based techniques are proposed to improve the efficiency (in terms of variance-test cases tradeoff) by better exploiting the information available to tester. An empirical study is conducted to assess the AST performance and compare the proposed sampling techniques to each other on real programs.

***Index Terms*—Reliability assessment, sampling, random testing, statistical testing, operational testing, reliability testing, adaptive testing, software testing**

I. Introduction

Knowing the level of reliability of a software program is an as much critical as challenging problem. Claiming an achieved level of reliability that remarkably deviates from the true value can have severe consequence in the context where the software is used. The most common practice to assess software reliability and, most importantly, the confidence we place in the assessment, is by testing [1] [2]. Usually, during the assessment process, tests emulating the real operational usage are conducted (namely, operational testing), and (failure) data are collected to estimate reliability [3]–[5]. The main challenge is to provide an estimate of reliability that is unbiased (hence, its expectation is the true reliability) and efficient (namely, with a minimal variance, that implies *high confidence*).

Statistical sampling methods are a natural way to cope with this problem, as their goal is to design sampling plans tailored for a population to study, and provide estimators with the mentioned properties. Specifically, while unbiasedness (and other basic properties, like consistency and sufficiency [6]) are easier to obtain, the driving principle to select an estimator is its *efficiency* in relation to the number of observations required. However, the current literature on sampling-based software testing proposed very few attempts to go beyond the conventional random or operational testing. The latter ones have been extensively proposed in the past [3], [7]–[9], but they are instances of simple sampling schemes that, even providing unbiased estimates, require a large number of test cases for a desired confidence, especially when few residual faults are in the software (e.g., in critical systems).

This paper proposes a two-stage testing strategy to improve the efficiency of reliability assessment in terms of estimator's variance *vs* required number of test cases. We define a conceptual framework, called **Adaptive Sampling-based Testing** (AST), that includes: *i)* the *adaptive allocation* of test cases to entities (e.g., components, input partitions, modules) at the first stage, which exploits the continuous feedback from test outcomes to direct more tests where is actually needed; *ii) test selection schemes*, among which a tester can choose, that aim at best exploiting the possible knowledge that tester has about the software's input domain characteristics and usage profile, so as to run more efficient test selection plans (and corresponding estimators) than simple random testing. In this paper, we instantiate the framework with one adaptive allocation method, based on importance sampling [10], and four test selection strategies. For the latter, we describe the selection plan, provide the estimators for the failure rate (hence for reliability), their variance and the estimators of such variance, and compare them analytically against currently adopted approach. Finally, an empirical comparison is performed on four real programs, comparing the AST test selection schemes both against each other and against the non-adaptive conventional operational testing. Besides the positive results in this case, the framework is open to several instantiations where the full potential of sampling-based methods can be leveraged to improve software reliability assessment practices.

II. Related Work

Reliability assessment through testing has been usually addressed either by software reliability growth models (SRGMs) during what is called "development testing" [3], or through statistical sampling-based testing at the end of development. In the former case, failure data observed during testing and debugging are used to build (parametric and non-parametric) models predicting the next time to failure, thus failure intensity at the end of testing. In this case, detected faults are removed (i.e., code is changed), and reliability grows during testing: the goal is to figure out when debug testing can be stopped. A plenty of SRGMs exist in the literature, all trying to capture the possible fault detection patterns of a testing process (e.g., [11]–[14]). The criticisms of this approach lie in their numerous

assumptions due to the difficulties in modeling the complex factors involved in a real testing and debugging process [15].

In sampling-based testing, software under test is frozen, i.e., no change introduced in the code during testing, and the goal is to assess reliability and accept/reject the product before release. Test cases are selected randomly either by uniform distribution or by an operational profile (i.e.,, by a distribution depending on the expected usage of functionalities). Many papers referred to the latter as operational testing, and it is a pillar of reliability testing. It was adopted for certification testing in the Cleanroom methodology [4], [8], [16], [17], [9], and in the Software Reliability Engineering Test process [3]. More recent work improved operational testing either in terms of adaptiveness to allocate test cases or of test selection scheme. *Adaptive testing* was proposed by Cai et *al.*, based still on operational profile but foreseeing adaptation in the assignment of test cases to input domains [5], [18], [19]. The authors formulate testing as an adaptive control problem using controlled Markov chains, with the goal of minimizing the variance of reliability estimator. In [1], it is used along with a gradient descent method to the same aim, while in [2], it exploits confidence intervals as driving criterion to select tests adaptively. In terms of test selection, few approaches went beyond the basic simple random sampling with replacement (SRSWR) scheme. In [20], authors propose to estimate reliability by stratified sampling. Cluster analysis is applied to execution profiles to stratify captured operational executions, and then sampling within strata is without replacement, which is known to be more efficient than the with-replacement counterpart. There is no adaptiveness to online test outcomes, though. In a recent PhD proposal [21], (non-adaptive) stratified sampling is still proposed, combined with symbolic execution to stratify profiles. In our previous work [22], we addressed the dualism between operational and debug testing, and come up with a combined approach to improve over conventional operational testing. In [23], we further elaborate on this, and introduce adaptiveness to improve reliability during development testing. The adaptive sampling technique adopted in [23] is also exploited in this work, in which we generalize that former approach to enable multiple test selection techniques for the problem of reliability assessment with code being frozen. Further approaches to assess reliability (and/or its bounds) are available that use failure data and possibly other evidence, based, for instance, on Bayesian approaches or uncertainty quantification [24]–[28], but are outside the scope of this work, as they do not target testing strategies.

AST also includes the mentioned without-replacement version of stratified sampling as one of the available schemes. In addition, to exploit further forms of auxiliary knowledge for more efficient tests selection, three further schemes are implemented. These are combined with an adaptive allocation strategy at the upper stage based on importance sampling [10], an approach successfully used for other problems, rarely for testing (e.g., in [29] for techniques selection, in [30] for mutation testing). In this regard, other than the discussed works by Cai et *al.*, existing works distribute test cases among entities (mostly components or subsystems) based on "static" optimization models (e.g., minimizing a testing cost or time under reliability constraint, and many variants [31]–[35]), not foreseeing any form of adaptation (not coping, of course, with the successive stage of test selection).

III. Adaptive Sampling-based Testing (AST)

Let us denote with S the system under test, with the set of all its inputs denoted as D. Assume the system can be decomposed into M independently testable entities $E_1, \dots E_m$, each one with its own input domain $D_1, \dots D_m$. An entity can be thought in several ways: it can be a subsystem, a component, a software module, or a simply partition in any partition-based testing strategy. Suppose a tester has a budget of T test cases to assess software reliability of S. We make the following usual assumptions about test cases (e.g., [1] [2] [18] [23]):

1) A test case leads to failure or success; we are able to determine when it is successful or not (perfect oracle).
2) The code is not modified during testing (i.e., it is frozen). The assessment is to to find the current status of reliability; code can be modified after the assessment.
3) Test case runs are independent; i.e., all the non-executed test cases are admissible each time. The execution of a test case is not constrained by the execution of some other test case before.
4) The output of a test case is independent of the history of testing; in other words, a failing test case is always such, independently from the previously run test cases.

The objective of AST is to provide an unbiased estimate of reliability R, denoted as $\hat{R}$. A "good" estimator is sought, namely that is *unbiased* and *efficient* (i.e., with *variance as low as possible* given T tests to run).

The two main stages of the AST framework are in Figure 1. The first phase is about **test cases allocation**, where the

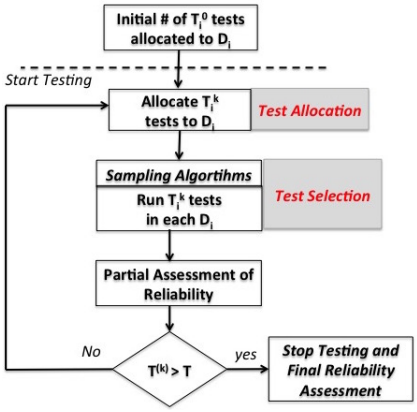


Fig. 1: AST Phases

number of tests for each entity are decided. This could be done by several methods, also depending on what entities are, such as: giving more tests to bigger entities; giving more tests to entities judged as more critical (by domain

experts); using historical data or design information about the expected defectiveness to spot critical entities (e.g., via defect prediction) or to allocate tests by optimization models (e.g., [32] [33]). Whatever the initial allocation is, AST foresees a prominent role of *adaptiveness*, aimed at re-allocating tests to improve reliability estimate as testing proceeds. The process includes a feedback from test results to the allocation step in order to *adjust* the allocation depending on where more tests are actually needed.

The output of the allocation stage is the assignment of a number of test cases to run to each entity E_i, denoted as T_i. The second stage of AST is about **test selection**, where the algorithm(s) decide how to select the T_i test cases from domain D_i to make the estimate more efficient. In Section III-B, several *test selection techniques* are presented, whose applicability are a trade-off between the knowledge that a tester could exploit to improve selection, the technique performance, and its implementation complexity.

All the techniques select test cases based on a more or less in-depth knowledge of the operational profile. A profile P is defined as a probability distribution where each input t has an expected occurrence probability p_t. With respect to knowledge of P, the techniques will generally consider each input either singularly or grouped by classes with similar characteristics (e.g., all inputs of a functionality, inputs of an equivalence class, etc.). To take the more general case, we consider an occurrence probability p_t assigned to each input $t \in D$. Thus, if no information is available at all about expected occurrence of inputs, we have $p_t = 1/|D|$ (i.e., same probability to all inputs). If tester has information at entity level, a p_i value is assigned to the entire domain D_i assuming the within-domain distribution being uniform with $p_t = p_i/|D_i|$. If tester distinguishes between classes of inputs within D_i, then different p_j values are given to each class (and uniform distribution within the class). Knowledge of the profile is assumed to be exact, like in most related literature [5], [18], [1], [19]. We dealt with partial knowledge of the profile in our previous work [23], whose Montecarlo-based approach can seamlessly be integrated in what presented here.

For each domain D_i, we define: $\varphi_i = \theta_i \sum_{t \in D_i} p_t$, where $\sum_{t \in D_i} p_t$ is the probability of selecting an input from D_i, and θ_i is the probability that an input selected from D_i is a failure point. Reliability is: $R = 1 - \Phi = 1 - \sum_{i=1}^{m} \varphi_i$ where Φ is the operational failure probability. The estimate of reliability is computed from domain-level estimates:

$$\hat{R} = 1 - \sum_{i=1}^{m} \hat{\varphi}_i = 1 - \sum_{i=1}^{m} p_i \cdot \hat{\theta}_i \tag{1}$$

where $p_i = \sum_{t \in D_i} p_t$, while $\hat{\theta}_i$ is the sought estimate of the probability that an input selected from D_i is a failure point. The variance of the estimator, being $\hat{\theta}_i$ independent, is:

$$V(\hat{R}) = V(\hat{\Phi}) = \sum_{i=1}^{m} p_i^2 V(\hat{\theta}_i) \tag{2}$$

A. Adaptive allocation of test cases

Adaptiveness aims at periodically re-allocating tests to improve reliability estimate efficiency in terms of variance. It iteratively assigns a subset of total test cases (T) to domains, giving more tests to domains with a bigger expected variance. At iteration $k = 0$, a subset $T^{(0)}$ of tests is distributed to entities. This initial allocation can be done by several alternatives depending on the initial knowledge about domains (e.g., proportional to domains size, via defectiveness- and usage-related historical data, via expert judgement about entities criticality). We assume that no information is available, and perform a size-proportional allocation[1]: $T_i^{(0)} = T^{(0)} \cdot \frac{|D_i|}{|D|}$. At next iterations, test cases are distributed by weighting the number of tests ($T^{(k+1)}$) foreseen for iteration *(k+1)*: $T_i^{(k+1)} = T^{(k+1)} \omega_i^{(k)}$. In the following, we describe the method implemented to determine $\omega_i^{(k)}$ and $T^{(k+1)}$. A simple solution is to keep on allocating tests proportionally to domains size, hence $\omega_i^{(k)} = \frac{|D_i|}{|D|}$. However, as the goal is to minimize the estimate's variance, allocation needs to be proportional not only to size, but also to variance. Assuming the costs to select a test case across domains approximately equal, it can be shown that the optimal allocation scheme is the *Neyman* allocation [38], where weights are proportional to size and standard deviation:

$$T_i^{(k+1)} = T^{(k+1)} \cdot \omega_i^{(k)} = T^{(k)} \frac{|D_i| \sqrt{V(\theta_i)^{(k)}} \cdot p_i}{\sum_{j=1}^{m} |D_j| \sqrt{V(\theta_j)^{(k)}} \cdot p_j} = \tag{3}$$

However, the true within-domain variances of θ_i are unknown. Thus, at each iteration, the estimates of $V(\theta_i)$ have to be provided by the test selection scheme adopted at domain-level (discussed in the next Section). To implement a robust adaptation with respect to fluctuations of such variance estimates, AST does not directly use Equation 3: it implements an *adaptive importance sampling* (AIS) algorithm to this aim.

Importance sampling aims at approximating the *true* distribution of a variable of interest [10]. Our true unknown distribution is the best number of test cases for each domain that minimizes the variance of reliability estimator. The algorithm represents the beliefs (i.e., hypotheses) about this distribution by means of sets of "samples". Each sample is associated with a probability that the belief is true: at each iteration, these probabilities are updated by examining some new samples of that hypothesis, and a larger number of samples (i.e., test cases) are drawn from hypotheses with a larger probability. The goal is to converge, in few iterations, to the "true" best distribution of test cases.

To establish how the probability of each hypothesis is updated based on new collected samples, an update rule is defined. Let us denote with $\pi^{(k)}$ the probability vector representing, for each domain, the likelihoods that *testing from that domain contributes to minimizing the variance of the estimator*. This information is well represented by weights $\omega_i^{(k)}$. Using the variance estimates of θ_i in lieu of true

[1] In adaptive allocation, the number of samples at the first iteration ($T^{(0)}$) is only required to be *much* smaller than T [30], in order to start the algorithm

(unknown) variances in Equation 3, the update rule of the probability vector $\pi^{(k)}$ is defined as follows:

$$\pi_i^{(k)} = \gamma\pi_i^{(k-1)} + (1-\gamma)\cdot(1-\pi_i^{(k-1)})\cdot\hat{\omega}_i^{(k)} \quad (4)$$

The rule tends to assign progressively more tests to domains with higher variance of the estimator, so as to diminish its impact on the overall variance. Given the same weights $\omega_i^{(k)}$, the increase is larger for domains that had fewer resources at the previous iteration. The smoothness of adaptiveness is further is determined by $\gamma \in [0,1]$, regulating how the algorithm considers past iterations' results with respect to current ones. The $\pi_i^{(k)}$ values are normalized, since they are probabilities ($\pi_i^{(k)} = (\pi_i^{(k)})/(\sum_{i\in D}\pi_i^{(k)})$). Starting from $\pi_i^{(k)}$, the bucket-filling procedure reported below is used to distribute the tests to domains, so as $T_i^{(k+1)} \approx T_i^{(k)}\pi_i^{(k)} \propto T_i^{(k)}\hat{\omega}_i^{(k)}$.

To determine the proper $T_i^{(k)}$ at each iteration, we consider the *adaptive* implementation of importance sampling [10]. Based on a desired error and confidence, this variant tends to progressively reduce the number of required samples as more information becomes available, so as to approximate the sought distribution earlier. Accordingly:

$$T^{(k+1)} = \frac{1}{2\xi}\chi^2_{\rho-1,1-\delta} \approx \frac{\rho-1}{2\xi}\{1-\frac{2}{9(\rho-1)}+\sqrt{\frac{2}{9(\rho-1)}}z_{1-\delta}\}^3 \quad (5)$$

where: ξ is the error that we want to tolerate between the sampling-based estimate and the true distribution; $1-\delta$ is the confidence we want in this approximation; ρ is the number of domains from which at least one test case has been drawn in the k-th iteration; $z_{1-\delta}$ is the normal distribution evaluated with significance level δ. These $T^{(k+1)}$ are distributed to domains according to $\pi_i^{(k)}$ vector by the following procedure.

AIS Procedure

The importance sampling procedure. **Inputs**: D_i, $\pi_i^{(k)}$: $i \in [1,m]$
//sort such that $\pi_i^k \geq \pi_{i+1}^{(k)}$
$b_1 = \pi_1^{(k)}$; *//Initialize Cumulative Distribution*
for i=1 to m $T_i^{(k+1)}$=0; **end for** *//initialization*
//Compute Cumulative Distribution
for i=2 to m $b_i = b_{i-1} + \pi_i^{(k)}$; **end for**
//Compute $T^{(k+1)}$ *according to Eq. 5*
$r_1 \sim U[0, \frac{1}{T^{(k+1)}}]$ *//Draw sample from uniform distribution*
i = 1; *//Distribute test cases to each domain*
for j = 1 **to** $T^{(k+1)}$
  while $r_j > b_i$ **do** $i = i+1$; *//Find the bucket to fill*
  end while
  $T_i^{(k+1)}$=$T_i^{(k+1)}$ + 1; *//Fill the bucket*
  r_{j+1}= $r_j + \frac{1}{\eta^{(k+1)}}$
end for
//Return re-ordered $\{T_i^{(k+1)}\} : i \in [1,m]$

The $T^{(k+1)}$ tests are distributed to domains proportionally to their relative importance. The resulting number of $T_i^{(k+1)}$ test cases are run within each domain according to one of the techniques described in the next Section: test results are in turn used to estimate the variances $V(\theta_i)$, hence allowing to update $\hat{\omega}_i$ (and π_i) based on the new information.

B. Selection of test cases

We describe test selection techniques within domain D_i by providing formulas to compute the failure rate estimator $\hat{\theta}_i$ (needed in Equation 1), its variance $V(\hat{\theta}_i)$, and a correct estimator of such variance $\hat{V}(\hat{\theta}_i)$ (needed in Equation 2 as well as in Equation 3 in lieu of the unknown $V(\hat{\theta}_i)$). The following description starts with the simpler case where simple random sampling is exploited to select tests, and then proceeds by refining the sampling scheme to better exploit available information for efficiency improvement. Hence, the below techniques require increasing pieces of information about the program under test, and this is a possible additional criterion to choose between them, besides efficiency and bias. All the steps described in the following refer to a given iteration k; we omit the superscript k in all the Equations for readability of formulas (e.g., T_i^k is T_i). Also, we denote: $|D_i| = N_i$.

1) SRSWR-based testing: This first technique makes no assumption about *(i)* which input or class of inputs (e.g., equivalence class) is more prone to fail within a domain D_i; *ii)* what is the expected operational usage of (class of) inputs/functionalities. Tester just has information at entity level, namely, p_i value is assigned to the entire domain D_i assuming the within-domain distribution being uniform, i.e., for each input t: $p_t = p_i/N_i$. The simplest form, which is the common one in the existing literature (e.g., [18], [2], [5], [19], [1]), allows the same input t to be selected more times, i.e., a *simple random sampling with replacement* (SRSWR) scheme. Test outcomes are a series of independent Bernoulli random variables $z_{i,t}$ such that $z_{i,t} = 1$ if the execution leads to a failure, $z_{i,t} = 0$ otherwise. Probability that $z_{i,t} = 1$ corresponds to proportion: $\theta_i = \frac{\sum_{t=1}^{N_i} z_{i,t}}{N_i}$. An unbiased estimator of θ_i is the observed proportion of failure points over the number of trials T_i:

$$\hat{\theta}_{i_{SRSWR}} = \frac{\sum_{t=1}^{T_i} z_{i,t}}{T_i}. \quad (6)$$

Accordingly, having assumed independent variables, the variance of the θ estimator is:

$$V(\hat{\theta}_{i_{SRSWR}}) = \frac{\theta_i(1-\theta_i)}{T_i} \quad (7)$$

being the numerator of Eq. 6 a binomial random variable. An unbiased estimator of $V(\hat{\theta}_{i_{SRSWR}})$ (i.e., such that $E[\hat{V}] = V$) is:

$$\hat{V}(\hat{\theta}_{i_{SRSWR}}) = \frac{\hat{\theta}_i(1-\hat{\theta}_i)}{T_i - 1} \quad (8)$$

using the Bessel-corrected version as unbiased estimator of a sample variance V: $\hat{V} = \frac{n}{(n-1)}V$ (n being the sample size). Although SRSWR keeps the mathematical treatment relatively simple, it is unable to exploit additional information a tester might have. New techniques are now introduced that try to improve the efficiency in terms of variance.

2) SRSWOR-based testing: This technique still makes no assumption about knowing failure proneness of (classes of) inputs/functionalities or their operational profile. Differently from the previous one, this technique uses a sampling *without*

replacement (SRSWOR), namely, the same test case is not selected twice. This technique is expected to be more efficient in terms of estimator's variance, as it avoids sampling an input twice. The proportion estimator is still obtained as ratio of observed failure points over tests executed:

$$\hat{\theta}_{i_{SRSWOR}} = \frac{\sum_{t=1}^{T_i} z_{i,t}}{T_i} = p_i \cdot \hat{\theta}_i \tag{9}$$

Variance of the estimator, $\hat{\theta}$, is different. Being a without-replacement scheme, the population units from which to sample are less and less. Thus, observations are not really independent. At the first draw, a test case t of T_i tests to run is drawn out of N_i units; at the second draw, another test case from the remaining $T_i - 1$ is drawn from a population of $N_i - 1$ units, and so on. Defining a random variable $\pi_t = 1$ if unit i is in the sample, $\pi_t = 0$ otherwise, $\hat{\theta}_i$ can be expressed as $\sum_{t=1}^{T_i} \pi_t \frac{z_{i,t}}{T_i}$. Since π_t are 0/1 variables, $E[\pi_t] = E[\pi_t^2] = T_i/N_i$, and $V(\pi_t) = E[\pi_t^2] - E[\pi_t]^2 = \frac{T_i}{N_i}(1 - \frac{T_i}{N_i})$. Moreover: $E[\pi_t \pi_{t'}] = P(\pi_{t'} = 1 | \pi_t = 1) P(\pi_t = 1) = (\frac{T_i - 1}{N_i - 1})(\frac{T_i}{N_i})$ – namely, if we know that test t is in the sample, we do have a small amount of information about whether test t' is in the sample, reflected in the conditional probability $P(\pi_{t'} = 1 | \pi_t = 1)$. Thus covariance is not null and: $Cov(\pi_t, \pi_{t'}) = E[\pi_t \pi_{t'}] - E[\pi_t] E[\pi_{t'}] = -\frac{1}{N_i - 1}(1 - \frac{T_i}{N_i})(\frac{T_i}{N_i})$. Given these preliminaries, and using properties of covariance:

$$\begin{aligned} &V(\hat{\theta}_{i_{SRSWOR}}) = \tfrac{1}{T_i^2} V(\textstyle\sum_{t=1}^{N_i} \pi_t z_{i,t}) = \\ &\tfrac{1}{T_i^2} \textstyle\sum_{t=1}^{N_i} \sum_{t'=1}^{N_i} z_{i,t}, z_{i,t'} Cov(\pi_t \pi_{t'})) = \\ &\tfrac{1}{T_i^2} [\textstyle\sum_{t=1}^{N_i} z_{i,t}^2 V(\pi_t) + \sum_{t=1}^{N_i} \sum_{t' \neq t}^{N_i} z_{i,t}, z_{i,t'} Cov(\pi_t \pi_{t'})] \end{aligned} \tag{10}$$

Using variance and covariance of π_t, $\pi_{t'}$ and taking out of the summation:

$$\begin{aligned} &V(\hat{\theta}_{i_{SRSWOR}}) = \\ &\tfrac{1}{T_i^2} \tfrac{T_i}{N_i}(1 - \tfrac{T_i}{N_i})[\textstyle\sum_{t=1}^{N_i} z_{i,t}^2 - \tfrac{1}{N_i - 1} \sum_{t=1}^{N_i} \sum_{t' \neq t}^{N_i} z_{i,t}, z_{i,t'}] = \\ &\tfrac{1}{T_i} \tfrac{T_i}{N_i}(1 - \tfrac{T_i}{N_i})(\tfrac{1}{N_i(N_i - 1)})[N_i \textstyle\sum_{t=1}^{N_i} z_{i,t}^2 - (\sum_{t=1}^{N_i} z_{i,t})^2] = \\ &\tfrac{N_i - T_i}{N_i} \tfrac{N_i}{N_i - 1} \tfrac{\theta_i(1-\theta_i)}{T_i} = \tfrac{N_i - T_i}{N_i - 1} \tfrac{\theta_i(1-\theta_i)}{T_i} \end{aligned} \tag{11}$$

Hence, with respect to the SRSWR case, variance is modified by adding what is called the *finite population correction* factor $\frac{(N_i - T_i)}{N_i}$, accounting for the fact that the population is finite, and using the $\frac{N_i}{N_i - 1}$ factor to make it unbiased.

An unbiased estimator of $V(\hat{\theta}_{i_{SRSWOR}})$ is:

$$\hat{V}(\hat{\theta}_{i_{SRSWOR}}) = \frac{N_i - T_i}{N_i} \frac{\hat{\theta}_i(1 - \hat{\theta}_i)}{T_i - 1} \tag{12}$$

since:

$$\begin{aligned} &E[\tfrac{N_i - T_i}{N_i} \tfrac{\hat{\theta}_i(1-\hat{\theta}_i)}{T_i - 1}] = \tfrac{N_i - T_i}{N_i} E[\tfrac{\hat{\theta}_i(1-\hat{\theta}_i)}{T_i - 1} \tfrac{T_i}{T_i}] = \\ &\tfrac{N_i - T_i}{N_i} \tfrac{\theta_i(1-\theta_i) N_i}{N_i - 1} \tfrac{1}{T_i} = \tfrac{N_i - T_i}{N_i - 1} \tfrac{\theta_i(1-\theta_i)}{T_i} \end{aligned} \tag{13}$$

using the fact that $\frac{\hat{\theta}_i(1-\hat{\theta}_i) T_i}{T_i - 1}$ unbiasedly estimates $\frac{\theta_i(1-\theta_i) N_i}{N_i - 1}$.

Assuming $T_i \geq 1$, SRSWOR is expected to be more efficient than SRSWR, since its variance is expected to be lower:

$$\frac{V(\hat{\theta}_{i_{SRSWR}})}{V(\hat{\theta}_{i_{SRSWOR}})} = \frac{N_i - 1}{N_i - T_i} \geq 1 \tag{14}$$

Since both SRSWR- and SRSWOR-based testing make the same assumptions about the knowledge available to tester, the latter is preferred: we use SRSWOR in the following for efficiency comparison, neglecting the SRSWR case.

3) Stratified SRS testing: The above two strategies can be improved if a tester has knowledge about which classes of inputs within D_i are expected to have a common behaviour. Tester, as a matter of fact, often uses partitioning to try reducing the number of useless tests. There are several ways in which s/he can partition an input domain, provided that test cases in a partition have some properties in common (e.g., based on functional, structural, or profile criteria). Regardless partitioning criteria, we denote as $C_{i,h}$ the h-th class within domain i (it can group inputs of a functionality, of an equivalence class, of a "choice" in category-partition testing, and so on), and $N_{i,h}$ the number of elements within $C_{i,h}$.

If such information is available, the stratified sampling (S-SRS) technique can be used to instead of SRSWOR and SRSWR, exploiting the principle of *stratified sampling*. In S-SRS testing, the proportion of failure points is estimated by combining the proportions obtained in each class:

$$\hat{\theta}_{i_{S-SRS}} = \frac{1}{N_i} \sum_{h=1}^{M_i} N_{i,h} \hat{\theta}_{i,h} \tag{15}$$

where M_i is the number of classes and $\hat{\theta}_{i,h}$ the estimate obtained by Equation 9 for each class. Since the selection from classes is independent, variance of the estimator is the linear combination of within-class variances:

$$V(\hat{\theta}_{i_{S-SRS}}) = \frac{1}{N_i^2} \sum_{h=1}^{M_i} N_{i,h}^2 V(\hat{\theta}_{i,h_{SRSWOR}}) \tag{16}$$

Similarly, its unbiased estimator is:

$$\hat{V}(\hat{\theta}_{i_{S-SRS}}) = \frac{1}{N_i^2} \sum_{h=1}^{M_i} N_{i,h}^2 \hat{V}(\hat{\theta}_{i,h_{SRSWOR}}) \tag{17}$$

using Equation 11 and Equation 12 in the two cases.

A task required by S-SRS is the assignment of test cases to classes. This is the same problem we faced at domain-level, and assume, without loss of generality, the same solution here: a "proportional allocation" in the first stage (i.e., $T_{i,h} = \frac{N_{i,h}}{N_i} T_i$), and "optimal Neyman allocation" (Equation 3) in the next stages when an estimate of variances becomes available.

Efficiency with respect to SRSWOR is expected to improve. Considering the first iteration (namely, under proportional allocation, being conservative), let us compare variances of S-SRS and SRSWOR. We set $v_i = \theta_i(1-\theta_i)$ and $v_{i,h} = \theta_{i,h}(1-\theta_{i,h})$.

Since in proportional allocation $\frac{N_{i,h}-T_{i,h}}{T_{i,h}} = \frac{N_i-T_i}{T_i}$, we have:

$$\frac{V(\hat{\theta}_{i_{S-SRS}})}{V(\hat{\theta}_{i_{SRSWOR}})} = \frac{\frac{1}{N_i^2}\sum_{h=1}^{M_i} N_{i,h}^2 \left(\frac{N_{i,h}-T_{i,h}}{N_{i,h}-1}\frac{v_{i,h}}{T_{i,h}}\right)}{\frac{N_i-T_i}{N_i-1}\frac{v_i}{T_i}} =$$

$$\frac{\frac{1}{N_i^2}\frac{N_i-T_i}{T_i}\sum_{h=1}^{M_i} N_{i,h}^2 \frac{v_{i,h}}{N_{i,h}-1}}{\frac{N_i-T_i}{N_i-1}\frac{v_i}{T_i}} = \frac{\frac{1}{N_i}\sum_{h=1}^{M_i}\frac{N_{i,h}}{N_{i,h}-1}N_{i,h}v_{i,h}}{\frac{N_i}{N_i-1}v_i} \tag{18}$$

For $N_{i,h}$ sufficiently large, the ratio $\frac{N_{i,h}}{N_{i,h}-1} \to 1$; the numerator corresponds to *within-class* variance and the denominator is the total variance (i.e., *within-class* plus *between-classes* variance, namely $v_i = \frac{1}{N_i}\sum_h N_{i,h}[v_{i,h} + (\theta_{i,h} - \theta_i)^2]$). Thus, the ratio is less than 1, unless class means are all equal[2].

S-SRS can be applied to any partition testing strategy, e.g., black- or white-box partitioning or category-partition testing.

4) PPS-based testing: Besides information that allows partitioning of D_i, let us assume to have an estimate of the operational profile at class-level, along with some auxiliary indication about the failure proneness of a class with respect to the others. The latter should be a driving principle of partitioning, wherein classes of inputs are separated with respect to their supposed failing behaviour. There are several methods to support the tester's intuition with quantitative figures about which functionality or class of inputs is more likely to fail, especially considering that reliability assessment is done at the end of the development process, and much information is available. For instance, the amount of testing, inspection or, generally, quality assurance activities that a functionality received or the achieved code coverage suggest where a high effort was devoted to assure few residual faults; the functionalities' code characteristics, such as size or complexity metrics, are often used as predictor for defect proneness by machine learning [39]; historical failure data, domain expert opinion, and other evidences can be used for such an assessment, as mentioned in Section II. These all can contribute to have a relative assessment of classes with higher expected failure rate[3]. However is assessed, we call it failure likelihood, denoted as $\vartheta \in [0, 1]$. The two techniques explained in this Section just assume a rough proportionality of the auxiliary information ϑ with the true (unknown) failure rate. Note that this knowledge is just supposed to be better than knowing nothing about the relative difference among failure rates.

In such a scenario, we change the problem formulation. Let us consider the quantity to estimate being not the proportion of failure points, but the total: $\varphi_i = \sum_h p_{i,h}\theta_{i,h} = \sum_{h=1}^{M_i}\frac{p_{i,h}}{N_{i,h}}\sum_{t\in h} z_{i,t} = \sum_{t\in D_i} p_t z_{i,t}$, where $p_{i,h}$ is the probability of selecting an input from class $C_{i,h}$, and: $p_t = p_{i,h}/N_{i,h}$, because of equal selection probability within classes[4]. We define the auxiliary variable x associated with each input t such that: $x_{i,t} = p_t\vartheta_{i,h}$ where $\vartheta_{i,h}$ is the failure likelihood of the class The corresponding probability of selection of each input point t as test case is: $\pi_t = \frac{x_{i,t}}{\sum_t x_{i,t}}$. This is called *proportional to size* (PPS) selection [38], where the "size" is the variable x. If no knowledge about failure likelihood is available, the method still works, but the higher the correlation between x and φ_i the higher the efficiency.

Given this general scheme, selection of test cases can be done, again, with or without replacement. Since $N_{i,h}$ and $p_{i,h}$ values are known, we need to estimate the total number of failure point $Z_i = \sum_t z_{i,t}$ to get $\hat{\theta}_i$ and $\hat{\varphi}_i$. In case of with-replacement selection, the estimator is the sample mean of observed values rescaled by the inverse of their selection probability π_t, namely: $\hat{Z}_i = \frac{1}{T_i}\sum_{t=1}^{T_i}\frac{z_{i,t}}{\pi_t}$, known as the Hansen-Hurwitz estimator. Variance is:

$$V(\hat{Z}_i) = E[(\hat{Z}_i - Z_i)^2] = \frac{1}{T_i}[\sum_{t=1}^{N_i}\pi_t(\frac{z_{i,t}}{\pi_t} - Z_i)^2] = \frac{1}{T_i}\left(\sum_{t=1}^{N_i}\frac{z_{i,t}^2}{\pi_t} - Z_i^2\right) \tag{19}$$

With respect to the simple random sampling counterpart (SRSWR), this is a generalization, since in SRSWR π_t are equal to $1/N_i$[5]. If we consider the corresponding without-replacement case (namely, PPS sampling without replacement), we expect to obtain better variance than Equation 19. Hence, we now consider two techniques, the RHC and the SDE schemes, to estimate Z_i.

PPS-RHC technique

This uses the Rao, Hartley and Cochran (RHC) sampling for selecting tests according to PPS [40]. It acts as follows:

1) Given the T_i test cases to execute in D_i, divide randomly the N_i units of the population into $g = T_i$ groups, by selecting G_1 inputs with a SRSWOR for the first group, then G_2 inputs out of the remaining $(N_i - G_1)$ for the second, and so on. This will lead to g groups of size $G_1, G_2, \ldots, G_g$ with $\sum_{r=1}^{g} G_r = N_i$. The group size is arbitrary, but we select $G_1 = G_2 = \cdots = G_g = N_i/T_i$, as this minimizes the variance [40].
2) One test case is then drawn by taking an input t in each of these g groups independently and with a probability proportional to size – in our case, according to π_t values.
3) Denote with $\pi_{t,r}$ the probability associated with the t-th unit in the r-th group, and with $q_r = \sum_{t\in G_r}\pi_{t,r}$ the sum in the r-th group. An unbiased estimator of Z_i is:

$$\hat{Z}_i = \sum_{r=1}^{g}\frac{\pi_t z_{i,t}}{\pi_r/q_r} \tag{20}$$

[2]Such a case means that partitioning is done so badly that it has no impact.

[3]Failure rate of a class is meant as probability of failing given that an input is selected from that class; the actual failure probability depends, of course, not only on the faults within the class, but also on the probability of selecting an input from that class in operation, namely on the operational profile. Thus, this information is later combined with the class-level operational profile

[4]Note that unequal probability of selection could be seamlessly used in the method formulation, but the information on the operational profile is rarely available at such fine level of granularity.

[5]Note that, the case of proportions θ of Equation 7 for SRSWR is similar, since $\theta(1-\theta) = \theta - \theta^2 = \sum_t z_{i,t}/N_i - \sum_t z_{i,t}^2/N_i^2 = \sum_t z_{i,t}^2/N_i - \sum_t z_{i,t}^2/N_i^2$, since $z_{i,t} = z_{i,t}^2$ being $z_{i,t}$ a dihcotomic (0/1) variable. Since proportions are "means" of the variable $z_{i,t}$, while here we have a total, Equation 7 multiplied by N_i^2 yields the variance of the total's estimator $\hat{Z}_i$ that is the same as Equation 19 with $\pi_t = 1/N_i$

with $z_{i,t} = 1$ if t is a failure point, 0 otherwise. The suffixes $1, 2, \ldots, r$ denote the g test cases selected from the g groups separately. This leads to: $\hat{\theta}_{i_{RHC}} = \frac{\hat{Z}_i}{N_i}$, which is the sought proportion of failure points.

The estimator is unbiased since E[$\hat{Z}_i$] = $E_1E_2[\hat{Z}_i]$ = $E_1[Z_i]$ = Z_i, where E_2 is the expectation for a given split and E_1 the expectation over all possible splits into T_i groups of the chosen sizes. Variance of $\hat{Z}_i$ is derived by observing that, under unbiasedness, $V(\hat{Z}_i) = E_1V_2(\hat{Z}_i)$, where V_2 is the variance within a split:

$$V(\hat{Z}_{i_{RHC}}) = \frac{\sum_r G_r^2 - N_i}{N_i(N_i-1)} \left(\sum_{t=1}^{N_i} \frac{z_{i,t}^2}{\pi_t} - Z_i^2 \right) \tag{21}$$

with $\sum_r$ denoting the sum over the $g = T_i$ groups. Its unbiased estimator is derived in [40] is:

$$\hat{V}(\hat{Z}_{i_{RHC}}) = \frac{\sum_r G_r^2 - N_i}{N_i^2 - \sum_r G_r^2} \left(\sum_{r=1}^{g} q_r (\frac{z_{i,r}}{\pi_r} - \hat{Z}_i)^2 \right). \tag{22}$$

Choosing $G_1 = G_2 = \cdots = G_g = N_i/T_i$, we have:

$$\frac{\sum_r G_r^2 - N_i}{N_i(N_i-1)} = \frac{T_i(N_i/T_i)^2 - N_i}{N_i(N_i-1)} = \frac{1}{T_i} \frac{(N_i-T_i)}{(N_i-1)} \tag{23}$$

Hence:

$$V(\hat{Z}_{i_{RHC}}) = \frac{1}{T_i} \frac{(N_i - T_i)}{(N_i - 1)} \left(\sum_{t=1}^{N_i} \frac{z_{i,t}^2}{\pi_t} - Z_i^2 \right) \tag{24}$$

which clearly less than the with-replacement case in Equation 19. Thus the without-replacement case is better, in terms of efficiency, than the with-replacement case by a factor $\frac{(N_i-T_i)}{(N_i-1)}$. The sought variance of $\hat{\theta}_{i_{RHC}}$ and its estimator are:

$$V(\hat{\theta}_{i_{RHC}}) = \frac{V(\hat{Z}_i)}{N_i^2} \qquad \hat{V}(\hat{\theta}_{i_{RHC}}) = \frac{\hat{V}(\hat{Z}_i)}{N_i^2} \tag{25}$$

Let us compare RHC against the SRSWOR case (denoted, for brevity, SRS). From Equation 11, writing $\theta_i = \frac{\sum_{t=1}^{N_i} z_{i,t}}{N_i}$ and recalling that $z_{i,t} = z_{i,t}^2$, being $z_{i,t}$ a 0/1 variable), we have that:

$$V(\hat{Z}_{i_{SRS}}) = N^2 V(\hat{\theta}_{i_{SRS}}) = \frac{1}{T_i} \frac{(N_i - T_i)}{(N-1)} \left(\sum_t N_i z_{i,t}^2 - Z_i^2 \right) \tag{26}$$

Therefore, RHC (Equation 24) is more efficient if this condition is verified:

$$\sum_{t=1}^{N_i} \frac{z_{i,t}^2}{\pi_t} < \sum_{t=1}^{N_i} N_i z_{i,t}^2 \tag{27}$$

Considering that $\pi_t = \frac{x_{i,t}}{\sum_t x_{i,t}} = \frac{x_{i,t}}{X_i} = \frac{x_{i,t}}{\bar{X}_i N_i}$, and $Z_i = \bar{Z}_i N_i$ ($\bar{X}$ and $\bar{Z}$ are the population means), the RHC variance becomes:

$$V(\hat{Z}_{i_{RHC}}) = \frac{(N_i - T_i)}{(N_i - 1)} \bar{X}_i \frac{N_i}{T_i} \sum_{t=1}^{N_i} \frac{1}{x_{i,t}} (z_{i,t} - \frac{\bar{Z}_i}{\bar{X}_i} x_{i,t})^2 \tag{28}$$

Expanding the expression and recalling that $Cov(X, \frac{Z^2}{X}) = E[X, \frac{Z^2}{X}] - E[\frac{Z^2}{X}]E[X]$, condition in Equation 27 is verified if and only if $Cov(X, \frac{Z^2}{X}) > 0$. But in PPS sampling, X is supposed to be roughly proportional to Z, thus their covariance should be at least positive. RHC turns out to be worse than SRSWR only in the case that auxiliary information is negatively correlated with the variable to estimate, which is a worse situation than a complete absence of knowledge about more or less failure-prone classes (i.e., knowledge is even misleading). In practice, an even partial knowledge (e.g., inputs from boundary-value regions more likely to fail than others) can suffice to distinguish more failure-prone classes; without such knowledge, partition testing is not convenient from the assessment point of view.

PPS-SDE technique

This technique still uses a PPS without replacement like RHC. It works in this way: on the first draw a unit t_1 is chosen with probability π_1; on the second draw a unit t_2 ($\neq t_1$) is chosen with probability $\pi_2/(1-\pi_1)$ leaving t_1 aside; on the third draw, $t_3(\neq t_1, t_2)$ is chosen with probability $\pi_3/(1-\pi_1-\pi_2)$, and so on. On the final *n*th ($n = T_i > 2$) draw, a unit in ($\neq t_1$,...,t_{n-1}) is chosen with probability:

$$\frac{\pi_n}{1 - \pi_1 - \pi_2 -, \ldots, -\pi_{n-1}} \tag{29}$$

It follows that:

$$\begin{aligned} e_1 &= z_1/\pi_1 \\ e_2 &= z_1 + \tfrac{z_2}{\pi_2}(1-\pi_1) \\ e_j &= z_1 + \cdots + z_{j-1} + \tfrac{z_j}{\pi_j}(1 - \pi_1 - \cdots - \pi_{j-1}) \end{aligned} \tag{30}$$

with $j = 3 \ldots, T_i$, are all unbiased estimators for Z_i. In fact, the conditional expectation: $E_c[e_j|(t_1, z_1), \ldots, (t_{j-1}, z_{j-1})]$ = $(z_1 + \cdots + z_{j-1}) + \sum_{j=1 \neq t_1, \ldots, t_{j-1}}^{N} \frac{z_k}{\pi_j/(1-\pi_j)} \frac{\pi_j}{(1-\pi_j)} = Z_i$.

The overall expectation $E(E_c(e_j)) = E(Z_i) = Z_i$, $\forall j = 1, \ldots, T_i$. So, unconditionally, $E(e_j) = Z_i$, and the *Des Raj* estimator is [38]:

$$\hat{Z}_{i_{RAJ}} = \frac{1}{T_i} \sum_{j=1}^{T_i} e_j \tag{31}$$

Notice that e_j, e_k ($j < k$) are pairwise uncorrelated; so, the variance of $\hat{Z}_i$ and its unbiased estimator are:

$$V(\hat{Z}_{i_{RAJ}}) = \frac{1}{T_i^2} \sum_{j=1}^{T_i} V(e_j) \qquad \hat{V}(\hat{Z}_{i_{RAJ}}) = \frac{1}{T_i(T_i-1)} \sum_{j=1}^{T_i} (e_j - \hat{Z}_i)^2 \tag{32}$$

Clearly, $\hat{Z}_{i_{RAJ}}$ depends on the order in which the units are drawn in the sample s. So, we apply the Murthy's unordering to get the a better variance, by averaging the ordered $\hat{Z}_{i_{RAJ}}$:

$$\hat{Z}_{i_{SDE}} = \sum_{s' \to s} p(s') \hat{Z}_{i_{RAJ}}(s', \underline{Z}) / \sum_{s' \to s} p(s') \tag{33}$$

where $s = (t, \ldots, t_{T_i})$ is a sample drawn as described and $s' \to s$ denotes the sum over all samples obtained by permuting the coordinates of s: this estimator is called Murthy's (1957) *symmetrized Des Raj estimator* (SDE). Variance and its unbiased estimator in this case are [41]:

$$V(\hat{Z}_{i_{SDE}}) = \frac{1}{2} \sum_{i \neq j \neq 1}^{N} \frac{\pi_i \pi_j (1 - \pi_i - \pi_j)}{N^2(2 - \pi_i - \pi_j)} \left(\frac{z_i}{\pi_i} - \frac{z_j}{\pi_j} \right)^2 \tag{34}$$

$$\hat{V}(\hat{Z}_{i_{SDE}}) = \sum_{i \neq j \neq 1}^{N} \frac{(1-\pi_i-\pi_j)(1-\pi_i)(1-\pi_j)}{N^2(2-\pi_i-\pi_j)^2} \left(\frac{z_i}{\pi_i} - \frac{z_j}{\pi_j}\right)^2 \tag{35}$$

Again: $\theta_{i_{SDE}} = \frac{1}{N^2} V(\hat{Z}_{i_{SDE}})$; $\hat{\theta}_{i_{SDE}} = \frac{1}{N^2}\hat{V}(\hat{Z}_{i_{SDE}})$. The variance of unordered estimators are known to be less than the corresponding ordered ones [41], so $V(\hat{Z}_{i_{SDE}}) < V(\hat{Z}_{i_{RAJ}})$. In turn, $V(\hat{Z}_{i_{RAJ}})$ is smaller than the PSSWR case, since:

$$\begin{aligned} V(e_1) &= \sum_{t=1}^{N_i} \pi_t (\frac{z_t}{\pi_t} - Z)^2 = \sum_{1 \leq t \leq j \leq N_i} \sum \pi_t \pi_j [\frac{z_{i,t}}{\pi_t} - \frac{z_{i,j}}{\pi_j}]^2 \\ V(e_2) &= \sum_{1 \leq t \leq j \leq N_i} \sum (1-\pi_t-\pi_j)\pi_t\pi_j [\frac{z_{i,t}}{\pi_t} - \frac{z_{i,j}}{\pi_j}]^2 < V(e_1); \end{aligned} \tag{36}$$

and so on $V(e_{T_i}) < V(e_{T_i-1}) \ldots V(e_1)$. So:

$$V(\hat{Z}_{i_{RAJ}}) = \frac{1}{T_i^2} \sum_{j=1}^{T_i} V(e_j) < \frac{V(e_1)}{T_i} \tag{37}$$

The latter corresponds to the variance in case of PPSWR (Equation 19). For what said: $V(\hat{Z}_{i_{SDE}}) < V(\hat{Z}_{i_{RAJ}}) < V(\hat{Z}_{i_{PPSWR}})$. Thus, both RHC and SDE are more efficient than the with-replacement case. We neglected the PPSWR case just like we neglected the SRSWR case. In summary, we consider the following sampling techniques: SRSWOR (denoted, for brevity, **SRS**), which selects tests by simple random sampling without re-selecting the same test cases; Stratified SRSWOR (denoted, for brevity, **S-SRS**), which refines the previous approach by stratifying the domain selects via SRS within each stratum; RHC for PPSWOR (denoted, for brevity, **RHC**) and SDE for PPSWOR (denoted, for brevity, **SDE**), which considers the unequal probability of selection to further improve efficiency of the estimator. These approaches are compared in the following Section experimentally. It is finally worth to note that estimations are available at each iteration; thus the final estimates are adjusted by using formulas of sampling on successive occasions for unmatched subsample [42] [41], not presented here for lack of space.

IV. Evaluation

This Section reports results of the empirical evaluation to assess the AST performance and compare the test selection techniques to each other on real programs under several scenarios. Besides described techniques, we also run the conventional operational testing as baseline for comparison.

A. Testing Scenarios

Subject Programs

We exploit the same testbed used in our mentioned recent paper [23]. Techniques are applied to four programs taken from the SIR repository [43]: *Make* (v3.79), *SIENA* (v1.15), *Grep* (v2.4) and *NanoXML* (v2.2). The programs have the availability of a limited number of test cases generated by the category-partition method (Table I, column 5). These have been enlarged by modifying the available TSL specifications. The additional test cases are generated by removing constraints (e.g., "single" and "error" constraints) and adding choices to the existing ones (e.g., environment choices), according to the category-partition method (Table I, column 7). Programs are available with faults seeded inside, but, since they are conceived for regression testing purpose, we ignored them and injected new faults from scratch according to the G-SWFIT technique [44], [45]. G-SWFIT acts at source code level, by exploiting a set of fault operators derived from the well-known Orthogonal Defect Classification (ODC). We considered the same distribution as the one actually observed in field studies about the presence of ODC fault types into programs [45]. An automatic injection tool based on G-SWFIT is used (SAFE - SoftwAre Fault Emulation) [46] [44], already adopted in our previous work [47] [23] [48], whose aim is to increases the representativeness by spotting all the potential locations for each different type of fault. As number of faults, we chose four equally-spaced levels (Table I, column 6) to test the approach with different reliability values.

At each test execution, the possible failure occurrence is recorded. To recognize failures, we keep a "gold" version without faults seeded, and the failure is said to have occurred if the output of the two versions under the same test case is discordant. To evaluate performance under different available testing budget, each experiment will select a fixed number of test cases from the available ones. We consider 8 points ranging from T = 100, 200, ..., 800 test cases.

Program	Lang.	LoC	Vers.	Initial N. of Test cases	Final N. of Test cases	N. of Faults
Make	C	35545	3.79	1043	9238	24
Siena	Java	6035	1.15	567	6846	6
Grep	C	10068	2.4	809	7041	12
NanoXML	Java	7646	2.2	237	7077	18

TABLE I: Overview of the considered programs.

Partitioning and operational profile

AST exploits the initial partitioning of the overall input space into entities E_i with domain D_i. As mentioned above, entities can be modules, components or partitions. In this experiment, an entity is a partition. Test cases are partitioned into disjoint domains on the basis of the functionalities they are intended to test, getting respectively, 7 partitions for Grep, 6 for Make, 6 for SIENA and 7 for NanoXML. At lower level, three out of four techniques (namely, S-SRS, RHC and SDE) allows exploiting also a further partitioning of inputs in D_i[6]. In this experiment, we considered the *choices* of category-partition as $C_{i,h}$ classes, having inputs in a choice very similar characteristics. As these classes are the finest grain that tester is assumed to know, profiles are generated by assignment of probabilities to classes. Specifically, we distinguish two cases corresponding to possible profile knowledge level of tester:

[6]This double-level of partitioning is more relevant in the cases where entities are components, subsystems or modules, e.g., in the case of large scale systems, where the advantage of adaptation could be more relevant. Here double-level is implemented for illustrative purpose.

- The case of RHC and SDE, where class-level knowledge is required: given a domain D_i, probabilities $p_{i,h}$ are randomly assigned to each class h so that $\sum_i \sum_h p_{i,h} = 1$. A constraint on these assignment is a distinction that we make between boundary-values classes (let us denote with C'_h) from the others (C''_h), since the former as assumed to occur less frequently in operation. To generate a profile, a uniformly distributed random value in [0,1], ν'_h, is assigned to each class C'_h, and is then normalized to sum up to 1: $p'_{i,h} = \frac{\nu'_h}{\sum'_h \nu'_h}$. The same is done for classes C''_h, obtaining $p''_{i,h}$. Then, probabilities $p'_{i,h}$ are rescaled by 1/3, while $p''_{i,h}$ are rescaled by 2/3, to account for the less expected occurrence of boundary-value choices. Probability assigned to each input is then: $p_t = \frac{p_{i,h}}{|C_{i,h}|} \quad \forall i, h$, and: $p_i = \sum_h p_{i,h}$ for domain D_i.
- In the case of S-SRS technique, the domain is still partitioned in classes (again, corresponding to *choices*), but no knowledge about the profile at class-level is assumed. Thus, the random assignment is made at domain level, by generating a [0,1] number ν_i attached to D_i and normalized ($\frac{p_i}{\sum_i p_i}$) to sum up to 1. The case of SRSWR and SRSWOR is the same as S-SRS, namely knowledge is limited to an entire domain D_i (in these cases: $p_t = p_i/|D_i|$).

As in [5], [23], [18], [1], we do not focus on one specific profile in the evaluation. The above procedure is repeated to generate three profiles, P_1, P_2, P_3.

B. Evaluation criteria

Given the above scenario, we have: 8 points for the number of test cases $\times$ 4 programs $\times$ 3 profiles $\times$ 4 techniques = 384 different scenarios. For each scenario j, we have the *true reliability* R_j as $R_j = 1 - \sum_{t \in D} p_t z_t$ ($z_t = 1$ if t leads to failure $z_t = 0$ otherwise)[7]. Since testing selection criteria are probabilistic, each scenario j is run 100 times to draw statistically valid conclusions. At the end of each run r, we compute the reliability estimate given by the technique under test, $\hat{R}_{r,j}$, using Equation 1 and Equations for θ_i estimate for each technique. Then, the sample mean value M of the estimates at each run and the sample variance S are computed:

$$\begin{aligned} M(\hat{R}_j) &= \tfrac{1}{100} \sum_{r=1}^{100} \hat{R}_{r,j} \\ S(\hat{R}_j) &= \tfrac{1}{100-1} \sum_{r=1}^{100} (\hat{R}_{r,j} - M(\hat{R}_j))^2 \end{aligned} \tag{38}$$

To assess the efficiency of the estimators, we compare the sample variances S[8].

V. Result

A. Results

Results for each program and profile are in Figures 2a-2l, reporting, the sample variance S. All the techniques remarkably outperform operational testing, hence adaptiveness for test allocation among entities plays a key role in reaching efficent estimates earlier. Very low values are achieved with 800 test cases, as reported in the Figures' captions. Regarding the relative comparison among test selection techniques, RHC and SDE have similar results in all the profile/program pairs, and in almost all the cases are better than SRS and S-SRS. The latter is also better than SRS in most cases, but not always. Results are quite invariably across profile-program pairs.

To have statistically significant results, we run non-parametric ANOVA, with significance at $\alpha = 0.01$. We adopt the *Friedman*'s test to test the hypothesis of "no difference" among compared techniques (rejected at $p\text{-}value < 0.001$), followed by *post hoc* analysis to detect which techniques differ significantly[9]. Table II lists the results. Results confirm that all the observed differences are statistically significant, except SDE and RHC, which thus resulted to provide reliability estimators with a similar efficiency.

Pairwise Comparison: *p-values*				
	SDE	*RHC*	*S-SRS*	*SRS*
SDE	-	0.4310	**4.42E-13**	**5.52E-25**
RHC	-	-	**8.31E-15**	**1.90E-23**
S-SRS	-	-	-	**4.22E-06**

TABLE II: Comparison for variance **S**.

B. Threats to validity

The accuracy of results depends on the number of test cases used as test input domain, thus we enlarged the initial test suite by an order of magnitude to limit this threat. The additional tests are generated by category-partition: although it is a well-defined method, its subjective application could affect the result. Representativeness of seeded faults is a further threat. Despite we reduce the bias of artificial fault seeding by injecting more representative faults than SIR's faults, real faults might be different. External validity threats are related to the *subject programs* and *profiles*. We used programs from a known repository, and with quite different features. However, changing programs might entail different results. About profiles, we used three different profiles generated randomly; further changing the profile could yield different results. Treatments are replicated 100 times in 384 combinations to limit these threats.

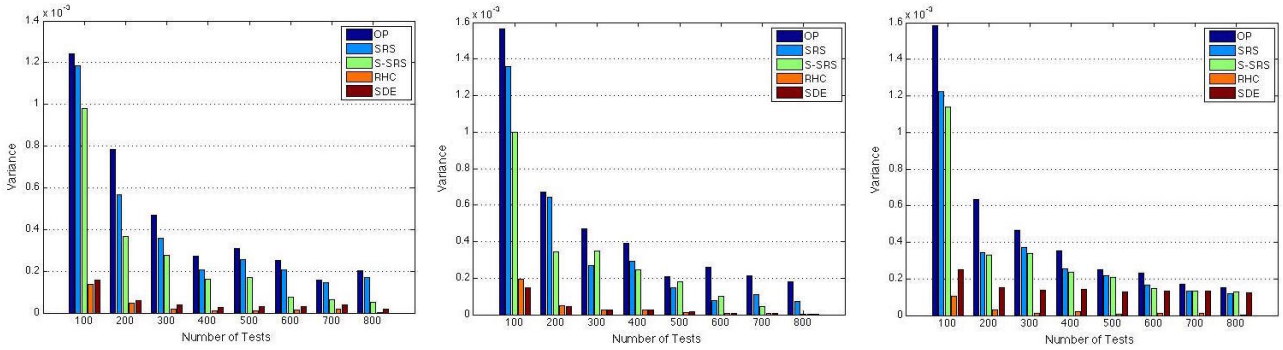
VI. Conclusion

The presented framework defines a strategy to adaptively allocate tests along with test selection algorithms to exploit knowledge about the input domain and expected usage. Analytical and empirical comparison provide figures about performance with respect to conventional operational testing and among presented algorithms. Besides results on our case studies, the AST framework generally means to pave the ground for a better integration of survey sampling methods in the theory of software reliability assessment, in order to better exploit the available testing-related knowledge to devise more efficient estimators.

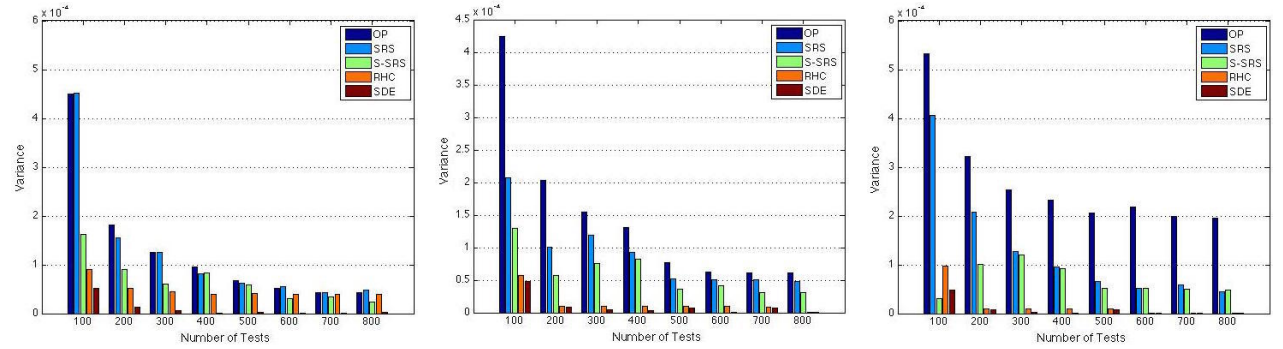
[7]We know the faults injected and get the faults matrixes by running all the tests and checking which input leads to failure; this allows getting the true failure rates θ_i and thus R

[8]Since the estimators are unbiased, both the sample variance and the sample MSE can be used for comparison. We opted for the former since the analytical comparison is in terms of variance

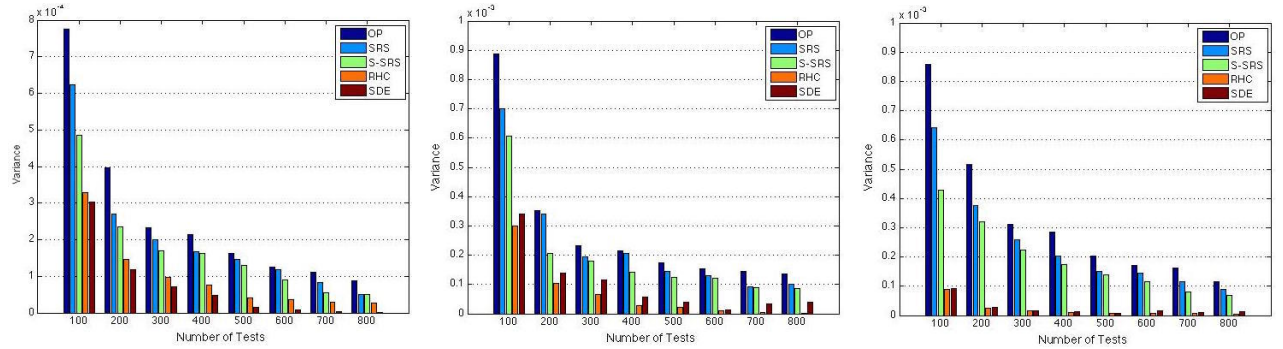
[9]The *Nemenyi* test is used for post-hoc after a non-parametric ANOVA [49]



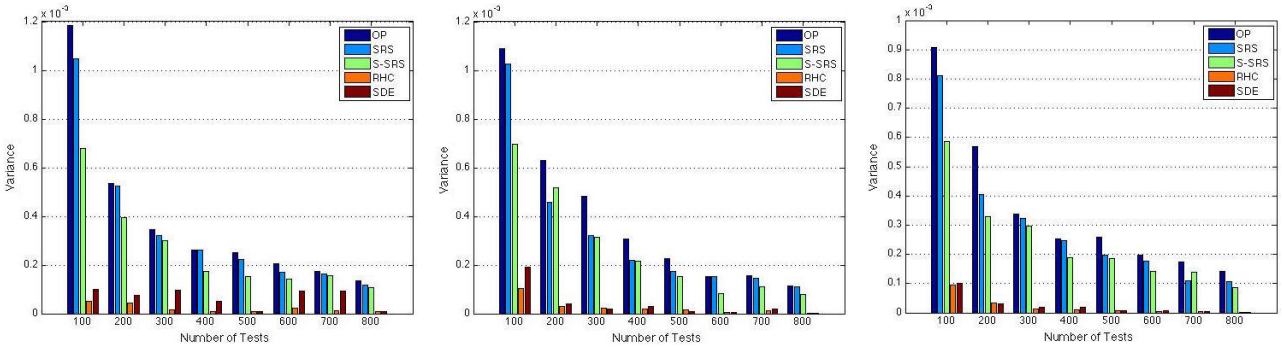
(a) Make Profile 1. *Best Variance: RHC*: 5.86E-07(b) Make Profile 2. *Best Variance: SDE*: 4.498E-07(c) Make Profile 3. *Best Variance: RHC*: 5.41E-07



(d) SIENA Profile 1. *Best Variance: SDE*: 5.00E-07(e) SIENA Profile 2. *Best Variance:* SDE: 2.60E-9(f) SIENA Profile 3. *Best Variance:* RHC: 7.21E-09



(g) Grep Profile 1. *Best Variance: SDE*: 9.37E-07(h) Grep Profile 2. *Best Variance: RHC*: 1.62E-06(i) Grep Profile 3. *Best Variance: RHC*: 5,62E-06



(j) NanoXML Prof. 1. *Best Variance: RHC::*(k) NanoXML Prof. 2. *Best Variance: SDE:*(l) NanoXML Prof. 3. *Best Variance: RHC*: 5.14E-07.67E-06 1.07E-07 07

Fig. 2: Sample variance of reliability estimate

ACKNOWLEDGMENT

This work has been supported by EU under the FP7 Marie Curie Industry-Academia Partnerships and Pathways (IAPP) projects ICEBERG (nr. 324356, www.iceberg-sqa.eu) and CECRIS (nr. 324334, www.cecris-project.eu).

REFERENCES

[1] J. Lv, B.-B. Yin, and K.-Y. Cai. On the asymptotic behavior of adaptive testing strategy for software reliability assessment. *IEEE Trans. on Software Engineering*, 40(4):396–412, 2014.

[2] J. Lv, B-B. Yin, and K-Y. Cai. Estimating confidence interval of software reliability with adaptive testing strategy. *Journal of Systems and Software*, 97:192–206, 2014.

[3] J.D. Musa. Software reliability-engineered testing. *Computer*, 29(11):61–68, Nov 1996.

[4] H.D. Mills, M. Dyer, and R.C. Linger. Cleanroom software engineering. *IEEE Software*, 4(55):19–24, 1987.

[5] K-Y. Cai, Y-C. Li, and K. Liu. Optimal and adaptive testing for software reliability assessment. *Information and Software Technology*, 46(15):989–1000, Dec 2004.

[6] H. Pham. *Software System Reliability*. New York, NY, USA: Springer-Verlag, 2006.

[7] R.W. Selby, V.R. Basili, and F.T. Baker. Cleanroom software development: An empirical evaluation. *IEEE Trans. on Software Engineering*, SE-13(9):1027–1037, Sept 1987.

[8] P.A. Currit, M. Dyer, and H.D. Mills. Certifying the reliability of software. *IEEE Trans. on Software Engineering*, SE-12(1):3–11, 1986.

[9] J.H. Poore. A case study using cleanroom with box structures adl. Technical report, Software Engineering Technology CDRL 1880, 1990.

[10] D. Fox. Adapting the Sample Size in Particle Filters Through KLD-Sampling. *Int. Journal of Robotics Research*, 22:2003, 2003.

[11] Amrit L. Goel and Kazu Okumoto. Time-dependent error-detection rate model for software reliability and other performance measures. *IEEE Trans. on Reliability*, R-28(3):206–211, 1979.

[12] A. L. Goel. Software reliability models: Assumptions, limitations and applicability. *IEEE Trans. on Software Engineering*, SE-11(12):1411–1423, 1985.

[13] S.S. Gokhale and K.S. Trivedi. Log-logistic software reliability growth model. In *Proc. 3rd Int. High-Assurance Systems Engineering Symposium (HASE)*, pages 34–41, 1998.

[14] K. Ohishi, H. Okamura, and T. Dohi. Gompertz software reliability model: Estimation algorithm and empirical validation. *Journal of Systems and Software*, 82(3):535–543, 2009.

[15] V. Almering, M. Van Genuchten, G. Cloudt, and P.J.M. Sonnemans. Using software reliability growth models in practice. *IEEE Software*, 24(6):82–88, 2007.

[16] R.H. Cobb and H.D. Mills. Engineering software under statistical quality control. *IEEE Software*, 7(6):45–54, Nov 1990.

[17] R.C. Linger and H.D. Mills. A case study in cleanroom software engineering: the ibm cobol structuring facility. In *12th Int. Computer Software and Applications Conference, COMPSAC 88*, pages 10–17, Oct 1988.

[18] K-Y. Cai, C-H. Jiang, H. Hu, and C-G. Bai. An experimental study of adaptive testing for software reliability assessment. *Journal of Systems and Software*, 81(8):1406–1429, 2008.

[19] K-Y. Cai. Optimal software testing and adaptive software testing in the context of software cybernetics. *Information and Software Technology*, 44(14):841–855, 2002.

[20] A. Podgurski, W. Masri, Y. McCleese, F.G. Wolff, and C. Yang. Estimation of software reliability by stratified sampling, 1999.

[21] F.b.N. Omri. Weighted statistical white-box testing with proportional-optimal stratification. In *Proc. 19th International Doctoral Symposium on Components and Architecture*, WCOP'14, pages 19–24. ACM, 2014.

[22] D. Cotroneo, R. Pietrantuono, and S. Russo. Combining Operational and Debug Testing for Improving Reliability. *IEEE Trans. on Reliability*, 62(2):408–423, 2013.

[23] D. Cotroneo, R. Pietrantuono, and S. Russo. Relai testing: A technique to assess and improve software reliability. *IEEE Trans. on Software Engineering*, 42(5):452–475, 2016.

[24] P. Popov. Proc. 21st int. conference on computer safety, reliability and security. SAFECOMP, pages 139–150. Springer, 2002.

[25] I. Gashi, P. Popov, and V. Stankovic. Uncertainty explicit assessment of off-the-shelf software: A bayesian approach. *Information and Software Technology*, 51(2):497–511, 2009.

[26] H. Singh, V. Cortellessa, B. Cukic, E. Gunel, and V. Bharadwaj. A bayesian approach to reliability prediction and assessment of component based systems. In *Software Reliability Engineering, 2001. ISSRE 2001. Proceedings. 12th International Symposium on*, pages 12–21, Nov 2001.

[27] L. Strigini and D. Wright. Bounds on survival probability given mean probability of failure per demand; and the paradoxical advantages of uncertainty. *Reliability Engineering & System Safety*, 128:66–83, 2014.

[28] L. Strigini and A. Povyakalo. *Computer Safety, Reliability, and Security: 32nd International Conference, SAFECOMP 2013, Toulouse, France, September 24-27, 2013. Proceedings*, chapter Software Fault-Freeness and Reliability Predictions, pages 106–117. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[29] D. Cotroneo, R. Pietrantuono, and S. Russo. A learning-based method for combining testing techniques. In *Proc. 35th Int. Conference on Software Engineering (ICSE)*, pages 142–151. IEEE, 2013.

[30] M. Sridharan and A.S. Namin. Prioritizing mutation operators based on importance sampling. In *21st Int. Symposium on Software Reliability Engineering (ISSRE)*, pages 378–387, Nov 2010.

[31] Chin-Yu Huang and Jung-Hua Lo. Optimal resource allocation for cost and reliability of modular software systems in the testing phase. *Journal of Systems and Software*, 79(5):653–664, 2006.

[32] Chin-Yu Huang, Jung-Hua Lo, Sy-Yen Kuo, and M.R. Lyu. Optimal allocation of testing resources for modular software systems. In *Software Reliability Engineering, 2002. ISSRE 2003. Proceedings. 13th Int. Symposium on*, pages 129–138, 2002.

[33] R. Pietrantuono, S. Russo, and K.S. Trivedi. Software Reliability and Testing Time Allocation: An Architecture-Based Approach. *IEEE Trans. on Software Engineering*, 36(3):323–337, 2010.

[34] M.R. Lyu, S. Rangarajan, and A.P.A. Van Moorsel. Optimal allocation of test resources for software reliability growth modeling in software development. *IEEE Trans. on Reliability*, 51(2):336–347, 2002.

[35] G. Carrozza, R. Pietrantuono, and S. Russo. Dynamic test planning: a study in an industrial context. *International Journal on Software Tools for Technology Transfer*, 16(5):593–607, 2014.

[36] K-Y. Cai. Towards a conceptual framework of software run reliability modeling. *Information Sciences*, 126(1–4):137–163, 2000.

[37] K.S. Trivedi. *Probability and statistics with reliability, queuing and computer science applications (2nd ed.)*. John Wiley and Sons Ltd., Chichester, UK, 2001.

[38] Sharon L. Lohr. *Sampling Design and Analysis*. Duxbury Press; 2 edition, 2009.

[39] C. Catal and B. Diri. A systematic review of software fault prediction studies. *Expert Systems with Applications*, 36(4):7346–7354, 2009.

[40] J.N.K. Rao, H.O. Hartley, and W.G. Cochran. On a simple procedure of unequal probability sampling without replacement. *Journal of the Royal Statistical Society. Series B (Methodological)*, 24(2):482–491, 1962.

[41] Arijit Chaudhuri. *Survey Sampling Theory and Methods*. Chapman & Hall/CRC, Second Edition, Taylor & Francis Group, 2005.

[42] A. Chauduri and J. W. E. Vos. *Unified theory and strategies of survey sampling*. North-Holland Publishers, Amsterdam., 1988.

[43] SIR: Software-artifact infrastructure repository.

[44] R. Natella, D. Cotroneo, J.A. Duraes, and H.S. Madeira. On fault representativeness of software fault injection. *IEEE Trans. on Software Engineering*, 39(1):80–96, 2013.

[45] J.A. Duraes and H.S. Madeira. Emulation of software faults: A field data study and a practical approach. *IEEE Trans. on Software Engineering*, 32(11):849–867, 2006.

[46] Software fault emulation tool: http://wpage.unina.it/roberto.natella/tools.html.

[47] D. Cotroneo, R. Pietrantuono, and S. Russo. Testing techniques selection based on odc fault types and software metrics. *Journal of Systems and Software*, 86(6):1613–1637, 2013.

[48] A. Bovenzi, D. Cotroneo, R. Pietrantuono, and G. Carrozza. Error Detection Framework for Complex Software Systems. In *Proc. 13th European Workshop on Dependable Computing*, EWDC '11, pages 61–66. ACM, 2011.

[49] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.