RELAI testing: a technique to assess and improve software reliability

Domenico Cotroneo, *Member, IEEE,* Roberto Pietrantuono, *Member, IEEE,* Stefano Russo *Member, IEEE*

***Abstract*—Testing software for assessing or improving reliability presents several practical challenges. Conventional operational testing is a fundamental strategy that simulates the real usage of the system in order to expose failures with the highest occurrence probability. However, practitioners find it unsuitable for assessing/delivering high reliability levels, and they do not see the adoption of a "real" usage profile estimate as a sensible idea, being it a source of non-quantifiable uncertainty. Debug testing techniques aim to expose as many failures as possible, but regardless of their impact on runtime reliability. These strategies are used either to assess or to improve reliability, but cannot improve and assess reliability in the same testing session.**

This article proposes *Reliability Assessment and Improvement* (*RELAI*) testing, a new technique thought to improve the delivered reliability, by an adaptive testing scheme, while providing, at the same time, a continuous assessment of reliability attained through testing and fault removal. The technique also quantifies the impact of a partial knowledge of the operational profile. *RELAI* is positively evaluated on four software applications compared, in separate experiments, with techniques conceived either for reliability improvement or for reliability assessment, demonstrating substantial improvements in both cases.

***Index Terms*—Software Testing, Reliability, Operational Testing, Random Testing, Sampling, Operational Profile**

I. Introduction

The objective of any software testing technique is to expose failures. Testers prioritize inputs according to the pursued quality attribute, e.g., correctness, robustness, reliability, security. When high reliability is desired, tester wishes to sample those failure-causing inputs that have the largest impact on operational failure probability. This is the trend followed by those techniques based on the expected operational profile to derive tests. *Operational profile based testing* (hereafter *operational testing*) selects test cases by looking for high-occurrence failure-causing inputs, as they mainly impact reliability. This is different from conventional *debug testing*: the latter refers to techniques whose aim is to expose as many failures as possible and remove the failure-causing bugs, regardless of their possible impact on runtime reliability. There is an inherent difference between them, since the rate of failure occurrence during actual operation affects reliability. It may happen that a technique exposes many failures, but they have low impact on reliability, i.e., their occurrence frequency is low; and, as opposite, a technique may uncover few but high-occurrence failures and deliver higher reliability. Therefore, although both can improve reliability by means of fault removal, operational testing is considered as the natural way to pursue high reliability. Besides reliability improvement, techniques based on operational profile have been largely used also for reliability assessment [1], [2], [3], [4]. This makes operational testing one of the pillars of software reliability engineering practices [5].

At the same time, operational testing has been strongly criticized over the years (e.g., in [6]), as it actually presents issues that limit its wide adoption in the industrial practice. Researchers and practitioners see two major limitations in operational testing: *i)* the unsuitability for systems demanding high reliability; *ii)* the assumption of knowing the operational profile of the software under test.

The reason behind the first issue is that operational testing aims at exposing failures that will occur in operation with higher probability, neglecting low-occurrence ones. Therefore, reliability will achieve a certain stable level that becomes difficult to improve further if remaining failures are not addressed. To boost reliability beyond that limit, testing should turn, at a certain point, to prefer exposing many low-occurrence failures rather than few high-occurrence ones. The same problem stands when operational testing is used for reliability assessment of high-quality software, wherein only low-occurrence failures are likely to be present, thus yielding insufficient failure data for the estimation. This aspect is emphasized by the target systems which operational testing is usually intended for, namely mission- or safety-critical systems with high reliability requirements.

Operational profile uncertainty is the second big concern. It is not always possible for practitioners to obtain meaningful profiles, and the impact of the error they commit on estimating such profile is not known a priori. Many studies are available that discuss how this error impacts reliability observed in operation (e.g., [7], [8], [2], [9], [10]), but none of them can account for it preventively - namely, by including it in the formulation of the testing strategy. A tester would discover too late, only after a long time of operation, that the profile estimate was wrong, and that reliability achieved/assessed at the end of testing was biased. This uncertainty causes strong scepticism that makes testers prefer debug testing techniques under the assumption that more failures exposed entail higher reliability, ignoring their operational occurrence frequency.

In addressing these issues, there is a large margin of improvement for reliability testing. While it is important to exploit the knowledge of the operational profile to derive better tests for both reliability improvement and reliability assessment, a suitable solution is needed to avoid the stall of operational testing on high-occurrence failures, as well as to minimize the epistemic uncertainty of results caused by the inaccurate estimate of the usage profile.

D. Cotroneo, R. Pietrantuono, and S. Russo are with the Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione (DIETI), at the Università di Napoli Federico II, Via Claudio 21, 80125 Naples, Italy. E-mail: {cotroneo, roberto.pietrantuono, Stefano.Russo}@unina.it

In this article, we propose a new approach called *Reliability Assessment and Improvement* (*RELAI*) testing. *RELAI* is an integrated technique to improve the final delivered reliability and provide, at the same time, an accurate assessment of the achieved reliability in the same testing session, while overcoming both the low-occurrence failures problem and the inaccurate profile issue. The main contributions of *RELAI* are summarized by the following objectives it fulfils:

- *Improvement of reliability: RELAI* adopts an adaptive sampling approach, that iteratively learns from test execution results as they become available, and, based on them, allocates test cases to the most reliability-impacting input regions. The adaptive strategy avoids the "saturation" of the conventional operational testing, yielding better results, since, once the high-occurrence failures are removed, *RELAI* proactively directs the selection toward the low-occurrence ones.
- *Assessment of the achieved reliability:* within each input region with an assigned number of test cases, *RELAI* defines a second sampling strategy able to provide the interval estimate of attained reliability during testing. Reliability assessment through operational testing is usually conducted assuming the code being frozen (i.e., no bugs are removed during testing) [2], thus entailing a separate testing session devoted to assess reliability and accept the software [11]. *RELAI* removes this assumption; it addresses the problem of reliability improvement and assessment in the same testing stage, providing estimates while removing defects.
- *Assessment of the profile uncertainty impact: RELAI* predicts the error on reliability estimate caused by a tester-specified *maximum acceptable error* on the profile estimate. This is an unaddressed challenge so far, and the proposed solution is a first attempt toward the control of the uncertainty in the usage of operational profile for testing purposes.

The output information is useful to implement several testing policies, as: maximizing reliability given a testing budget, or minimizing the testing effort to attain a reliability goal, under a maximum tolerable profile error. Since the algorithm is iterative, tester can implement dynamic policies; he has always an updated view of the cost/benefit evolution during testing.

We evaluated *RELAI* through a controlled experiment on four software applications under several scenarios through four separate experiments. Results are compared with 6 techniques, 3 for reliability improvement and 3 for reliability assessment. They show a relevant gain of *RELAI* in terms of delivered reliability on already high reliability levels (improving reliability from 21% up to 90% with respect to the maximum achievable gain), a mean squared error (MSE) in reliability assessment ranging from *6E-4* to *2E-4* with 100 test cases up to *8E-6* to *1E-8* with 800 test case, and an ability to confidently predict the impact of the profile estimation error on reliability estimate in more than 98% of the cases.

In the following, we first survey the related work and introduce the basic concepts behind RELAI testing (Section II and III). In Section IV and V we present the strategy. Section VI describes the experiment; Section VII presents the results, followed by conclusive remarks (Section X).

II. Related work

A. Testing based on operational profile

Testing based on operational profile estimation, known as operational testing, is historically considered as the natural approach to test software either for reliability assessment or for reliability improvement. Its rationale is to select test cases with the same probabilities as expected in operation[1].

Several researchers, since the end of the eighties, worked to improve this promising approach [15], [16], [12], [13], [17], [18]. The most important contexts in which operational testing was used are in the frame of the Cleanroom methodology [19], [20], [15], and in the process defined by Musa, Software Reliability Engineering Test (SRET) [11]. In both cases, there are several examples of positive experience. In the Cleanroom approach, it is used as a means to assess reliability and certify the software against a given MTTF [20], [16], [21], [22]. Operational testing is formulated as reliability improvement technique also in n [23], where an analytical study is conducted to compare operational against debug testing. Further empirical evidences in favour of operational testing are reported in [24] and in [25], where it is used with multimodal systems.

In [1], [2], an adaptive testing strategy based on operational profile is used for reliability assessment via feedback-driven test case selection. The authors formulate software testing as a feedback and adaptive control problem: they use a controlled Markov chain to describe the testing process with the goal of minimizing the variance of reliability estimator. Similarly, adaptive testing is used in a recent work along with a gradient descent method [3]. Authors' results show the superiority of adaptive testing over uniform random testing and operational testing. Along the same line, a recent work used confidence intervals as driving criterion to select test cases adaptively for reliability assessment purposes [4]. All these works on adaptive testing are inspired to the area of software cybernetics [26], to which one of the two sampling schemes used by *RELAI* is also related to.

In [27], operational testing is used for reliability improvement by a Bayesian method using online data as feedback. Authors use test data to estimate the failure rates and select test cases based on them, assuming that failing test cases are a very small percentage of total tests. Authors in [28] adopt robust optimization to distribute test cases across test modules, also considering the problem of inaccurate operational profile. Operational testing is also viewed as a special case of random testing, although this term is wider and refers to any distribution (not only the operational usage one)[2]. In the field of random testing, there is a wide share of work

[1]The term *statistical testing* was also used to denote this technique [12], [13]; however, it is also being used to denote testing to satisfy adequacy criteria in terms of functional/structural properties, e.g., in [14].

[2]Indeed, random testing is also often taken to mean "uniform" random testing, where all points in the input domain are equally likely to be selected.

focusing on test effectiveness in terms of failure exposure (hence fault detection) ability. Papers more related to our idea are the ones using adaptive strategies to improve testing online. Adaptive random testing (ART) is a family of testing techniques proposed by Chen et *al.* [29], [30] [31], [32], [33]. The intuition is to improve random testing by using test results online in order to evenly distribute test cases across the input domain. There are many variants of ART, adopting different algorithms to select the next test cases given a test history – see [34] for a detailed survey. However, these are all intended to improve reliability in the broad meaning of the term, namely by exposing as many failures as possible like any debug testing approach. For what said, this is not the same as improving operational reliability, because the expected occurrence frequency in operation determines which bugs are more or less important from reliability perspective (intended as probability of not failing in operation). It might happen that such techniques deliver higher reliability than conventional operational testing, if all the detected and removed bugs collectively have a larger expected occurrence probability than the ones removed by operational testing. Therefore, we also consider ART techniques as counterpart which *RELAI* will be compared with.

RELAI is an integrated technique that combines test selection strategies for reliability improvement and for reliability assessment. *RELAI* formulates the reliability improvement problem as an adaptive iterative sampling problem, whose aim is to find the distribution of test cases among partition that maximizes the delivered reliability. This is to overcome the dualism between operational and debug testing (including ART), arguing if it is better, from reliability perspective, to expose few but high-occurrence failures or many failures but regardless their occurrence probability [6]: *RELAI* selects, at each iteration, the input cases that are expected to contribute more to the operational failure probability reduction in the next iteration.

Moreover, the usage of operational-profile based testing for either reliability improvement or assessment in a mutually exclusive way, led us to integrate a reliability assessment step within the improvement strategy. In fact, whenever operational-profile based testings is used for reliability assessment, the code is assumed to not change during testing, i.e., detected defects are not removed [2], [3], [4], [35]. Unlike *RELAI*, these techniques also assume to use a sampling with replacement (i.e., the same test case can be selected several times) and an equal probability of selection within partitions. *RELAI* is formulated to remove these assumptions, and to provide a reliability estimate while also removing detected defects. This is achieved by a new sampling scheme based on survey sampling method that admits a sampling without replacement with unequal selection probabilities [36], leading to more accurate estimates.

B. Impact of operational profile estimation

The further problem that operational-profile based testing suffer from is the estimation of a correct operational profile. The effectiveness of operational testing, both in assessing and in improving reliability, has always been seriously undermined by the difficulties in obtaining meaningful profiles. All the mentioned approaches assume a perfect knowledge of the runtime usage, which is most likely untrue. A non-negligible slice of literature proposes solutions to support the task of obtaining good profiles, e.g., through state models [37], Markovian or Bayesian models describing the expected usage [12], [38], or UML documentation [39]. But basically the accuracy of the profile depends on the accuracy of the information on how the system will be used [40], which is difficult to obtain. Empirical analyses have been conducted to evaluate the *impact* of the profile estimation error on reliability estimate. These studies are quite contradictory. The authors in [7], [8] state a non-negative effect of operational profile errors on reliability estimate. In [41] reliability estimates are shown to not be affected by such errors. This issue has been studied also recently in [2], where authors experimentally evaluate techniques performance in assessing reliability in presence of error. Differently from the previous ones, the work by Chen et *al.* [9] attests relevant deviations on reliability estimates caused by the profile error. A more recent analysis shows that the relation between the profile variation and reliability depends on the error on the testing effort [10]. Despite these contrasting results, it is important to remark that the relation between the profile estimation error and reliability has been studied empirically so far, by observing the effect of the error on the true reliability assumed to be known. In a real situation, this impact would be observed only after a long operational time, when the true reliability can be assessed faithfully. To our knowledge, there is no technique to assess, preventively, the impact of a possible profile error on reliability attained soon after testing, making it hard to apply operational testing in practice. In our previous work [42], we started addressing this challenge, analyzing theoretically the relation between the profile error and the number of tests. That analysis did not end up with a practical testing method, but paved the ground to the formulation of the strategy presented hereafter. Overall, *RELAI* differs from the cited works in several aspects: the test selection algorithm accounts for both reliability improvement and reliability assessment needs, which are challenges usually addressed separately; both reliability improvement and assessment are taken in a novel way, by previously unused sampling schemes (this is important especially for reliability improvement, where considerably less research has been conducted); *RELAI* considers the profile estimation error in its formulation, allowing accounting for the impact of such a source of uncertainty before the system goes in operation.

III. Terminology

This Section introduces the terminology adopted in the following. Testing a program is the process of *i)* exercising it with different test cases, selected from the set of all possible inputs according to a selection criterion, and *ii)* observing the output, comparing it with the expected one such that, if they are discordant, a failure is said to have occurred. Inputs provoking failures are called failure-causing inputs or *failure*

points. When a failure occurs, a change is made to the program to remove what is believed to be the cause of the failure, or "fault". Since there may be several possible changes able to avoid the failure, the fault related to an observed failure is not uniquely defined. We thus rely on the notion of failure, rather than that of fault, and borrow the concept of *failure region* of the input space (as in, e.g., [23], [43]). A failure region is the set of failure points that is eliminated by a program change aimed at removing the fault. An input point t to a program under test is characterized by a predicate: $y_t = 1$ if the execution leads to a failure, namely, it is a failure point; $y_t = 0$ otherwise.

An operational profile is a quantitative characterization of how a system will be used. There are several models of operational profiles [44]. In this work, the Musa's model is considered. We build the profile by assigning probability values to all input cases representing the probability that each will occur in operation. Thus, it can be thought as a probability distribution over the set of the input points D. We denote this distribution with P, that assigns a probability p_t to each input $t \in D$. In operational testing, assuming a perfect estimate of the operational profile, p_t is also the probability that the input t will be selected during testing. But in the real world, the profile estimate is affected by an error, and another probability distribution is actually used to select test cases. We denote this distribution with $\hat{P}$, and its probability values with $\hat{p}_t$. To account for the profile estimation error, we consider the differences $\varepsilon_t = p_t$ - $\hat{p}_t$, representing the estimation error for each point of the input domain.

Reliability is the probability of not failing in operation: in the literature, it is mainly measured either on a continuous-time or on a discrete-time basis - the two approaches are related, as explained in [45], [46]. While the former assesses the failure probability as a function of time (e.g., calendar time, CPU time), the latter uses the frequency (or probability) of successful runs, which in many cases is more meaningful [3]. As for other studies in the software testing field, this study focuses on the discrete-time definition; reliability is: $R = 1 - \sum_{t \in D} p_t y_t$, where the summation represents the probability of selecting at least one failure point in operation.

IV. The RELAI Testing strategy

A. Assumptions

To formulate *RELAI*, we do the following assumptions:

1) Each test case leads the software under test to failure or success. Given a software program under test (SUT) and a test case t applied to it, let ω_t be the actual output value of the SUT executed with test case t and ω'_t the expected output value of the SUT under t. Let y_t be a label denoting the outcome of the execution of the SUT with test case t; then:

$$\begin{cases} y_t = 1 & \text{iff} \quad \omega_t \neq \omega'_t \\ y_t = 0 & \text{iff} \quad \omega_t = \omega'_t \end{cases} \tag{1}$$

assuming we are able to know $\omega'_t \ \ \forall t$. In other words, t applied to the SUT is either a failure point or not. We assume we are always able to correctly determine if a test is successful or not by comparing the actual output ω_t with the known expected output ω'_t.

2) The possibility to select a test case at a given time is independent of previously selected and executed test cases; namely, all the non-executed test cases are selectable each time. Given the set of all the test cases $T = \{t_1, \ldots, t_n\}$, and the set of executed test cases at a given time, $T' = \{t_1, \ldots, t_k\}$, with $k < n$, any of the non-executed test cases ($T'' = T - T' = \{t_{k+1} \ldots, t_n\}$) can be selected. This affects the way in which a "test case" is defined, since, if the assumption is not met, a set of tasks can be grouped together in a single test case, so that at the end of the test case the system goes back to the initial state [3].

3) The output of a test case execution is independent of the history of testing; in other words, a failing test case is always such, independently from the previously executed test cases. Note that, even though the possibility to select a test case is not influenced by previous tests, its outcome can still be influenced. Formally, let $T = \{t_1, \ldots, t_n\}$ be the set of all the test cases for a given program s and $N = |T|$ the number of test cases. Denote with $D_{K,(N-1)}$ the set of sequences without repetition (i.e., dispositions) of K over $N-1$ test cases, and with d_k one of such sequences. Then, given a test case t_j ($j \in [1, N]$) executed on s after the execution of K test cases (with $t_i \neq t_j$, for $i = 1, \ldots, i = K$), its output ω_j is independent from any previously executed sequence $d_k \in D_{K,(N-1)}$ and is always the same (suppose it being equal to ω^*):

$$\begin{aligned} Pr(\omega_j = \omega^* | d_k) = 1 \quad & \forall d_k \in D_{K,(N-1)} \\ & k = 1, 2, \ldots, K; \\ & K = 1, 2, \ldots, (N-1) \end{aligned} \tag{2}$$

4) Let F be the set of q faults present in the program: $F = \{f_1, f_2, \ldots, f_i, \ldots, f_q\}$, and R be the set of the corresponding failure regions: $R = \{r_1, r_2, \ldots, r_i, \ldots, r_q\}$. As defined in Section III, a failure region is a set of failure points, namely: $r_i = \{t_1, t_2, \ldots, t_k, \ldots, t_s\}$, with $s = |r_i|$, such that t_k activates the fault f_i and cause a failure ($y_{t_k} = 1$). If an executed test case t_j exposes a failure, namely $t_j \in r_i$, a debugging action A is performed without introducing new faults (*perfect debugging*), and all the failure points of the corresponding failure region r_i are corrected, so that re-executing any test case of that region ($t_j \in r_i$) no longer causes a failure. Formally, after the execution of the debugging action A, we have that:

$$\begin{aligned} & |F|_A = |F| - 1 \\ & |r_i|_A = 0 \\ & \textstyle\sum_{h=1 \neq i}^{|R|} |r_h|_A = \sum_{h=1 \neq i}^{|R|} |r_h| - |r_i| \end{aligned} \tag{3}$$

where $|F|_A$, $|r_i|_A$, $|r_h|_A$ are the cardinalities of faults and failure regions after the execution of the debugging action A.

5) In this study, we assume that the selection of test cases is *without replacement*, namely any executed test case (either successful or not) will be no longer repeated in the future. Formally, given the set of all the test cases $T = \{t_1, \ldots, t_n\}$, and the set of executed test cases at a given time, $T' = \{t_1, \ldots, t_k\}$, with $k < n$, the next test case to select, t_j, cannot be in T': $t_j \notin T'$.
6) The input domain D is decomposed into a set of m subdomains: $\{\mathrm{D}_1, \mathrm{D}_2, \ldots, \mathrm{D}_m\}$. The number of subdomains and the partitioning criterion are decided by the tester. In general, there are several ways in which a tester can partition the test suite, provided that test cases in a partition have some properties in common (e.g., based on functional, structural, or profile criteria). These are usually dependent on the information available to test designers and on tester's objective. The choice does not affect the proposed strategy, which just assumes the presence of subdomains, but of course different results can be obtained according to it. The effect of different partitioning criteria on results is out of the scope of this paper and is left to future research.

B. Overview

Reliability Assessment and Improvement Testing (RELAI) starts as a conventional operational profile based testing. An estimate of the profile, $\hat{P}$, is used as reference distribution to select test cases, with tester assigning an occurrence probability $\hat{p}_t$ to each input[3]. For each subdomain, we define the probability of selecting a failure point from D_i as: $\varphi_i = \theta_i \sum_{t \in D_i} p_t$, 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. Thus, the true reliability is computed as:

$$R = 1 - \Phi = 1 - \sum_{i=1}^{m} \varphi_i \tag{4}$$

where Φ is the operational failure probability. The objective of *RELAI* is to improve the expected delivered reliability while providing an estimate of the achieved level along with the offset caused by a possible profile estimation error. The *RELAI* algorithm is based on a *two-stage sampling*, that we call *domain-level* and *subdomain-level* sampling scheme, respectively. At domain level, sampling is to decide the number of test cases for each subdomain in order to exercise input regions mainly impacting Φ. At subdomain level, a different type of sampling is adopted to select test cases in a way to enable the estimation of θ_i, and thus φ_i, values. At the end of each iteration, the output at subdomain level is used to drive the algorithm at domain level. The algorithm considers, as input:

- T: the total number of test cases available as budget;
- R_0: the reliability target (and, optionally, the desired confidence interval, denoted as CI(R_0)=[R_{0_l}; R_{0_u}], at the specified C_{R_0}% confidence level);
- ε_0: the maximum error on the operational profile estimate that the tester decides to tolerate. It is expressed as ε_0 such that $\varepsilon_t < |\varepsilon_0|$, with $t \in D$, in at least C_{ε_0}% of cases (e.g., the error is $\varepsilon_t < |0.01|$ for 99% of the input points). This represents an important input for an accurate reliability assessment, as it allows *RELAI* accounting for the error that tester inevitably commits in estimating the profile.

Depending on the tester choice, these inputs are used to implement (at least) the following policies:

Policy 1 - Reliability improvement:
Goal: maximize expected reliability. The algorithm selects test cases in order to improve the expected delivered reliability.
Input: the budgeted number of test cases T; the maximum acceptable profile error $\langle \varepsilon_0, C_{\varepsilon_0}\% \rangle$; optionally, the desired confidence C_R%, determining the output confidence interval.
Output: the interval estimate of the achieved reliability ($\langle \hat{R}$, CI($\hat{R}$)$\rangle$ at C_R%), and the impact of the specified profile error on reliability assessment accuracy, let us denote it, for now, as a generic offset Δ.

Policy 2 - Reduction of the number of test case to execute:
Goal: reducing the number of required test cases to achieve a reliability target. The algorithm tends to attain the target as soon as possible, and runs until it is reached.
Input: the reliability target, R_0; the desired acceptable error $\langle \varepsilon_0, C_{\varepsilon_0}\% \rangle$; optionally, the desired confidence interval CI(R_0)=[R_{0_l}; R_{0_u}] at C_{R_0}% confidence level.
Output: the output is the number of executed test cases, the interval estimate of the achieved reliability, and the possible offset Δ under the specified profile error. This information can be used to stop testing according to several criteria. For instance, tester can consider the point reliability estimate $\hat{R}$, stopping testing when $\hat{R} \geq R_0$; the lower bound of the interval estimate $\hat{R}_l$ (which is a more conservative choice, requiring $\hat{R}_l \geq R_0$); or the upper bound $\hat{R}_u$ (e.g., to stop testing earlier, as soon as $\hat{R}_u \geq R_0$, at the expense of less confident results). The algorithm provides this estimate at each iteration, thus supporting informed decisions during testing. The high-level steps to implement these policies are schematized in Figure 1:

1) Choose the policy and provide the corresponding inputs.
2) Decide the number of test cases to execute at the first iteration, denoted as $T^{(0)}$, and distribute them to each subdomain D_i by the *domain-level* sampling scheme. $T_i^{(0)}$ denotes the number of test cases to draw from D_i.
3) At each iteration k, with $k \geq 0$:
   a) Select and run test cases by the *subdomain-level* sampling scheme, drawing $T_i^{(k)}$ test cases from each subdomain. From the observed results, compute: $\hat{\varphi}_i^{(k)}$, namely the estimates of φ_i at iteration k; the interval estimate of reliability, $\langle \hat{R}^{(k)}$, CI($\hat{R}^{(k)}$)$\rangle$; the offset Δ.
   b) For policy 1, evaluate if the number of test cases executed so far is not less than T; if yes, stop the

[3] In [2], [11] and others, a global $\hat{p}_i$ value is assigned to an entire subdomain (i.e., $\sum_{t \in D_i} \hat{p}_t = \hat{p}_i$); namely, the within-domain distribution is uniform, with every input having $p_t = p_i / |D_i|$. Depending on the partitioning strategy, other choices could be done assigning the same p_t values to groups of inputs (e.g., to boundary values of a functionality with the same expected occurrence, or to a functionality's inputs in a subdomain including more similar functionalities). Here we are considering the more general case of each input with an assigned probability value.

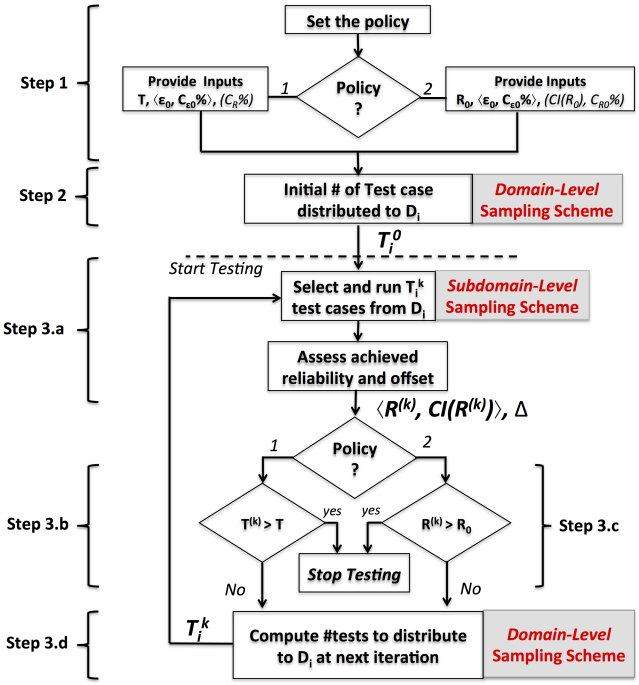


Fig. 1: Block scheme of *RELAI*

testing, providing results of 3.(a) as final output; otherwise, go to to step 3.(d).

c) For policy 2, evaluate if the current estimate satisfies the target $\langle R_0, \text{CI}(R_0)\rangle$. If yes, stop the testing, provide the number of run test cases and results of 3.(a) as output; otherwise, go to step 3.(d).

d) By using the $\varphi_i^{(k)}$ estimates and the *domain-level* sampling scheme, compute the number of test cases for each subdomain for the next iteration, assigning more tests to subdomains more likely to improve reliability. Skip to the next iteration.

In the following, we detail the main steps of the algorithm, considering first the case of no profile estimation error ($\varepsilon_t = 0$, $\forall t \in D$), and then the case of profile estimate affected by error.

C. Initial test case assignment - step 2

This step is accomplished by the domain-level sampling scheme, which is based on the *Importance Sampling* (IS) approach. Importance sampling is a Monte Carlo method that has been used successfully in many domains [47]. It is an inference method to approximate the computation of the *true* distribution of variables of interest, which in many practical tasks is intractable. The method samples from the true (unknown) distribution, and thus represents the beliefs (i.e., hypotheses) about the state of the system by sets of samples. Each sample is associated with a probability that the belief is true, and at each iteration: (1) the hypotheses are modified to account for changes in the system, (2) the probability of each hypothesis is updated by examining some samples of that hypothesis; and (3) a larger number of samples are drawn from hypotheses with a larger (relative) probability, to be analyzed in the next iteration [48]. The goal is to converge, in few iterations, to the true probability distribution over the set of hypotheses, identifying the ones more likely to be true. We also used this procedure in [49] for the problem of selecting testing techniques adaptively. Referring to *RELAI*, we formulate the problem as follows: samples are the selected test cases in each iteration; the hypotheses are the tester's belief about which subdomain D_i is more likely to improve reliability if test cases are selected from it; the true unknown distribution to approximate is the optimal number of test cases for each subdomain that maximizes reliability.

As first step, importance sampling requires a relatively small initial set of samples. The number of test cases at the first iteration ($T^{(0)}$) is only required to be *much* smaller than the total number of test cases [48], as it is needed only to start the algorithm. If no information is available about the likelihood that some subdomain will contribute more to reliability improvement, the number of test cases may be distributed uniformly among the m subdomains: $T_i^{(0)} = \frac{T^{(0)}}{m}$. In this case, the initial probabilities are $\pi_i^{(0)} = \frac{1}{m}$. An alternative criterion is to distribute $T^{(0)}$ proportionally to the subdomain size. In the following, we opt for this choice and distribute test cases as follows:

$$T_i^{(0)} = T^{(0)} \cdot \frac{|D_i|}{|D|} \tag{5}$$

with initial probabilities being $\pi_i^{(0)} = \frac{|D_i|}{|D|}$

D. Reliability assessment - step 3.(a)

The output of step 2 is the number of test cases to run per subdomain, $T_i^{(0)}$. Step 3.(d) provides such values for the k-th iteration, with $k > 0$. These values ($T_i^{(k)}$) are the inputs of step 3.(a). The goal of the subdomain-level sampling scheme adopted here is to select, for each subdomain, $T_i^{(k)}$ test cases in such a way to get unbiased estimates of the $\varphi_i^{(k)}$ values. These are used both for reliability assessment (Eq. 4), and for the next reliability improvement step (step 3.(d)).

Let us formulate the estimation problem. The parameter of interest for each subdomain is $\varphi_i = \theta_i \sum_{t \in D_i} p_t$, representing the probability of selecting an input from D_i and that such input is a failure point. At this stage, we assume a correct profile estimate, hence: $\hat{p}_t = p_t$, $\forall t \in D$. While p_t values are given for each input, an unbiased estimator of θ_i is the proportion of residual failure points over $|D_i|$ after the test iteration run. The proportion is the probability that an input selected from D_i will fail: $Pr(y_t{=}1)$, with y_t being realizations of a Bernoulli distribution. Therefore, as $|D_i|$ is given, we estimate the total number of residual failure points, namely the initial failure points minus the removed ones. Let us recall that, differently from most studies on reliability assessment, we see the test case selection as a *sampling without replacement* problem, with selection probability regulated by the testing profile distribution $\hat{P}$ – thus *with unequal selection probability* – and the sample unit being the input t (either a failure or a

correct point) with its own probability of selection p_t[4].

There is however a peculiarity in our problem, as the population characteristics change during testing: when a failure is exposed, failure points are removed from future selection, but the corresponding failure region becomes a set of correct input points that are still available for future selection. Therefore, at the end of the test iteration, we have not the same situation we had at the beginning; we have: *i)* a number of undetected failure regions, each with an unknown number of failure points (this is what we want to estimate, let us denote it as Y_R - i.e., "*Residual*"), and *ii)* a number of failure regions that have been detected by selecting exactly one failure point and have been soon eliminated (namely, all its failure points are converted into correct points) - we denote the exposing failure points (exactly one per region) as Y_D, i.e., "*Detected*".

In this problem, the actual estimate we get is not the total number of initial failure points, but the (smaller) sum: $\hat{Y}$ = $\hat{Y}_R$+ Y_D. It represents the total initial number of failure points in an ideal situation where we have (one or) many failure points for the undetected regions but exactly one failure point for each detected failure region. It resembles to the situation at the end of the iteration and is even finer for our purpose, as we are interested in $\hat{Y}_R$. The latter is obtained as: $\hat{Y}_R$ = $\hat{Y}$ - Y_D. Since Y_D is known, the problem reduces to estimate $\hat{Y}$.

Let us set up an unbiased estimator for this problem. We denote with N_i the population size for the subdomain D_i; $T_i^{(k)}$ is the sample size from which we get the estimate; Y_i is the population total to estimate. For this *sampling without replacement with unequal probability of selection* scheme, it would be relatively simple to adopt an unbiased estimator (e.g., the Horvitz and Thompson estimator [50]) only when the sample is of little size (≤ 2). However, when the sample size is greater than 2, we need methods acting also on the sampling scheme itself (and thus on the test case selection strategy). We use the Rao, Hartley and Cochran (RHC) scheme [36], which is a popular sampling method adopted in numerous contexts for its simplicity and practicability. The method steps follow:

1) Suppose we have to execute $T_i^{(k)}$ test cases for the subdomain D_i at iteration k. Each input of the subdomain is associated with a rescaled selection probability $\hat{p}'_t = \hat{p}_t / \sum_{t \in D_i} \hat{p}_t$, so that $\sum_{t \in D_i} \hat{p}'_t = 1$. On this population, referred to one subdomain D_i, with $N_i$units in the population and a sample of size to select of $T_i^{(k)}$ test cases, we can apply the RHC strategy (steps 2, 3, and 4).
2) First of all, divide randomly the N_i units of the population into $g = T_i^{(k)}$ groups, by selecting G_1 units with a *simple random sampling without replacement* for the first group, then G_2 units out of the remaining $(N_i - G_1)$ for the second, and so on. This will lead to g groups of sizes G_1, G_2, ..., G_g with $\sum_{r=1}^{g} G_r = N_i$. The group size is arbitrary, but selecting $G_1 = G_2 = \cdots = G_g$ reduces the variance [36].
3) One test case is then drawn from each of these g groups independently and with a "probability proportional to size" (PPS) method - in our case, according to $\hat{p}'_t$ values.
4) Denote with $\hat{p}'_{t,r}$ the probability associated with the t-th unit in the r-th group, and with $q_r = \sum_{t \in G_r} \hat{p}'_{t,r}$ the sum of probabilities in the r-th group. An unbiased estimator of the population total for D_i is [36]:

$$\hat{Y}_i = \sum_{r=1}^{g} \frac{y_r}{\hat{p}'_r / q_r} \tag{6}$$

where the suffixes $1, 2, \ldots, r$ denote the g units selected from the g groups separately, $y_r = 1$ if the test case led to failure, 0 otherwise. Since $\hat{p}_t = p_t$ (i.e., correct profile estimate), the estimate is unbiased, E[$\hat{Y}_i$]=Y_i, as showed in [36].
5) The residual failure points at the end of the iteration are given by $\hat{Y}_i$ subtracted by the detected (and removed) failure points, that we denote as $Y_{D_i} = \sum_{r \in D_i} y_r$. This leads to: $\hat{\theta}_i^{(k)} = (\hat{Y}_i - Y_{D_i})/N_i$, which is the proportion of residual failure points. It follows that:

$$\hat{\varphi}_i^{(k)} = \hat{\theta}_i^{(k)} \sum_{t \in D_i} \hat{p}_t = \hat{\theta}_i^{(k)} \cdot \hat{p}_i. \tag{7}$$

with $\hat{\varphi}_i^{(k)}$ representing the probability estimate of selecting an input from D_i and that such input is a failure point.
6) In accordance with Equation 4, reliability at iteration k is assessed as:

$$\hat{R}^{(k)} = 1 - \hat{\Phi}^{(k)} = 1 - \sum_{i=1}^{m} \hat{\varphi}_i^{(k)} = 1 - \sum_{i=1}^{m} \hat{p}_i \cdot \hat{\theta}_i^{(k)} \tag{8}$$

and, in this case, being $\hat{p}_t = p_t$, the estimate is unbiased:

$$E[\hat{R}^{(k)}] = 1 - E[\hat{\Phi}^{(k)}] = 1 - \sum_{i=1}^{m} p_i \theta_i = R \tag{9}$$

7) We are also interested in the variance and confidence interval of the estimator. The variance computation of the estimate $\hat{Y}_i$ is provided by the RHC method itself [36], and looks as follows:

$$V(\hat{Y}_i) = \frac{\sum_r G_r^2 - N_i}{N_i(N_i - 1)} \left(\sum_{t=1}^{N_i} \frac{y_t^2}{\hat{p}_t} - Y_i^2 \right) \tag{10}$$

with $\sum_r$ denoting the sum over the g groups; its unbiased estimator is also provided:

$$\hat{V}(\hat{Y}_i) = \frac{\sum_r G_r^2 - N_i}{N_i^2 - \sum_r G_r^2} \left(\sum_{r=1}^{g} q_r \frac{y_r^2}{\hat{p}_r^2} - \hat{Y}_i^2 \right). \tag{11}$$

8) Based on them, and assuming independence among $\hat{\theta}_i^{(k)}$, we can compute the variance for $\hat{R}^{(k)}$ as:

[4]Studies on reliability assessment assume a sampling with replacement and an equal probability of selection among partitions, which are relatively easy to manage. Choosing sampling without replacement with unequal probability of selection is known to be theoretically more efficient than sampling with replacement, at the expense of more complex mathematics. This choice, other than more efficient, reflects more the actual practice whenever the defects are removed during testing, because in reality, test cases should not be repeated [31], once, of course, verified that the defect has been really removed. This applies to our case because the actions that we take on the code are only bug removal actions and, due to assumption 4, the bug removal process is assumed to be perfect.

$$V(\hat{R}^{(k)}) = V(1-\hat{\Phi}^{(k)}) = V(\hat{\Phi}^{(k)}) = \sum_{i=1}^{m} \hat{p}_i^2 V(\hat{\theta}_i^{(k)})$$
$$= \sum_{i=1}^{m} \frac{\hat{p}_i^2}{N_i^2} V(\hat{Y}_i^{(k)} - Y_{D_i}^{(k)}) = \sum_{i=1}^{m} \frac{\hat{p}_i^2}{N_i^2} V(\hat{Y}_i^{(k)}) \quad (12)$$

and:

$$\hat{V}(\hat{R}^{(k)}) = \hat{V}(\hat{\Phi}) = \sum_{i=1}^{m} \frac{\hat{p}_i^2}{N_i^2} \hat{V}(\hat{Y}_i^{(k)}) \quad (13)$$

The corresponding confidence interval at C_R%=100(1-α)% is:

$$CI(\hat{R}^{(k)}) = \hat{R}^{(k)} \pm t_{\alpha/2} \sqrt{\hat{V}(\hat{R}^{(k)})} \quad (14)$$

Expectedly, the variance decreases as more test cases are devoted to the subdomain, and the confidence interval is tighter. The interval estimates of reliability at each iteration are used either for steps 3.(b) or 3.(c), depending on the selected policy. Then, the improvement step takes place.

E. Reliability improvement: probability update - step 3.(d)

The next steps at the end of an iteration is to allocate test cases to subdomain for the next iteration. This is carried out at domain-level by the introduced Bayesian approach, the importance sampling (IS) method. With respect to the other mentioned strategies for allocating test cases to subdomains [4] [3] [2], we do not assume the code being frozen, since detected defects are removed during testing – thus the code changes, and so does the failure rate of each subdomain, making those approaches not applicable in this case. With a problem similar to ours, the approach taken in [27] also uses the Bayesian inference, taking test data to estimate the failure rates and selecting test cases based on them, but assuming a prior knowledge on failure rates (assumed to following a *Beta* distribution), assuming failing test cases to be a very small percentage of total tests, and assuming one failure point removed with one defect removal. To overcome these problems, we decided to follow a sampling-based approach to infer the real distribution of the optimal allocation of test cases, that learns and adapts the selection of samples online and rely exclusively on observed data.

IS has been used also for testing purposes, in order to prioritize mutation operators in mutation testing [48]. We also used it to prioritize the testing technique among a set of techniques. Here, IS is used to prioritize subdomains. The approach foresees to update a probability vector, $\pi^{(k)}$, whose elements represent the probability that an hypothesis is true: in this case, an element represents the probability that selecting tests from a given subdomain has the greatest impact on delivered reliability. Therefore, such values are made proportional to to $\hat{\varphi}_i$, i.e., to the expected runtime failure probability associated with the D_i, which was estimated previously. The update rule is as follows:

$$\pi_i^{(k)} = \gamma \pi_i^{(k-1)} + (1-\gamma) \cdot (1 - \pi_i^{(k-1)}) \cdot \varphi_i^{(k)} \quad (15)$$

The rule tends to penalize those subdomains whose expected impact on failure probability (assessed by φ values) is lower. Larger values will be assigned to those subdomains with higher φ. Moreover, given the same φ, the increase is larger for subdomains that had fewer resources at the previous iteration. $\gamma \in [0,1]$ is a learning factor, which regulates the extent to which the algorithm considers past iterations' results with respect to the current one[5]. The values of $\pi_i^{(k)}$ are finally normalized, since they are probabilities: $\pi_i^{(k)} = (\pi_i^{(k)})/(\sum_{i \in D} \pi_i^{(k)})$. *These probabilities represent the estimate at iteration k of the relative importance of the subdomain D_i.* Starting from $\pi_i^{(k)}$, the number of test cases per subdomain is determined by the *IS procedure*, reported below, modified to target our specific problem. We consider the KLD variant [51] that adapts the number of sample in each iteration to the desired error and confidence. The scheme tends to progressively reduce the number of required samples as more information becomes available, in order to approximate the sought distribution earlier. The number of samples to generate at iteration k in this variant is given by [51]:

$$\eta^{(k+1)} = \frac{1}{2\xi}\chi^2_{\rho-1,1-\delta} \approx \frac{\rho-1}{2\xi}\{1 - \frac{2}{9(\rho-1)} + \sqrt{\frac{2}{9(\rho-1)}} z_{1-\delta}\}^3 \quad (16)$$

where: ξ represents the error between the sampling-based estimate and the true distribution that we want to tolerate; $1-\delta$ is the confidence that we have in this approximation; ρ is the number of subdomains 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; the result, η^{k+1}, is the output number of test cases to execute in the $(k$+1)-th iteration. Considering this number and the probability vector of Eq. 15, the IS procedure is as follows.

IS 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; *//initialization*
end for
for i=2 to m
  $b_i = b_{i-1} + \pi_i^{(k)}$; *//Compute Cumulative Distribution*
end for
//Compute $\eta^{(k+1)}$ *according to Eq. 16*
$r_1 \sim U[0, \frac{1}{\eta^{(k+1)}}]$ *//Draw sample from uniform distribution*
//Distribute test cases to each criterion
i = 1;
for j = 1 **to** $\eta^{(k+1)}$
  while $r_j > b_i$ **do** *//Find the bucket to fill*
    $i = i + 1$;
  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 algorithm first computes the cumulative probability distribution of the subdomains, with probabilities set in descending order. Then, it computes the number of test cases for the next iteration ($\eta^{(k+1)}$ according to Eq. 16), and distributes, in the last *for* loop, test cases to subdomains proportionally to

[5]At the extremes, if $\gamma = 1$, then the online results are not considered and probabilities are fixed at the initial value; if it is 0, results of current iteration are fully considered to update the probabilities; any intermediate value makes sense.

their relative importance (that is represented by the cumulative distribution b_i over subdomains D_i). It is executed until the number of available test cases for that iteration, $\eta^{(k+1)}$, ends. At the end of this step, a new iteration of *RELAI* starts from step 3.(a), and the assigned test cases are run, for each subdomain D_i, according to the RHC scheme illustrated above.

Summarizing, step 3.(a) for *reliability assessment* is carried out by the subdomain-level scheme through RHC procedure, whereas steps 2, and then 3.(d) for *reliability improvement* are run at domain-level with the IS method. Next, we include the impact of the uncertain profile knowledge on these results.

V. Including the Impact of the Error

In the above algorithm, we assumed a correct operational profile as assessed by tester. In practice, having an exact estimate of the profile is unlikely. In this Section, we account for the impact of the profile estimate error on results. Let us define the total error variable $\epsilon = \sum_{t\in D} \varepsilon_t$, with $\varepsilon_t = \hat{p}_t - p_t$, representing the deviation of the real operational profile P from the estimated one $\hat{P}$. Let us first consider how the error impacts the estimate of Y_i, and consequently of φ_i. The RHC estimator for Y_i is, in this case, biased, namely: $E[\hat{Y}_i'] \neq Y_i$. In fact, if $p_t \neq \hat{p}_t$, Equation 6 becomes:

$$\begin{aligned} E[\hat{Y}_i'] &= \textstyle\sum_{r=1}^{g} E[\frac{y_r}{\hat{p}_r/q_r}] \\ &= \textstyle\sum_{r=1}^{g} \sum_{t=1}^{G_r} (p_t/q_r)(\frac{y_t}{\hat{p}_t/q_r}) \\ &= \textstyle\sum_{t=1}^{N_i} y_t \frac{p_t}{\hat{p}_t} = \sum_{t=1}^{N_i} y_t(1-\frac{\varepsilon_t}{\hat{p}_t}) \end{aligned} \tag{17}$$

where we developed the expectation within the summation, and then considered that $N_i = \sum_{r=1}^{g} G_r$. Having a real selection probability p_t different form the estimated one $\hat{p}_t$ leads the estimation to be biased. Specifically, the bias is:

$$\begin{aligned} B_{Y_i}[\hat{Y}_i'] &= E[\hat{Y}_i'] - Y_i \\ &= \textstyle\sum_{t=1}^{N_i} y_t(1-\frac{\varepsilon_t}{\hat{p}_t}) - \sum_{t=1}^{N_i} y_t \\ &= -\textstyle\sum_{t=1}^{N_i} y_t \frac{\varepsilon_t}{\hat{p}_t} \end{aligned} \tag{18}$$

When $p_t = \hat{p}_t$, then $\varepsilon_t = 0$ and the bias is 0.

As consequence, by knowing the bias of $\hat{Y}_i'$, we can compute the bias of reliability estimate as:

$$\begin{aligned} B_R[\hat{R}'] &= E[\hat{R}'] - R \\ &= (1 - E[\hat{\Phi}']) - (1 - \Phi) = \Phi - E[\hat{\Phi}'] \\ &= \textstyle\sum_{i=1}^{m} p_i\theta_i - \sum_{i=1}^{m} \hat{p}_i E[\hat{\theta}_i'] \end{aligned} \tag{19}$$

Thus, the bias depends on $\hat{p}_i = (p_i + \varepsilon_i)$, and on $E[\hat{\theta}_i']$. Recalling that $\hat{\theta}_i$ is computed, in Section IV-D, as $\hat{\theta}_i^{(k)} = (\hat{Y}_i - Y_{D_i})/N_i$, its expected value is:

$$\frac{E[\hat{Y}_i'] - Y_{D_i}}{N_i} = \frac{B_{Y_i}(\hat{Y}_i') + Y_i - Y_{D_i}}{N_i} \tag{20}$$

Putting together Equation 19 and 20, the bias of reliability estimate, under the error of ϵ, is:

$$B_R[\hat{R}'] = \sum_{i=1}^{m} (p_i\theta_i - (p_i + \varepsilon_i)\frac{B_{Y_i}(\hat{Y}_i') + Y_i - Y_{D_i}}{N_i}) \tag{21}$$

The variance of the reliability estimate and its estimator, as well as confidence interval, expressed, respectively, in Equations 12, 13, and 14, are modified, in this case of profile affected by error, by just replacing $\hat{Y}_i$ with $\hat{Y}_i'$, and setting $\hat{p}_i$ = $(p_i + \varepsilon_i)$.

By these equations, the tester can specify the tolerable error for each input t, ε_t, imposing constraints like "the maximum error for input $t = 1$ is ε_t = 0.01, for $t = 2$ is ε_t = 0.05", and so on, in order to obtain the reliability estimate $\hat{R}'$ under this error. Although this is fine for cases in which the granularity of the input t with assigned a probability value p_t is coarse (e.g., we have the same p_t for an entire partition or a functionality), it can be too expensive if we have one p_t value for each single input. Thus, we hereafter specify properties of the error at global level in order to allow tester synthesizing just one constraint on the error. In particular, tester can express the desired maximum error as a triple $\langle \varepsilon_0, \varepsilon_\mu, C_{\varepsilon_0}\% \rangle$, meaning that he wants the error to be: $\varepsilon_\mu - \varepsilon_0 \leq \varepsilon_t \leq \varepsilon_\mu + \varepsilon_0$ in at least $C_{\varepsilon_0}\%$ of cases (e.g., for 99% of inputs). From the Chebichev inequality, it follows that:

$$\begin{aligned} &Pr(\varepsilon_\mu - \varepsilon_0 \leq \varepsilon_t \leq \varepsilon_\mu + \varepsilon_0) \geq 1 - \tfrac{1}{(\varepsilon_0/\sigma_0)^2} = C_{\varepsilon_0}\% \\ &\Rightarrow \sigma_0 = \varepsilon_0\sqrt{1 - C_{\varepsilon_0}\%}. \end{aligned} \tag{22}$$

Eq. 22 tells that the requirement expressed by a tester as desired triple $\langle \varepsilon_0, \varepsilon_\mu, C_{\varepsilon_0}\% \rangle$, can be satisfied by any error distribution with standard deviation σ_0. Without any specific knowledge on the error, ϵ can be supposed to be normally distributed with zero mean (i.e., the tester commits an error either underestimating or overestimating p_t, with equal probability). Zero mean, $\varepsilon_\mu = 0$, allows tester to specify, in a more concise way, the desired error as a pair $\langle \varepsilon_0, C_{\varepsilon_0}\% \rangle$. Note, however, that this assumption is not strictly necessary, as the Chebichev inequality is independent from the distribution. The procedure explained hereafter works also with non-normal distributions and with non-zero mean[6]. Expressing the maximum tolerable error in such a way, we implement the following steps to consider its potential impact:

1) Start from the estimated profile $\hat{P}$ and generate K ideally correct profiles, P_h, with $h = 1, \ldots, K$, so that $p_t = \hat{p}_t + \varepsilon_t$, $\forall\ t \in D$, and ε_t has the characteristic required, namely $\varepsilon_t \leq |\varepsilon_0|$ in at least $C_{\varepsilon_0}\%$ of the cases. P_h is generated in the following way:
   a) for each t, add an error ε_t to $\hat{p}_t$, obtaining $p_t = \hat{p}_t + \varepsilon_t$, drawing it from a normal distribution, with zero mean and standard deviation σ_0. This guarantees that $\varepsilon_t \leq |\varepsilon_0|$ in $C_{\varepsilon_0}\%$ of cases.
   b) Rescale the distribution so that $0 \leq p_t \leq 1\ \forall t \in D$ and $\sum_t p_t = 1$; thus $p_t' = 0$ if $p_t \leq 0$ and then $p_t' = p_t / \sum_t p_t\ \forall t \in D$. We call P_h the new distribution made up of p_t' probability values.
2) The new profile P_h represents an ideally correct profile, deviating from $\hat{P}$ by the required error $\langle \varepsilon_0, C_{\varepsilon_0}\% \rangle$. With this, we assess reliability $\hat{R}_h$ assuming P_h as the correct

[6]Tester might want to specify a non-normal distribution or an error biased by a value $\varepsilon_\mu \neq 0$, e.g., if he has some knowledge on the characteristics of the error committed in the past. Zero-mean normal error is assumed for simplicity and for its suitability to represent a non-systematic random error.

profile (again, by Eq. VII-B), and the offset of the real estimation from it: $\Delta_{(\hat{R}_h,\hat{R})} = \hat{R}_h - \hat{R}$. It represents the impact on reliability estimate accuracy of the profile error random variable in one specific realization.

Repeating this procedure for $h = 1, \ldots, K$ provides us with a distribution of $\Delta_{(\hat{R}_h,\hat{R})}$ over the K possible ways of deviating from a correct profile. Tester selects the expected value of this distribution: $\bar{\Delta}_{(\hat{R}_h,\hat{R})} = E[\Delta_{(\hat{R}_h,\hat{R})}]$, using it as offset deviation prediction. The number of repetitions K determines the accuracy of such prediction. We compute K by the bootstrap method, in order to be independent from the distribution selected to generate the profile. It works as follows: given a set of repeated measures, they are randomly sampled M times with replacement; the measure is computed for each sample to obtain an estimate of its statistical distribution and of its C% confidence interval. The number of repetitions K is considered sufficient if the error margin of the estimated measure (i.e., the half-width of the confidence interval) is less than $r\%$. We adopt common values, i.e.: C%=99%, $r\%$ = 5%. In our experiment, K=50 was far sufficient to satisfy the criterion. Note that creating the modified profile is a fully automatic operation requiring the generation of $|D|$ random numbers to add to the p_t values, with a negligible cost in terms of execution time, increasing linearly with K.

Summarizing, the output of this procedure is: under a maximum acceptable profile error as threshold, the estimate of reliability ($\langle \hat{R}$, CI($\hat{R}$)$\rangle$ computed as in Section IV) is affected by a deviation $\bar{\Delta}_{(\hat{R}_h,\hat{R})}$ lying within the confidence interval CI($\bar{\Delta}_{(\hat{R}_h,\hat{R})}$) in 99% of cases. Thus, tester has an indication of the extent to which the operational profile error impacts the reliability assessment accuracy. The offset can be used to adjust the reliability estimate, or, in general, to take more informed decisions (e.g., understanding the level of trust we can place on reliability assessment, thus deciding if keeping on testing or not; understanding the severity of the profile error, if it is really acceptable or we should improve it). The offset estimate is seamlessly used in both stopping policies of the algorithm illustrated above.

VI. Evaluation

A. Experiments Design

The objectives of the experiments are to evaluate the following three benefits that *RELAI* is expected to bring: *i)* delivering higher reliability than other techniques, given the same testing budget (*Policy 1*) or achieving a predefined reliability target with less test cases (*Policy 2*); *ii)* providing, at the same time, an accurate estimate of delivered reliability; *iii)* assessing the impact of the error committed in estimating the profile on the reliability estimate accuracy. The evaluation is designed on a set of subject programs considered under several different scenarios. We consider the following factors: the *subject program*, the number of *test cases* to run (*Policy 1*), or, conversely, the *reliability target* to achieve (*Policy 2*), the *operational profile*, and the *testing technique*. Based on factors' levels defined hereafter, a full design is planned according to a *Design of Experiment* approach. Each treatment over a program is a testing scenario with a given number of test cases, an operational profile, and a testing technique. Since *RELAI* has the unique feature of pursuing the double-objective of improving and assessing reliability in the same testing session, we cannot compare it, in one single experiment, with any existing technique, whose goal is either to improve or to assess reliability. Also, the third objective (assessing the impact of profile error) requires an additional experiment. Thus, we consider, in total, four separate experiments (named Experiment 1, 2, 3 and 4): experiments 1 and 2 to evaluate the trade-off between the *improvement* of delivered reliability and the number of executed test cases (under Policy 1 and 2), where *RELAI* is compared against other "reliability improvement" techniques (namely, techniques that detect and remove defects during testing); experiment 3 is to evaluate the reliability *assessment* accuracy, where *RELAI* is compared against techniques conceived for "reliability assessment" (i.e., acting on frozen code); experiment 4 runs only *RELAI* with operational profiles simulated as "erroneous" profiles to evaluate the ability of predicting the impact of the inaccurate profile. After that, we also run a sensitivity analysis on parameters of the algorithm.

B. Factors: Testing Scenarios

Subject Programs

Techniques are applied to four programs, taken from the SIR repository [52]: *Make*, *SIENA*,*Grep*, and *NanoXML*, whose characteristics are in Table I. The programs have different applicative targets and characteristics - there are two C programs and two Java program, with size ranging from $6K$ to $35K$ *LoC*. *Make* is the well-known Unix build utility; the version used is 3.79. *SIENA* (Scalable Internet Event Notification Architecture) is an framework for constructing event notification services; the version used is 1.15. *Grep* is the command-line utility to search for lines matching a regular expression; the version used is 2.4. *NanoXML* is a simple XML parser for Java; the version used is 2.2.

The programs have the availability of a limited number of test cases generated by the category-partition method [53] via TSL (Test Specification Language). The provided test suites have been, in any case, enlarged, by generating test cases by means of the available TSL specifications. 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 [53]. The final number of test cases is in Table I.

Programs are available with faults seeded inside; however, since faults from the SIR repository are conceived for regression testing purpose, they might be placed only in specific locations identifying the changes from one version to another. Thus, we ignored those faults and injected faults according to the G-SWFIT technique [54], [55], a source-code level approach that we also adopted in our previous work [56]. G-SWFIT technique exploits a set of fault operators derived from the well-known Orthogonal Defect Classification (ODC) [57]. We considered the same distribution as the one actually observed in field studies about the presence of ODC fault types into programs [55]. An automatic injection tool based on

G-SWFIT is used (SAFE - SoftwAre Fault Emulation) [58], developed by our research group, which, parsing the code, automatically identifying suitable locations where each type of fault can be injected [54]. The number of faults injected per program is given in Table I. With such numbers of faults and test suite sizes, the starting reliability in each program are approximately 0.84 for Make, 0.88 for NanoXML, 0.92 for Grep, 0.96 for SIENA (the exact value will depend on the operational profile).

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.

Number of test cases and reliability levels

The number of total available test cases per program is in Table I, and represents the space from which test cases can be selected. To evaluate techniques performance under different values of the available testing budget (*Policy 1*), we consider 15 points ranging from T = 100 to T = 800 test cases (T = 100, T = 150, T = 200, ..., T = 800). Similarly, we consider 10 levels of reliability target to achieve (*Policy 2*), ranging from 0.981 to 0.995 (R = 0.981, R = 0.981, ..., R = 0.995). Each treatment is run, in either *Policy 1* or *Policy 2*, with a fixed value of such levels.

Operational Profile

For evaluation purposes, we do not focus on any specific operational profile; rather, profiles are generated randomly. Specifically, in Experiment 1, 2, and 3 (namely, reliability improvement and reliability assessment) we use three randomly generated operational profiles denoted as P_1, P_2, and P_3, supposed to be correct, true, profiles (i.e., no error in the profile estimate by the tester). In particular, to generate a profile, we assign a value between 0 and 1 to each input test case, representing the probability with which it would actually occur in operation. Each value is obtained by a uniform random number generation in [0; 1], then normalized over the sum of all values to add up to 1. Experiment 4 concerns with the impact of incorrect profiles: in this case, we generate one profile supposed to be the correct one, denoted as P_C, Then, we generate three random profiles that meet the required condition on ϵ, with a maximum tolerable error set to $\varepsilon_0 = 0.01$, $\varepsilon_\mu = 0$, and $C_{\varepsilon_0} = 99\%$. Specifically, we repeatedly generate profiles and compute the error ϵ with respect to P_C (by taking the sum over the single $\varepsilon_t = \hat{p}_t - p_t$ values), and discard the profile until the required condition on ϵ is not met. The three profiles generated in this way are named $\hat{P}_1, \hat{P}_2, \hat{P}_3$, to be distinguished from the previous ones.

Testing Techniques

As mentioned, we compared *RELAI* against different techniques in different experiments, depending on the objective. Techniques are all black-box techniques suitable for system and acceptance testing purposes. *RELAI* performance regarding reliability improvement (Experiment 1 and 2) is compared against:

Fixed Size Candidate Set - Adaptive Random Testing (**FSCS-ART**) [31]: this is one of the best variants of adaptive random testing (introduced in Section II), that uses a distance-based selection criterion to evaluate a *fixed set* of randomly generated test case candidates each time. This has been empirically showed to be one of the most effective ART criteria [59].

Evolutionary Adaptive Random Testing (**EAR**) [33]: this is an evolution of ART that uses a genetic algorithm to select the best test case in terms of maximum input-based distance from all the previous test cases.

Operational testing for reliability improvement (**OPv1**): in operational testing, inputs are selected according to the estimated operational profile; this technique can be used for either improving delivered reliability [23], [11] (by detecting and removing faults during testing) or for reliability assessment [60], [1], [2] (by detecting but not removing defects). In Experiment 1 and 2, we use the reliability improvement version, and call it *OP variant 1* (OPv1).

RELAI performance regarding reliability assessment accuracy is compared with:

Adaptive Testing-Gradient Descent (**AT-GD**) [3]: this is a state-of-the-art technique, evolving from conventional adaptive testing [60], [1], [2] which selects the next test case in a way to maximize the reduction of the reliability estimator variance, based on the negative of the gradient of reliability estimator variance at a given state.

Operational testing for reliability assessment (**OPv2**): as mentioned, each input is selected according to the estimated operational profile; we use the variant of operational testing acting on frozen code, similarly to previous works on reliability assessment (e.g., [2], [3], [4]).

Uniform random testing (**Random**): it is the conventional technique in which test cases are selected randomly according to a uniform distribution from the input space [61]. Again, we use it to detect but not remove defects for reliability assessment purposes, as in [2]. For this technique, a uniform operational profile distribution is used (i.e., all inputs the same probability) instead of P1, P2, or P3.

As for *RELAI*, we set $\gamma = 0.5$ and $\xi = 0.1$. A sensitivity analysis is then performed (Section VIII) on such parameters.

RELAI and *AT-GD* make use of partitions[7]. We partition the domain into 4, 5, and 6 classes when we use, respectively, Profile P1, P2, and P3. Without loss of generality (see assumption 5 in Section IV-A), we use, as partitioning strategy, an approach based on the expected occurrence probability, thus dividing the test suite considering the potential impact of test cases on delivered reliability. Specifically, the range of expected occurrence probabilities $[p_{min}; p_{max}]$ is divided into

[7]We used the generalized versions of *OP* and *Random testing*, since we use profiles specifying a probability value for each input – taking the sum of probabilities over partitions, we fall in the more specific case of specifying a profile specifying a global probability value for each partition (cf. with footnote 3)

intervals of equal size (4, 5, or 6); the inputs are assigned to classes based on their p_t value (therefore, classes contain a different number of inputs). This implies that each time the *profile* is generated, the partitions will change too, allowing to evaluate the approach under various partitioning.

C. Procedure

The j-th testing scenario, S_j, includes these steps:

1) Select the subject program.
2) Select the total number of test cases T^* as budget for Experiment 1, 3, and 4; select the reliability level R^* to achieve for Experiment 2;
3) Select the operational profile (uniform profile for random testing).
4) Select one of the techniques under comparison (except for Experiment 4, which tests only *RELAI*).
5) Submit the test case to the program, selected according to the chosen strategy.
6) Observe if it generates a failure or not.
7) If a failure has occurred, the action depends on the technique: for *RELAI, EAR, FCFS, OPv1*, remove the fault[8]; for *AT, OPv2*, and *Random testing* just record its occurrence.
8) Repeat from step 5 until T^* test cases are executed (Experiment 1, 3, and 4) or until R^* is achieved (Experiment 2).
9) At the end of the session, compute the reliability metrics, presented hereafter, useful for evaluation.

D. Evaluation criteria

Given the above scenario, we have: 15 points for number of test cases $\times$ 4 programs $\times$ 3 profiles $\times$ 4 techniques $\times$ 3 Experiments = 2160 treatments, plus further 180 treatments in Experiment 4 regarding just *RELAI*, these are 2340 treatments. Since testing selection criteria are probabilistic, running the same testing scenario twice does not necessarily yield the same result. To draw statistically valid conclusions, we replicate each treatment 100 times[9]. The following metrics are computed depending on the experiment:

1) *Delivered reliability improvement*. As for Experiment 1, at the end of each run, we compute the actual delivered reliability after testing as: $R = 1 - \sum_{i \in Z} p_i$ where Z is the set of failure points of the residual faults, and p_i is their probability of occurrence in operation, according to the selected profile. Failure points correspondence with faults are known by preliminarily running all the test suite against faulty program versions (i.e., by faults matrices). Given the jth treatment, we evaluate the mean delivered reliability and its sample variance:

$$\begin{aligned} Mean(R_j) &= \tfrac{1}{100} \textstyle\sum_{r=1}^{100} R_{r,j} \\ Var(R_j) &= \tfrac{1}{100-1} \textstyle\sum_{r=1}^{100} (R_{r,j} - Mean(R_j))^2 \end{aligned} \tag{23}$$

where $R_{r,j}$ is the true reliability for the r-th run of the j-th treatment. To compare techniques with respect to reliability improvement, the absolute difference between mean reliability values is insufficient to indicate the actual gain: if a certain absolute improvement is obtained over a low reliability value, the gain is marginal; whereas if the same difference is observed on a very high reliability, it represents a high gain. To account for this, we compute the metric G representing the gain of technique a over b (supposing that reliability delivered by a is the greatest one) *with respect to the maximum potential gain that could be attained:*

$$G(a,b)\% = \frac{Rel(a) - Rel(b)}{1 - Rel(b)} \cdot 100 \tag{24}$$

where denominator is the maximum gain achievable by b. Given the same absolute difference, the higher the b's reliability, the higher the gain. G is evaluated on both reliability means (G_μ) and medians (G_{Mdn}).
In order to test *Policy 2* (Experiment 2), we consider the mean and variance of the number of test cases to be executed in order to achieve a given reliability level in a treatment j (for a given program, profile, and testing technique):

$$\begin{aligned} Mean(T_j) &= \tfrac{1}{100} \textstyle\sum_{r=1}^{100} T_{r,j} \\ Var(T_j) &= \tfrac{1}{100-1} \textstyle\sum_{r=1}^{100} (T_{r,j} - Mean(T_j))^2 \end{aligned} \tag{25}$$

where $T_{r,j}$ is the number of required test cases in the r-th run of the j-th treatment. endequation
2) *Accuracy of reliability estimate*. As for Experiment 3, at the end of each run r, we compute the reliability estimate given by the technique under test, $\hat{R}_{r,j}$. Then, the mean value over runs, its sample variance, and the mean squared error (MSE) are computed, the latter giving the accuracy of the estimate with respect to the true reliability[10]:

$$\begin{aligned} Mean(\hat{R}_j) &= \tfrac{1}{100} \textstyle\sum_{r=1}^{100} \hat{R}_{r,j} \\ Var(\hat{R}_j) &= \tfrac{1}{100-1} \textstyle\sum_{r=1}^{100} (\hat{R}_{r,j} - Mean(\hat{R}_j))^2 \\ MSE(\hat{R}_j) &= \tfrac{1}{100} \textstyle\sum_{r=1}^{100} (\hat{R}_{r,j} - R_{r,j})^2 \end{aligned} \tag{26}$$

3) *Accuracy of profile estimation error impact by RELAI*. Experiment 4 is run considering a profile assumed to be correct (P_C) and three incorrect profiles, $\hat{P}_1, \hat{P}_2, \hat{P}_3$. For

[8]Note that for one failure, it might be happen that there are more faults that can be removed (failure regions are not actually disjoint); in our case we choose to remove one of them randomly (by a uniform distribution between 1 and the number of faults corresponding to the activated failure regions). As shown later, we repeat the execution of a testing session several times to have statistical significance; thus the bias of removing one fault instead of another, if there were such a case, is minimized by random selection as the number of repetitions increases.

[9]The 100 repetitions for each treatment were sufficient to satisfy the criterion of the error margin of 5% (cf. with Section V).

[10]Note that, since for reliability assessment techniques (all but *RELAI*) the code is frozen, the failure points (hence the true reliability) in this case are known from the beginning, they do not change during testing; the true reliability is the same across runs of each treatment: $R_{100,j} = R_j$). This is not true for *RELAI*: the latter, as discussed, assesses reliability while improving it, foreseeing the removal of detected faults during testing – hence the final reliability will be, in general, different in each run. In both cases, the metric of interest is the difference of the estimated reliability with the true one (whatever the true reliability value is), namely the MSE.

each run of *RELAI*, we compute again the true delivered reliability and the offset, denoted as $\Delta_{(\hat{R},R)_{r,j}} = \hat{R}_{r,j} - R_{r,j}$. It represents the reliability estimation accuracy in presence of profile error.

We need to evaluate the *RELAI* ability to predict the impact of the profile error on the accuracy. In fact, for profiles $\hat{P}_1, \hat{P}_2, \hat{P}_3$, the offset will depend on the estimator's error *plus* the profile error impact. *RELAI* aims to predict this impact by the $\bar{\Delta}_{(\hat{R}_h,\hat{R})}$ value as computed in Section V, representing the potential deviation of the assessment due to profile error; we wish to evaluate how much accurate this prediction is. To this aim, we first consider the difference between offsets obtained under the correct profile ($\Delta' = \Delta_{(\hat{R},R)_{[P=P_C]}}$) and offsets (for the same program and test size) under inaccurate profiles ($\Delta'' = \Delta_{(\hat{R},R)_{[P=\hat{P}_{1,2,3}]}}$). This difference $\Delta' - \Delta''$ assesses the impact due to $\hat{P}_i$ (i.e., the share of the offset caused by the profile error); this is the value the procedure aims to predict. Then, we compute the predictions ($\bar{\Delta}_{(\hat{R}_h,\hat{R})}$) as foreseen by the presented procedure, and count how many times their confidence interval contains the actual difference $\Delta' - \Delta''$. The metrics computed for a given treatment are:

$$Mean(\bar{\Delta}_{(\hat{R}_h,\hat{R})_j}) = \frac{1}{100}\sum_{r=1}^{100} \bar{\Delta}_{(\hat{R}_h,\hat{R})_{r,j}}$$

$$hit\Delta_{r,j} = \begin{cases} 1 & if\ (\Delta' - \Delta'') \in CI(Mean(\bar{\Delta}_{(\hat{R}_h,\hat{R})_j})) \\ 0 & otherwise \end{cases}$$

$$hit\Delta_j\% = \frac{hit\Delta_{r,j}}{100} \cdot 100 \tag{27}$$

where $Mean(\bar{\Delta}_{(\hat{R}_h,\hat{R})_j})$ is the mean of predictions, and $hit\Delta_j\%$ is the percentage of times in which the actual difference is within the predicted difference confidence interval.

VII. Result

A. Comparison in terms of delivered reliability vs test cases trade-off

This Section targets the first evaluation criterion, comparing the three considered techniques in terms of delivered reliability, given a fixed number of tests (policy 1), and of test cases required to get a fixed reliability target (policy 2).

1) Experiment 1. Delivered reliability with a fixed test budget: Table II reports information about profiles and partitions for this experiment, showing, for each profile and program, the range $[p_{min}; p_{max}]$, the number of test cases per partition, and the sum of occurrence probability values over a partition D_i ($p_i = \sum_{t \in D_i} p_t$), denoting the probability that an input will be selected from that partition.

Results are in Figure 2a-2l, depicting the sample mean of reliability achieved by each technique in each of the 12 program/profile pair scenarios, with respect to an increasing number of executed test cases. Performance of *RELAI* is significantly higher than the other techniques in all the experimented scenarios. The greatest differences are with 100 test cases, wherein *RELAI* is able to deliver a high reliability soon, while the other techniques need, in almost any case, the double of test cases to achieve the same reliability level. The absolute difference between reliability delivered by *RELAI* and the other techniques ranges from a minimum (with 800 test cases) of 3.60E-4 (profile P2, *Grep*, with respect to *EAR*) to a maximum of 0.0367 (profile P3, *Make*, with respect to *OPv1*) with 100 test cases. In terms of *Gain* index, the difference ranges from G_μ = 21.76% (profile P1, *Grep*, 100 test cases, with respect to *EAR*), to G_μ = 90.48% (profile P3, *Grep*, 800 test cases, with respect to *OPv1*). It is interesting to note that, in the average, the gain G_μ obtained in scenarios with 100 test case are 44.22%, 49.28%, and 50.46%, while the gain obtained in scenarios with 800 test cases is 70.17%, 80.63%, 79.52%: this means that, although the absolute differences are higher in scenarios with 100 test cases, the gain with respect to the maximum achievable improvement is more relevant on the highest reliability values (800 test cases). The ability of improving already high reliability is important especially for highly critical systems. Mean gain values for each program/profile pair are reported in Figure 3. The bar graph displays the mean gain over the number of run test cases in each scenario, with error bars denoting the minimum and the maximum gain achieved in the treatments of a pair program/profile. The mean gains G_μ of *RELAI* over all scenarios are: **66.98%**, **74.96%**, and **74.10%** with respect to *EAR*, *OPv1*, and *FSCS*, respectively. Considering medians instead of means, the gains G_{Mdn} are: **69.58%**, **77.18%**, **76.09%** in the three cases. Looking at the other techniques, *EAR* behaves better than *OPv1* and *FSCS*, while the latter two are roughly equivalent in terms of delivered reliability.

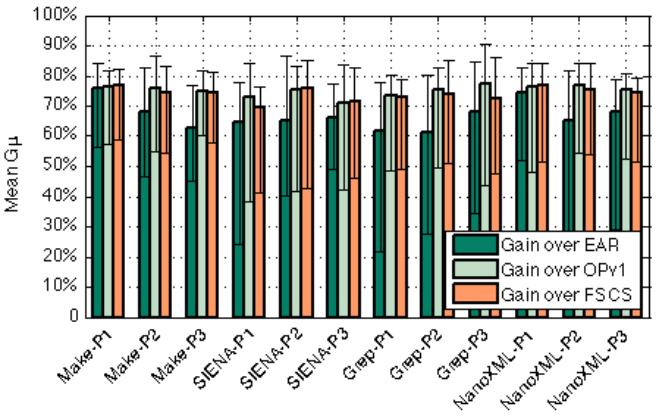


Fig. 3: Mean gain index per scenario (and min-max range). *The maximum gain of RELAI is over OPv1 in the P3-Grep case with 800 test cases: 90.48%*

Figure 4 reports synthetically the sample variances of all the treatments over the 100 repetitions, with mean sample variances and the minimum-maximum range in each program/profile scenario. Except few cases (*SIENA* scenarios) where variances are very similar, the other scenarios highlight that results provided by *RELAI* are more stable than the others. In particular, sample variances of *RELAI* are confirmed to be considerably lower than the other techniques, with an average, over the 12 scenarios, of 9.93E-6, against 3.24E-5, 3.07E-5, and 3.03E-5 of *EAR*, *OPv1* and *FSCS*, respectively.

		Profile 1	*Profile 2*	*Profile 3*
Make	$[p_{min}; p_{max}]$	[1.56E-8 ; 2.19E-4]	[4.83E-8 ; 2.17E-4]	[1.80E-8 ; 2.18E-4]
	Test cases per partition	2406; 2289; 2292; 2251	1865; 1866; 1839; 1832; 1836	1552; 1536; 1622; 1544; 1481; 1503
	$\sum_t p_t$ per partition	6.66E-2; 1.88E-1; 3.13E-1; 4.31E-1	4.04E-2; 1.21E-1; 2.00E-1; 2.78E-1; 3.59E-1	2.81E-2; 8.37E-2; 1.47E-1; 1.97E-1; 2.42E-1; 3.01E-1
SIENA	$[p_{min}; p_{max}]$	[1.49E-8 ; 2.94E-4]	[1.27E-8 ; 2.90E-4]	[1.19E-8 ; 2.91E-4]
	Test cases per partition	1741; 1728; 1726; 1651	1336; 1412; 1344; 1344; 1410	1163; 1131; 1169; 1053; 1136; 1194
	$\sum_t p_t$ per partition	6.38E-2; 1.90E-1; 3.17E-1; 4.27E-1	3.875E-2; 1.23E-1; 1.95E-1; 2.72E-1; 3.69E-1	2.78E-2; 8.14E-2; 1.42E-1; 1.79E-1; 2.49E-1; 3.18E-1
Grep	$[p_{min}; p_{max}]$	[2.69E-8; 2.85E-4]	[3.05E-10 ; 2.82E-4]	[4.41E-8 ; 2.81E-4]
	Test cases per partition	1744; 1806; 1775; 1716	1367; 1456; 1378; 1437; 1403	1154; 1160; 1193; 1171; 1138; 1225
	$\sum_t p_t$ per partition	6.13E-2; 1.93E-1; 3.16E-1; 4.28E-1	3.83E-2; 1.23E-1; 1.95E-1; 2.85E-1; 3.57E-1	2.70E-2; 8.28E-2; 1.39E-1; 1.93E-1 ; 2.40E-1; 3.16E-1
Nano	$[p_{min}; p_{max}]$	[3.35E-8 ; 2.81E-4]	[3.38E-8 ; 2.86E-4]	[1.12E-8 ; 2.84E-4]
	Test cases per partition	1753; 1753 ; 1790; 1781	1432; 1455; 1449; 1370; 1371	1205; 1195; 1157; 1189; 1147; 1184
	$\sum_t p_t$ per partition	6.30E-2; 1.84E-1; 3.14E-1; 4.37E-1	4.12E-2; 1.25E-1; 2.06E-1; 2.74E-1; 3.52E-1	2.84E-2; 8.52E-2; 1.36E-1; 1.96E-1; 2.44E-1; 3.08E-1

TABLE II: Operational profiles for Experiment 1

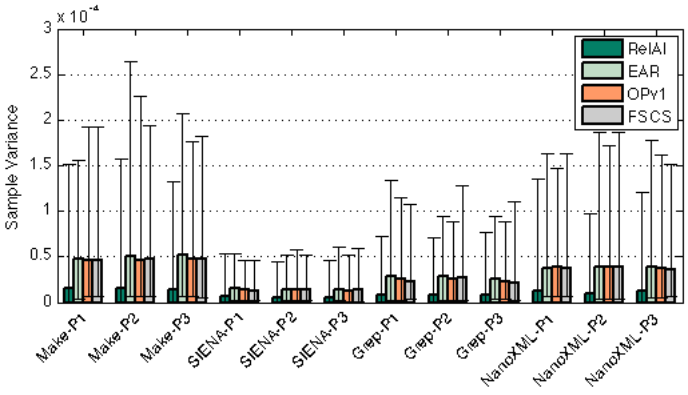


Fig. 4: Mean sample variance per scenario of delivered reliability (and min-max range). *Minimum variance is RELAI in SIENA-P3 under 800 test cases: 5.50 E-08*

For statistical significance, one-way analysis of variance (ANOVA) test is conducted (significance level 0.01). We test the null hypothesis that delivered reliability by two techniques do not differ. If the hypothesis is rejected, a *post hoc* analysis follows to detect the techniques that differ significantly. We first test the properties of data, namely the normality of residuals and homoscedasticity of variances, in order to determine the type of ANOVA to apply. The *Kolmogorov-Smirnov-Lillefors* (KSL) test is run to verify normality of residuals; the null hypothesis of data coming from a normal distribution is rejected at $p\text{-}value < 0.001$. Homoscedasticity is verified by the *Levene*'s test, being it less sensitive to non normality. The null hypothesis of variances being homogeneous is also rejected at $p\text{-}value < 0.001$. Thus, we adopt the *Friedman*'s test, a non-parametric test for repeated-measures data robust to non-normality and heteroscedasticity, to detect if at least one difference among reliabilities delivered by techniques. The hypothesis of no difference among techniques is rejected at $p\text{-}value < 0.001$.

To figure out the differences among compared techniques, we run a *post hoc* analysis, by using the *Nemenyi* test [62], a powerful test for pairwise comparisons after a non-parametric ANOVA [63]. The test uses the *critical difference* (CD): two levels are significantly different if the corresponding average ranks differ by at least CD = $q_\alpha \sqrt{k(k+1)/6N}$, where q_α values are based on the Studentized range statistic divided by $\sqrt{2}$, and adjusted according to the number of comparisons[11], k is the number of levels compared, N is the sample size. Table III lists the results:

Pairwise Comparison: *p-values*				
	RELAI	*EAR*	*OPv1*	*FSCS*
RELAI	-	**9.14 E-20**	**2.84E-33**	**1.41E-31**
EAR	-	-	0.0436	0.0939
OPv1	-	-	-	0.9915

TABLE III: Comparison for *delivered reliability*. Text in boldface indicates that the difference is significant at least at 0.01. The ranking is: *RELAI, EAR, FSCS, OPv1*

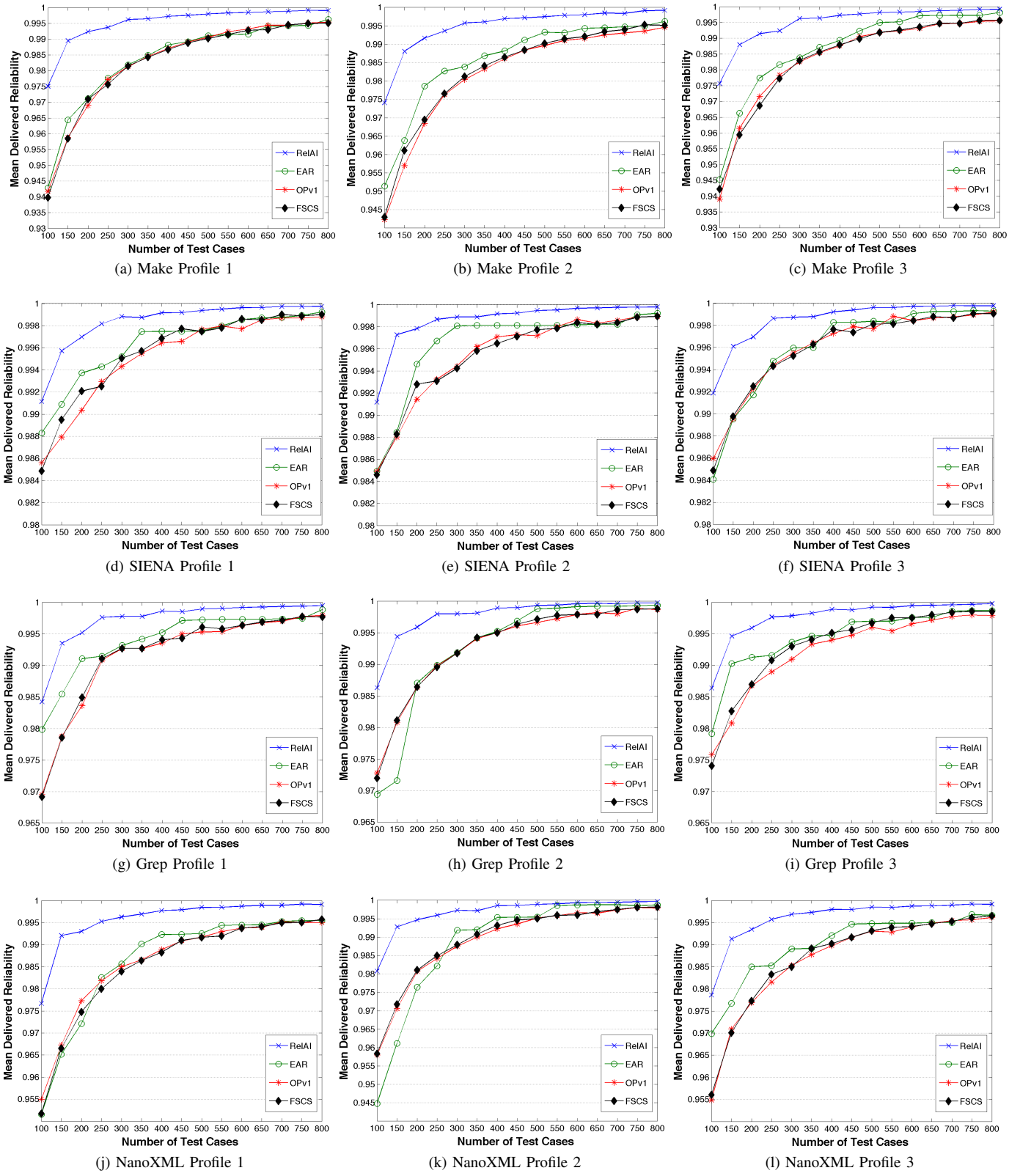
Results tell that the reported differences between *RELAI* and the others is by far significant; the difference between *EAR* (the second best technique) and *OPv1* is significant only at 95% of confidence, while *EAR-FSCS* is significant only at 90%; OPv1 and FSCS are statistically equivalent to each other.

As final remark, it is worth noting that, while *EAR*, *FSCS*, and *OPv1* are not able to provide an estimate of the achieved reliability, *RELAI* is also able to estimate the reliability delivered at the end of testing. In order to evaluate the characteristic, we have run a further experiment (*Experiment 3*) to compare *RELAI* with techniques conceived exclusively for reliability assessment. Results of this experiment are in Section VII-B.

2) Experiment 2. Number of required test cases under fixed reliability levels: This experiment is specular to the previous one, and tests the *RELAI* performance in reducing the number of required test cases to attain the fixed reliability level. In particular, 10 reliability targets and running, for each treatment, the technique under evaluation until the target is not achieved. The output performance metric is the average number of necessary test cases to attain the target. Operational profiles features for this experiment are in Table IV

Figure 5a-5l show the results in each of the 12 program/profile pair scenarios. Performance of Experiment 1 are confirmed; form these graphs, it is more evident the gain of *RELAI* on high reliability values, in which the gap with the other techniques becomes larger and larger – a behavior in line with the *Gain* index. Figure 6 reports again the mean sample variances over the scenarios, and the corresponding *min-max* range. Sample variances of *RELAI* are confirmed to

[11] As the family-wise error rate is already controlled by considering q_α, no other multiple comparison protection procedure is needed.



(a) Make Profile 1 (b) Make Profile 2 (c) Make Profile 3

(d) SIENA Profile 1 (e) SIENA Profile 2 (f) SIENA Profile 3

(g) Grep Profile 1 (h) Grep Profile 2 (i) Grep Profile 3

(j) NanoXML Profile 1 (k) NanoXML Profile 2 (l) NanoXML Profile 3

Fig. 2: Sample mean of delivered reliability *vs* number of test cases

be lower than the other techniques, with an average, over the 12 scenarios, of 1.05E-5, against 3.27E-5, 3.08E-5, 3.12E-5 of *EAR*, *OPv1* and *FSCS*, respectively.

The same statistical tests as the former are adopted in this case (data are non-normal and heteroscedastic at $p\text{-}value < 0.001$ for both KSL and Leven's test); the Friedman

		Profile 1	*Profile 2*	*Profile 3*
Make	$[p_{min}; p_{max}]$	[1.47E-8 ; 2.13E-4]	[1.61E-8 ; 2.18E-4]	[2.34E-9 ; 2.18E-4]
	Test cases per partition	2223; 2269; 2347; 2399;	1903; 1845; 1834; 1844; 1812	1524; 1579; 1573; 1543; 1500; 1519
	$\sum_t p_t$ per partition	5.9E-2; 1.83E-1; 3.13E-1; 4.46E-1	4.07E-2; 1.22E-1; 2E-1; 2.82E-1; 3.56E-1	2.78E-2; 8.58E-2; 1.43E-1; 1.96E-1; 2.45E-1; 3.03E-1
SIENA	$[p_{min}; p_{max}]$	[1.38E-9 ; 2.93E-4]	[1.34E-7 ; 2.91E-4]	[1.22E-8 ; 2.91E-4]
	Test cases per partition	1728; 1671; 1741; 1706	1362; 1362; 1393; 1320; 1409	1156; 1094; 1147; 1152; 1174; 1123
	$\sum_t p_t$ per partition	6.18E-2; 1.83E-1; 3.18E-1; 4.36E-1	4.01E-2; 1.2E-1; 2.03E-1; 2.69E-1; 3.69E-1	2.85E-2; 8.06E-2; 1.39E-1; 1.96E-1; 2.56E-1; 3E-1
Grep	$[p_{min}; p_{max}]$	[3.92E-8 ; 2.81E-4]	[6.06E-8 ; 2.86E-4]	[1.78E-8 ; 2.83E-4]
	Test cases per partition	1699; 1763; 1773; 1806	1445; 1450; 1348; 1385; 1413	1173; 1168; 1177; 1135; 1186; 1202
	$\sum_t p_t$ per partition	5.87E-2; 1.88E-1; 3.1E-1; 4.43E-1	4.22E-2; 1.24E-1; 1.93E-1; 2.77E-1; 3.63E-1	2.74E-2; 8.25E-2; 1.38E-1; 1.87E-1; 2.52E-1; 3.13E-1
Nano	$[p_{min}; p_{max}]$	[4.7E-8 ; 2.84E-4]	[2.46E-8 ; 2.81E-4]	[3.14E-8 ; 2.79E-4]
	Test cases per partition	1789; 1783; 1746; 1759	1404; 1426; 1371; 1431; 1445	1090; 1163; 1217; 1227; 1213; 1167
	$\sum_t p_t$ per partition	6.27E-2; 1.91E-1; 3.09E-1; 4.37E-1	4E-2; 1.2E-1; 1.92E-1; 2.82E-1; 3.66E-1	2.52E-2; 8.13E-2; 1.41E-1; 1.99E-1; 2.54E-1; 2.99E-1

TABLE IV: Operational profiles for Experiment 2

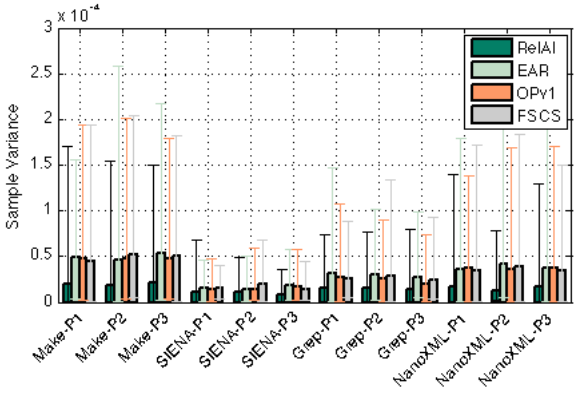


Fig. 6: Mean sample variance per scenario of required number of test cases (and min-max range). Minimum variance is RELAI in SIENA-P3 under 800 test cases: 2.06E-08

Pairwise Comparison: *p-values*				
	RELAI	*EAR*	*OPv1*	*FSCS*
RELAI	-	**2.96E-22**	**2.70E-106**	**1.56E-61**
EAR	-	**-**	**4.42E-31**	**8.40E-10**
OPv1	-	**-**	**-**	**3.65-06**

TABLE V: Comparison for *number of test cases*. Text in boldface indicates that the factor is significant at least at 0.01. The ranking is: *RELAI, EAR, FSCS, OPv1*

test provides again a $p\text{-}value < 0.001$, namely the technique has a significant impact on the number of test cases needed to achieve a given reliability level. Table V reports the pairwise difference $p\text{-}values$. In this case, all the differences turned to be significant. The ranking of the best techniques is again: *RELAI*, *EAR*, *FSCS*, *OPv1*. Since these treatments are characterized, in the average, by lower reliability values than Experiment 1 and few test cases, it turns out that the improvement of *RELAI* over the others, of *EAR* over the others, and of *FSCS* over OPv1 is, in such cases, more significant.

B. Experiment 3. Reliability estimate accuracy

Experiment 3 is to evaluate the ability of *RELAI* to accurately assess the achieved reliability. We consider the case of no error in the profile (i.e., assuming that the three generated profiles are correct). Operational profiles are in Table VI.

Figures 7a-7l report, for each treatment, the estimates accuracy in terms of *MSE*.

In all the cases, *RELAI* provides a relevantly lower MSE than the other techniques. The second best technique is the *AT-GD* method, whereas *OP* and *Random* testing behave similarly. Each figure reports the best *MSE* achieved by *RELAI*, with 800 test cases: values range from *1.25E-08* to *8.05E-6*. The best values, for all the techniques, are achieved with those programs where the (starting) reliability to estimate was higher (i.e,. SIENA, with a reliability of approximately 0.96 in the three profiles). Taking the average *MSE* in the best scenarios, namely the treatments with 800 test cases, we have: *1.91E-06, 8.19.E-5, 1.09E-04, 1.17E-4*, for, respectively, *RELAI*, *AT-GD*, *OPv2*, and *Random*. The overall average MSEs are: *8.20E-5, 2.42E-4, 2.92E-4, 3.007E-4*.

It is important to recall that *RELAI* is a technique that contemporary improves and assess the improved reliability: therefore, while for *AT-GD, OPv2*, and *Random* the code is frozen, and the reliability does not change during testing, *RELAI* is able to improve the delivered reliability while still providing an accurate estimate. Figure 8 reports the delivered reliability by *RELAI* in this experiment: considering the reliabilities before testing (of approximately 0.84, 0.88, 0.92 and 0.96 for Make, NanoXML, Grep and SIENA, respectively – the exact value depending on the profile), there is a significant gain in all the cases in achieved reliability after testing, besides the accurate MSE given as estimate. Of course, this does not apply for the other compared techniques, for which the final reliability is the same as the starting one.

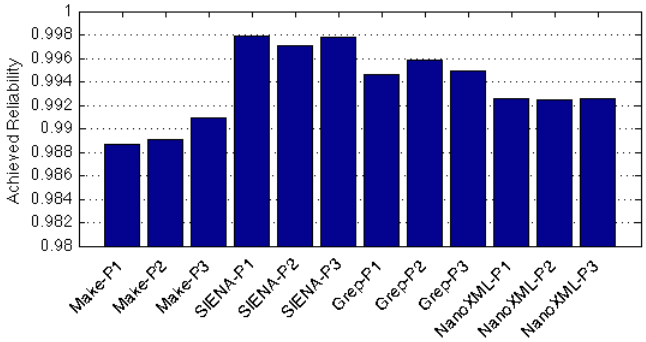
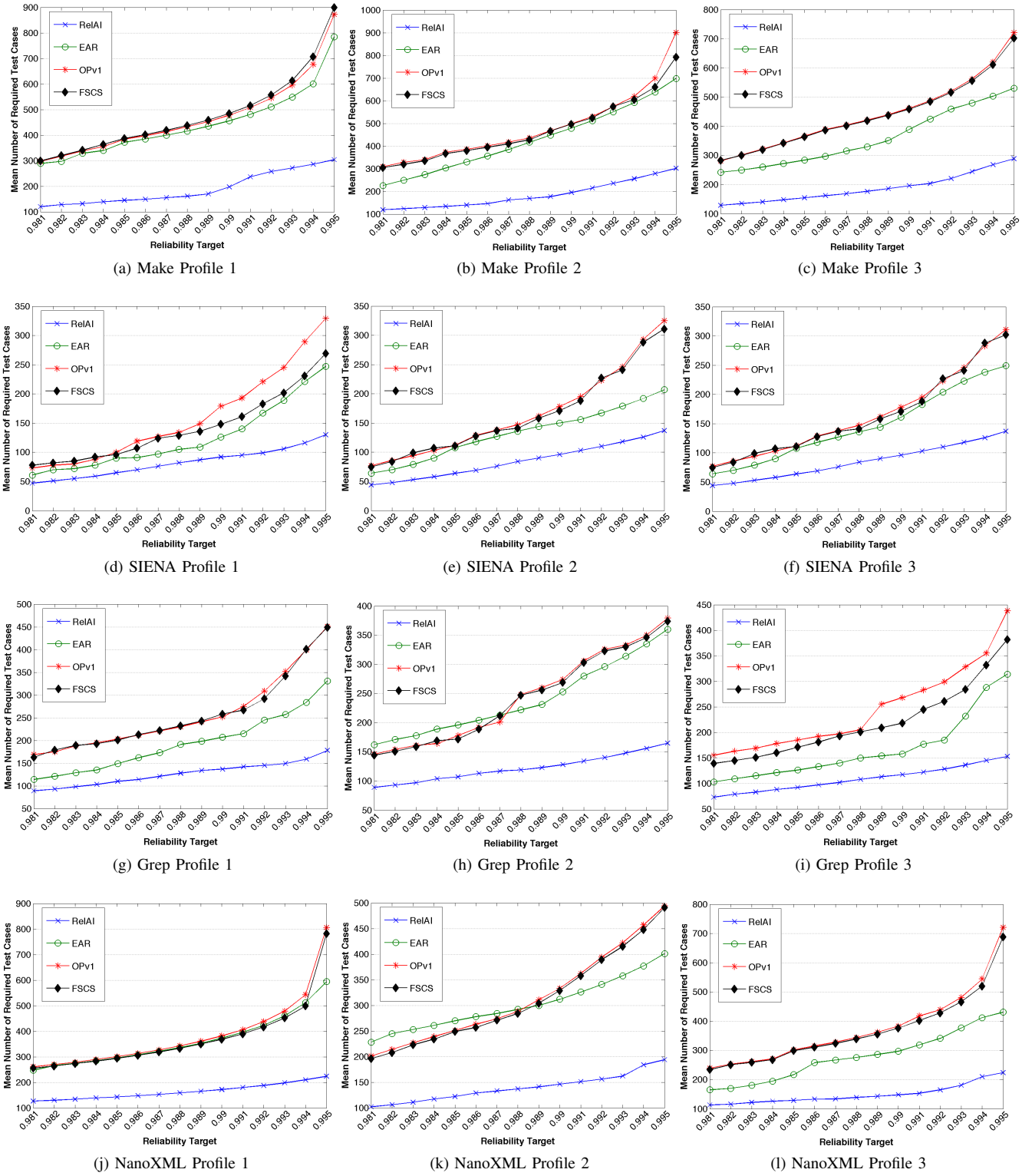


Fig. 8: Reliability achieved by RELAI in Experiment 3

Figure 9 reports synthetically the sample variances of all the treatments, with again the *RELAI* case with a variance lower than the others of an order of magnitude. The average of variances over all the scenarios are: *1.83E-5, 1.82E-4, 2.73E-4,*



(a) Make Profile 1 (b) Make Profile 2 (c) Make Profile 3

(d) SIENA Profile 1 (e) SIENA Profile 2 (f) SIENA Profile 3

(g) Grep Profile 1 (h) Grep Profile 2 (i) Grep Profile 3

(j) NanoXML Profile 1 (k) NanoXML Profile 2 (l) NanoXML Profile 3

Fig. 5: Sample mean of required number of test cases *vs* reliability targets

2.77E-4, for, respectively, *RELAI*, *AT-GD*, *OPv2*, and *Random*.

The Friedman test rejects the null hypothesis of equal MSEs with a $p\text{-}value = 7.66E\text{-}97$. Table VII reports the pairwise difference $p\text{-}values$. There is a highly significant difference in all the pairwise comparisons, except the *OPv2-Random* case, which turned out to be statistically equivalent.

C. Experiment 4: Impact of the profile error on estimation accuracy

Experiment 4 tests the *RELAI* ability to predict the profile error impact. Since we opted for a zero mean error on the

		Profile 1	*Profile 2*	*Profile 3*
Make	$[p_{min}; p_{max}]$	[2.73E-8 ; 2.17E-4]	[3.06E-8 ; 2.19E-4]	[4.83E-8 ; 2.17E-4]
	Test cases per partition	2323; 2250; 2367; 2298	1943; 1809; 1823; 1860; 1803	1572; 1567; 1509; 1536; 1524; 1530
	$\sum_t p_t$ per partition	6.2E-2; 1.82E-1; 3.2E-1; 4.35E-1	4.22E-2; 1.19E-1; 1.99E-1; 2.85E-1; 3.55E-1	2.87E-2; 8.59E-2; 1.37E-1; 1.95E-1; 2.48E-1; 3.05E-1
SIENA	$[p_{min}; p_{max}]$	[1.92E-9 ; 2.9E-4]	[9.08E-8 ; 2.93E-4]	[9.08E-8 ; 2.93E-4]
	Test cases per partition	1681; 1702; 1753; 1710	1364; 1381; 1381; 1319; 1401	1120; 1174; 1145; 1139; 1098; 1170
	$\sum_t p_t$ per partition	6.12E-2; 1.86E-1; 3.18E-1; 4.35E-1	3.99E-2; 1.21E-1; 2.01E-1; 2.7E-1; 3.69E-1	2.67E-2; 8.54E-2; 1.39E-1; 1.94E-1; 2.41E-1; 3.14E-1
Grep	$[p_{min}; p_{max}]$	[1.82E-8 ; 2.84E-4]	[2.96E-8 ; 2.84E-4]	[1.25E-8 ; 2.8E-4]
	Test cases per partition	1742; 1737; 1800; 1762	1413; 1375; 1381; 1502; 1370	1130; 1116; 1223; 1163; 1212; 1197
	$\sum_t p_t$ per partition	6.03E-2; 1.84E-1; 3.19E-1; 4.37E-1	4E-2; 1.16E-1; 1.96E-1; 2.98E-1; 3.49E-1	2.64E-2; 7.85E-2; 1.43E-1; 1.9E-1; 2.55E-1
Nano	$[p_{min}; p_{max}]$	[5.44E-8 ; 2.86E-4]	[2.77E-10 ; 2.84E-4]	[2.04E-7 ; 2.85E-4]
	Test cases per partition	1813; 1777; 1747; 1740	1422; 1430; 1457; 1364; 1404	1149; 1255; 1197; 1168; 1159; 1149
	$\sum_t p_t$ per partition	6.42E-2; 1.89E-1; 3.12E-1; 4.35E-1	3.97E-2; 1.22E-1; 2.07E-1; 2.72E-1; 3.59E-1	2.77E-2; 8.93E-2; 1.42E-1; 1.94E-1; 2.47E-1; 3E-1

TABLE VI: Operational profiles for Experiment 3

		Profile 1	*Profile 2*	*Profile 3*
Make	$[p_{min}; p_{max}]$	[5.03E-8 ; 2.18E-4]	[3.76E-8 ; 2.15E-4]	[3.78E-8 ; 2.18E-4]
	Test cases per partition	2377; 2242; 2354; 2265	1797; 1840; 1861; 1880; 1860	1521; 1595; 1514; 1584; 1508; 1516
	$\sum_t p_t$ per partition	6.53E-2; 1.83E-1; 3.2E-1; 4.32E-1	3.77E-2; 1.19E-1; 2E-1; 2.83E-1; 3.6E-1	2.73E-2; 8.7E-2; 1.38E-1; 2E-1; 2.46E-1; 3.02E-1
SIENA	$[p_{min}; p_{max}]$	[2.79E-8 ; 2.93E-4]	[1.59E-8 ; 2.95E-4]	[7.71E-8 ; 2.91E-4]
	Test cases per partition	1706; 1721; 1751; 1668	1422; 1396; 1367; 1304; 1357	1160; 1086; 1141; 1125; 1144; 1190
	$\sum_t p_t$ per partition	6.29E-2; 1.89E-1; 3.2E-1; 4.28E-1	4.26E-2; 1.24E-1; 2.03E-1; 2.69E-1; 3.61E-1	2.77E-2; 7.81E-2; 1.38E-1; 1.91E-1; 2.49E-1; 3.16E-1
Grep	$[p_{min}; p_{max}]$	[4.21E-8 ; 2.84E-4]	[1.58E-8 ; 2.85E-4]	[1.01E-8 ; 2.87E-4]
	Test cases per partition	1776; 1738; 1749; 1778	1442; 1391; 1398; 1382; 1428	1222; 1183; 1169; 1155; 1159; 1153
	$\sum_t p_t$ per partition	6.22E-2; 1.86E-1; 3.1E-1; 4.41E-1	4.16E-2; 1.19E-1; 1.99E-1; 2.75E-1; 3.66E-1	2.9E-2; 8.53E-2; 1.4E-1; 1.93E-1; 2.5E-1; 3.03E-1
Nano	$[p_{min}; p_{max}]$	[5.78E-8 ; 2.84E-4]	[8.55E-9 ; 2.8E-4]	[5.75E-8 ; 2.8E-4]
	Test cases per partition	1765; 1810; 1743; 1759	1425; 1418; 1323; 1414; 1497	1147; 1171; 1175; 1213; 1173; 1198
	$\sum_t p_t$ per partition	6.18E-2; 1.92E-1; 3.1E-1; 4.37E-1	4E-2; 1.2E-1; 1.85E-1; 2.78E-1; 3.77E-1	2.72E-2; 8.2E-2; 1.38E-1; 1.99E-1; 2.46E-1; 3.08E-1

TABLE VIII: Operational profiles for Experiment 4

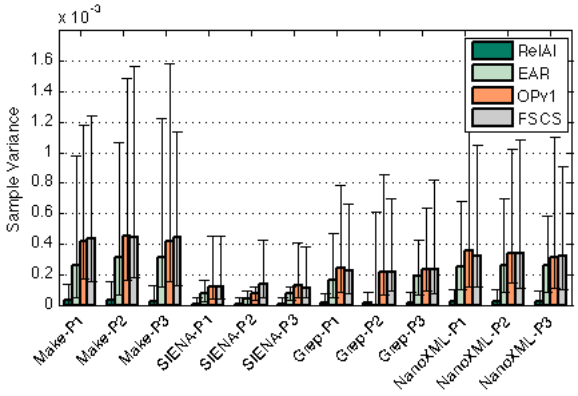


Fig. 9: Mean sample variance per scenario of reliability estimate (and min-max range). *Minimum variance is RELAI in SIENA-P1: 7.21 E-09*

Pairwise Comparison: *p-values*				
	RELAI	*AT-GD*	*OPv2*	*Random*
RELAI	-	**8.17E-15**	**2.91E-65**	**3.58E-73**
AT-GD	-	-	**7.39E-18**	**4.49E-22**
OPv2	-	-	-	0.7912

TABLE VII: Comparison for *MSE*. Text in boldface indicates that the factor is significant at least at 0.01. The ranking is: RELAI, AT-GD, OPv2, Random

profile, the accuracy in presence of error in the operational profile is only slightly worse than the case of correct profile. Thus, the impact of the profile error on the assessment accuracy deviation is small. Profiles features for this experiment are in Table VIII. Nonetheless, the procedure in Section V is able to detect such a deviation and closely predict its value. Figure 10 shows, for each program/profile scenario, the sample mean (over the 15 treatments with different number of test cases) of $\bar{\Delta}_{(\hat{R}_h,\hat{R})}$ values, which is the *prediction* of the share of reliability estimate offset caused by the profile error (namely, the *predicted offset*), and of the value $\Delta' - \Delta''$, which is the *actual offset* caused by the profile error (offset under P_C minus offset under $\hat{P}_{1,2,3}$). In all the scenarios, the two values are very close, with differences in the order of *1.00 E-4*.

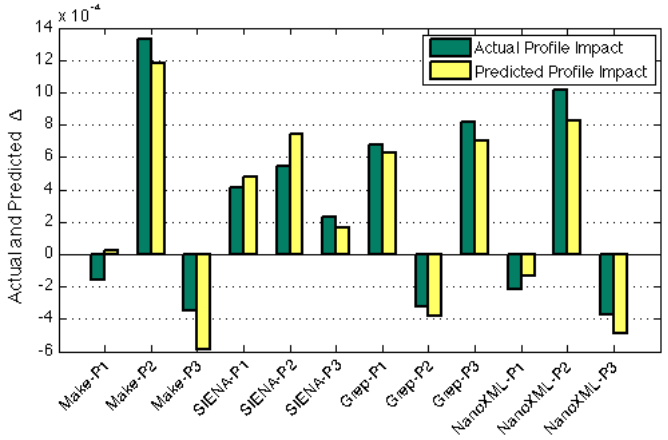
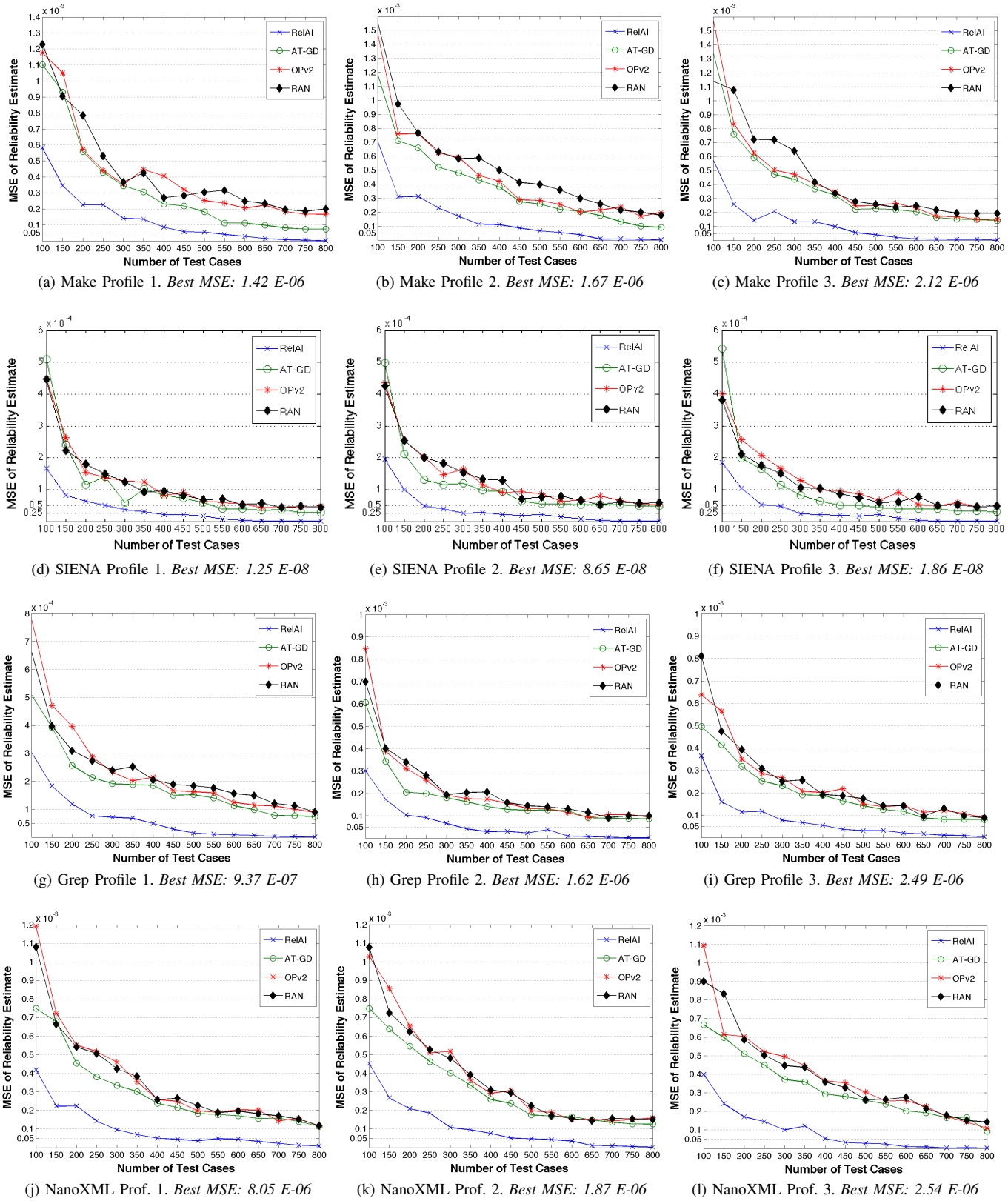


Fig. 10: Actual *vs* Predicted Profile Impact

Results are confirmed by the $hit\Delta\%$ metric: values for each program/profile pair are reported in Figure 11. In the average, the performance of the offset deviation's prediction procedure tell that in **98.22%** of the cases, the actual offset share due to the profile error is within the confidence interval of the predicted one, CI(*Mean*($\bar{\Delta}_{(\hat{R}_h,\hat{R})}$)).

This result can be used to adjust the reliability estimate, purging it from the impact of the profile estimation error:



(a) Make Profile 1. *Best MSE: 1.42 E-06*
(b) Make Profile 2. *Best MSE: 1.67 E-06*
(c) Make Profile 3. *Best MSE: 2.12 E-06*
(d) SIENA Profile 1. *Best MSE: 1.25 E-08*
(e) SIENA Profile 2. *Best MSE: 8.65 E-08*
(f) SIENA Profile 3. *Best MSE: 1.86 E-08*
(g) Grep Profile 1. *Best MSE: 9.37 E-07*
(h) Grep Profile 2. *Best MSE: 1.62 E-06*
(i) Grep Profile 3. *Best MSE: 2.49 E-06*
(j) NanoXML Prof. 1. *Best MSE: 8.05 E-06*
(k) NanoXML Prof. 2. *Best MSE: 1.87 E-06*
(l) NanoXML Prof. 3. *Best MSE: 2.54 E-06*

Fig. 7: MSE of reliability estimate

$\hat{R}' = \hat{R} - \bar{\Delta}_{(\hat{R}_h, \hat{R})}$, under the hypothesis that the true profile respects the initial condition imposed on the error, $\langle \varepsilon_0, C_{\varepsilon_0}\% \rangle$. Such an outcome reflects the ability of taking the error into account preventively, as the tester can implement more or less conservative stopping policies based on the predicted impact of the profile error on the reliability estimate. Moreover, the

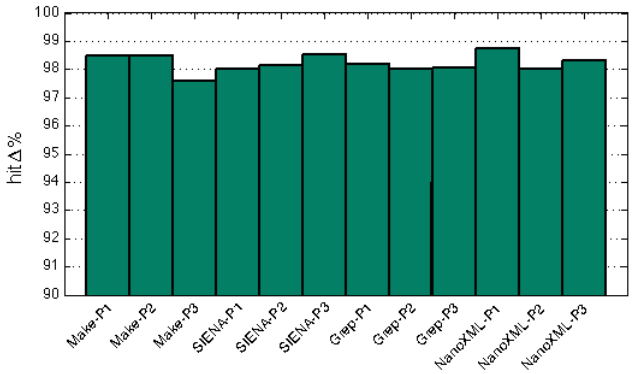


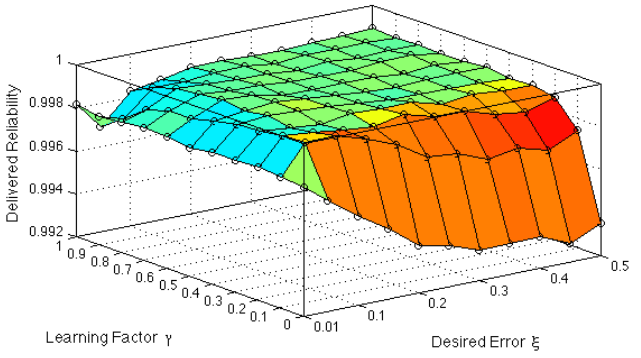
Fig. 11: Mean hitΔ% value per scenario

tester can adjust (hence get more accurate) reliability estimate, reducing considerably the uncertainty about the impact of this error. Increasing the number of repetitions for predicting the offset error can provide even more faithful $\hat{R}'$ values.

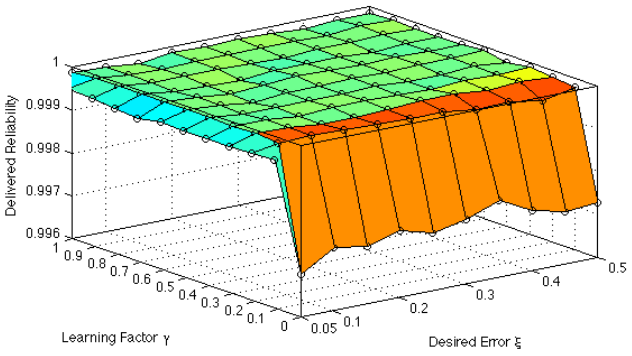
VIII. Sensitivity analysis

We hereafter report the sensitivity analysis carried out with respect to two relevant parameters of *RELAI*, both set at *domain*-level sampling, where the number of test cases to devote to each subdomain is decided. The former parameter is the desired error ξ between the unknown true distribution of the optimal number of test cases to allocate to each subdomain and the estimate based on samples. This parameter regulates how many test cases should be run at each iteration: the smaller the desired error, the higher the number of test cases required in a given iteration, the fewer the iterations for a fixed budget of available tests. This is important because few iterations with more test cases is likely to improve the assessment accuracy at the expense of a worse adaption ability with respect to the best distribution of test cases among partitions (hence, a potentially worse reliability improvement). The second analyzed factor is γ, namely the learning factor used in Equation 15, which tells how much the next allocation should depend on the previous one.

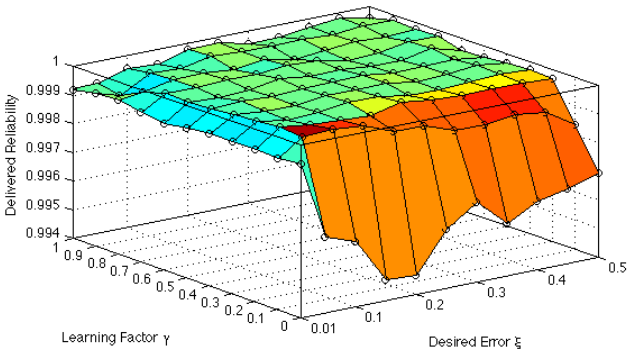
To conduct this analysis, we consider four scenarios, one per program, with the maximum number of test cases ($T = 800$) and a new random profile P under 5 partitions generated in the same way as for the previous experiments. Figure 12a-12d report the variation of the delivered reliability as ξ and γ vary, whereas Figure 13a-13d show the *MSE* variation. Regarding delivered reliability, it is clear, from all the cases, that a learning factor at the extremes ($\gamma \leq 0.1$ and $\gamma \geq 0.7$) penalizes the final results. On the other hand, results are quire invariably with respect to ξ, with slightly worse performance when ξ reaches 0.5. Regarding *MSE*, ξ has a significant impact. Whenever ξ is bigger than 0.1 or 0.2, the *MSE* increases, and exhibits a great sensitivity with respect to γ values. Under 0.1, the *MSE* is very small and approximately constant with respect to γ variation. Given these values, a good configuration is to keep $\xi \leq 0.1$ and $0.1 \leq \gamma \leq 0.7$ able to assure high delivered reliability and a small MSE. Other configurations can make sense if one is interested in only one objective, e.g., either improving or assessing reliability.



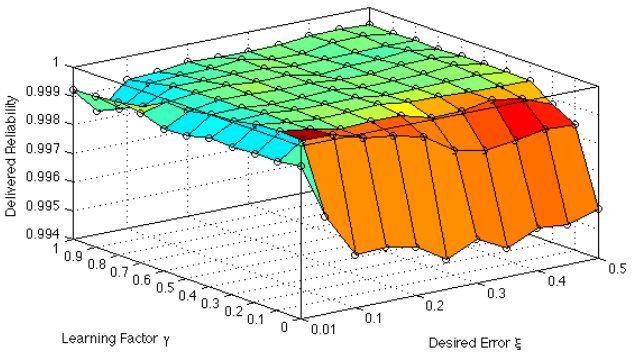
(a) Make



(b) SIENA

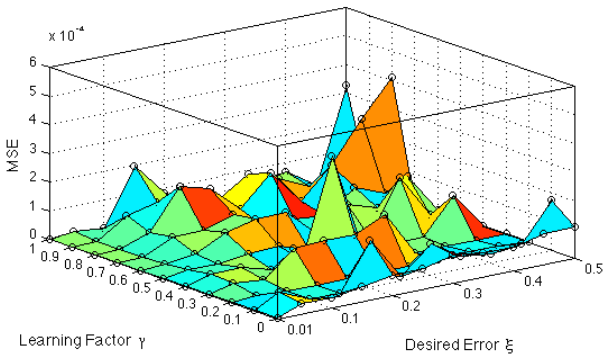


(c) Grep

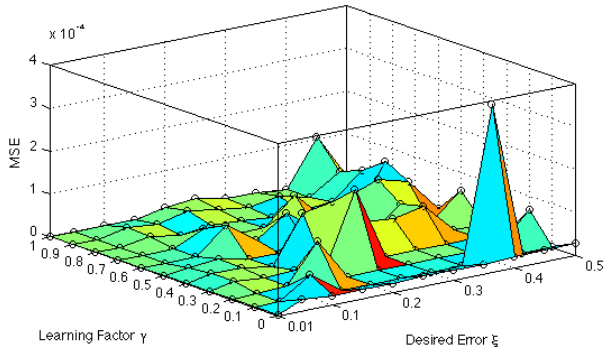


(d) NanoXML

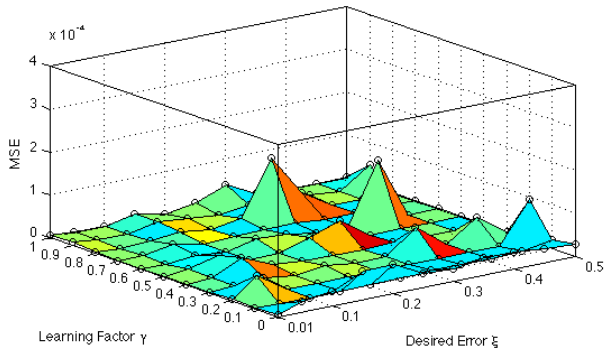
Fig. 12: Sensitivity analysis of delivered reliability



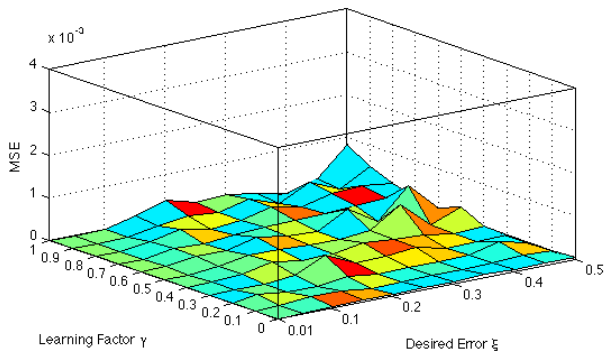
(a) Make



(b) SIENA



(c) Grep



(d) NanoXML

Fig. 13: Sensitivity analysis of MSE

IX. Threats to the validity and open issues

Practitioners adopting *RELAI* based on the presented conclusions should be aware of potential threats to results validity. Choices made for setting up and executing the experiment limit the generality of obtained results:

Test suite: we have generated test cases from the TSL specifications available with the programs. While we have applied the same test generation method (i.e., category-partition) for all the program under study, there is a subjective application of the method that may differ from program to program. The application of the method by the same authors to all the four programs limits this internal validity threat.

Seeded faults: program in the SIR repository were available with a set of seeded faults. However, their representativeness is undermined by the intention of seeding faults in the change among versions. We therefore adopted the G-SWFIT technique to inject faults by means of an automatic tool whose specific aim is to increases the representativeness [54] by spotting possible locations for each different type of fault, and considering commonly observed percentages of faults of different types. Despite this reduces the bias of artificial fault seeding (and a more representative faults than SIR's faults), real faults, of course, might still be present in a different way (type distribution and/or location).

A further internal validity threat includes the correctness of the implementation of all the experiment techniques, as well as of scripts for data collection and analysis, carried out by authors.

Results are also subject to external validity threats, as any empirical study, due to the following choices:

Subject programs: the experiment is performed on a set of programs selected from a publicly available repository; thus, care must be taken in extending conclusions to other programs. This is mitigated by selecting four different subjects, with diverse features in terms of: lines of code, complexity, implementation language, applicative target. Treatments are replicated on diverse programs in order to improve their generality.

Profiles: profiles are generated randomly as described in-Section VI. Although we re-generate three profiles in each experiment and use randomization, the adopted profiles can never represent all the possible profiles, and thus results with other profiles could, in principle, differ.

Replicating 100 times on four programs, in a full design configuration with 2340 treatments, mitigates the impact of these threats, but the described possible bias should be taken into account before extending conclusions.

Besides these threats, we left to future research the sensitivity analysis of *RELAI* with respect to the following choices made for carrying out the experiment: *i)* partitioning criterion: this is decided by the tester and, as explained, the choice does not affect the implementation of the strategy, but different results can be obtained depending on it. The effect of different partitioning criteria and how these can improve the approach need further investigation; *ii)* profile inaccuracy sensitivity: we experimented the approach under an error of $\langle 0.01, 99\% \rangle$ and with a zero mean; a theoretical and experimental evaluation

of *RELAI* under several error profiles will provide us with further insights about the properties of the reliability estimator. We expect that under larger errors the difference between the estimates biased by the profile error and the unbiased ones are bigger, and the benefit of *RELAI* is thus more impactful.

X. Conclusion and future work

RELAI is a new strategy oriented toward reliability improvement and assessment. Its underlying idea is to improve the delivered reliability and, at the same time, provide an estimate of the achieved level by: *i)* actively looking for failure regions most impacting the expected failure probability, not just waiting for them to come out in a "simulated" real usage, and *ii)* selecting test cases by a sampling scheme enabling the assessment during the fault removal, unlike current reliability assessment techniques. This is implemented through an adaptive scheme, which learns from the current state to drive future selection of test cases. A key feature of *RELAI* is the inclusion of the uncertainty on the real operational profile in the strategy definition. This allows tester to specify a constraint on this uncertainty, obtaining faithful estimates of reliability and predicting the share of inaccuracy caused by the specified profile error. It enables to control the error that a tester inevitably commits in deriving the profile, as *RELAI* assesses its impact before deploying the system. Results confirm the good performance of *RELAI* in terms of improvement, assessment, and mitigation of the profile error problem.

We believe *RELAI*'s characteristics can pave the ground to new attractive scenarios in the field of reliability testing, where debug and operational testing research areas can fruitfully benefit from each other's peculiarities. In particular, besides the mentioned sensitivity analyses, future studies could address these challenges: *i)* developing further on the relation between the uncertainty associated with the operational profile and the resulting uncertainty in the reliability estimation (e.g., developing mathematical model of uncertainty propagation with no or little burden on the tester, who should be unaware of details to adjust the estimation, and conceiving simpler approaches for the user, with respect to the current ones, to describe the profile with an associated confidence); *ii)* investigating new sampling strategies at domain level (namely, for reliability improvement) while keeping the feature of assessing at sub-domain level (e.g., investigating other Montecarlo-based inference methods to approximate the distribution of the "best" subdomains via stochastic sampling, or trying new probability update formulas in the adopted IS method); *iii)* similarly, investigating other survey sampling techniques (namely, for reliability assessment), e.g., adopting stratified sampling in combination with the RHC method; *iv)* using confidence intervals derived form the RHC-based method to devise new techniques based on maximizing the confidence in the estimate (e.g., similar to [4]); *v)* comparing the assessment ability with software reliability growth models, which take a different approach, or combine them for a better assessment; *vi)* removing the assumptions of perfect debugging, envisioning strategies that contemplates the possibility to introduce new bugs during fault removal (e.g., considering an imperfect debugging factor like in the literature on software reliability growth models [64], [65], [43]), *vii)* and removing the assumption of perfect oracle knowledge (e.g., borrowing methods of survey sampling under "random responses" to integrate with the RHC scheme, or by bootstrap-based methods). These challenges are the starting point of our next research to the improvements and best tuning of *RELAI* testing.

XI. Acknowledgement

This work has been partially supported by the European Commission 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] K.-Y. Cai, Y.-C. Li, and K. Liu, "Optimal and adaptive testing for software reliability assessment," *Information and Software Technology*, vol. 46, pp. 989 – 1000, Dec 2004.
[2] 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*, vol. 81, no. 8, pp. 1406 – 1429, 2008.
[3] 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*, vol. 40, pp. 396–412, April 2014.
[4] J. Lv, B.-B. Yin, and K.-Y. Cai, "Estimating confidence interval of software reliability with adaptive testing strategy," *J. Syst. Softw.*, vol. 97, pp. 192–206, Oct. 2014.
[5] M. R. Lyu, ed., *Handbook of Software Reliability Engineering*. Hightstown, NJ, USA: McGraw-Hill, Inc., 1996.
[6] B. Beizer, "Cleanroom process model: a critical examination," *IEEE Software*, vol. 14, pp. 14–16, Mar 1997.
[7] A. Pasquini, A. Crespo, and P. Matrella, "Sensitivity of reliability-growth models to operational profile errors vs. testing accuracy [software testing]," *IEEE Trans. on Reliability*, vol. 45, pp. 531–540, Dec 1996.
[8] J. Musa, "Sensitivity of field failure intensity to operational profile errors," in *5th Int. Symposium on Software Reliability Engineering (ISSRE)*, pp. 334–337, Nov 1994.
[9] M.-H. Chen, A. Mathur, and V. Rego, "A case study to investigate sensitivity of reliability estimates to errors in operational profile," in *5th IEEE Int. Symposium on Software Reliability Engineering (ISSRE)*, pp. 276–281, Nov 1994.
[10] C.-Y. Huang and M. Lyu, "Optimal testing resource allocation, and sensitivity analysis in software development," *IEEE Trans. on Reliability*, vol. 54, pp. 592–603, Dec 2005.
[11] J. Musa, "Software reliability-engineered testing," *Computer*, vol. 29, pp. 61–68, Nov 1996.
[12] J. Whittaker and G. Thomason, Michael, "A markov chain model for statistical software testing," *IEEE Trans. on Software Eng.*, vol. 20, pp. 812–824, Oct 1994.
[13] C. Kallepalli and J. Tian, "Measuring and modeling usage and reliability for statistical web testing," *IEEE Trans. on Software Engineering*, vol. 27, pp. 1023–1036, Nov 2001.
[14] S. Poulding and J. A. Clark, "Efficient software verification: Statistical testing using automated search," *IEEE Trans. on Software Eng.*, vol. 36, pp. 763–777, Nov. 2010.
[15] P. Currit, M. Dyer, and H. Mills, "Certifying the reliability of software," *IEEE Trans. on Software Engineering*, vol. SE-12, pp. 3–11, Jan 1986.
[16] R. Cobb and H. Mills, "Engineering software under statistical quality control," *IEEE Software*, vol. 7, pp. 45–54, Nov 1990.
[17] J. Musa, "Operational profiles in software-reliability engineering," *IEEE Software*, vol. 10, pp. 14–32, March 1993.
[18] P. Thevenod-Fosse and H. Waeselynck, "An investigation of statistical software testing," *Software Testing, Verification and Reliability*, vol. 1, no. 2, pp. 5–26, 1991.
[19] H. Mills, M. Dyer, and R. Linger, "Cleanroom software engineering," *IEEE Software*, vol. 4, no. 55, pp. 19–24, 1987.
[20] R. Selby, V. Basili, and F. Baker, "Cleanroom software development: An empirical evaluation," *IEEE Trans. on Software Engineering*, vol. SE-13, pp. 1027–1037, Sept 1987.

[21] R. Linger and H. Mills, "A case study in cleanroom software engineering: the ibm cobol structuring facility," in *12th Int. Computer Software and Applications Conference, COMPSAC 88*, pp. 10–17, Oct 1988.
[22] J. Poore, "A case study using cleanroom with box structures adl," tech. rep., Software Engineering Technology Technical Report CDRL 1880, 1990.
[23] P. Frankl, C. Dept, D. Hamlet, B. Littlewood, and L. Strigini, "Evaluating testing methods by delivered reliability," *IEEE Trans. on Software Engineering*, vol. 24, pp. 586–601, Aug 1998.
[24] L. Strigini and B. Littlewood, "Guidelines for statistical testing," Tech. Rep. PASCON/WO6-CCN2/TN12, ESA/ESTEC project PASCON, 1997.
[25] L. Madani, C. Oriat, I. Parissis, J. Bouchet, and L. Nigay, "Synchronous testing of multimodal systems: an operational profile-based approach," in *16th IEEE Int. Symposium on Software Reliability Engineering (ISSRE)*, pp. 10 pp.–334, Nov 2005.
[26] K. C. J., Cangusso, R. Decaylo, and A. Mathur, *An overiew of software cybernetics*. IEEE Digital Lib, 2004.
[27] C. Bai, C. Jiang, and K. Cai, "A reliability improvement predictive approach to software testing with bayesian method," in *Chinese Control Conference*, pp. 6031–6036, 2010.
[28] P. Cao, Z. Dong, K. Liu, and K.-Y. Cai, "Robust dynamic selection of tested modules in software testing for maximizing delivered reliability," *ArXiv e-prints*, vol. 1309.3052, 2013.
[29] T. Y. Chen, F.-C. Kuo, and H. Liu, "Application of a failure driven test profile in random testing," *IEEE Trans. on Reliability*, vol. 58, pp. 179–192, March 2009.
[30] T. Y. Chen, F. Kuo, and H. Liu, "Distributing test cases more evenly in adaptive random testing," *Journal of Systems and Software*, vol. 81, no. 12, pp. 2146 – 2162, 2008.
[31] T. Chen, H. Leung, and I. Mak, "Adaptive random testing," in *Advances in Computer Science - ASIAN 2004. Higher-Level Decision Making* (M. Maher, ed.), vol. 3321 of *Lecture Notes in Computer Science*, pp. 320–329, Springer Berlin Heidelberg, 2005.
[32] X. Zhang, T. Chen, and H. Liu, "An application of adaptive random sequence in test case prioritization," in *Proceedings of the Twenty-Sixth International Conference on Software Engineering and Knowledge Engineering (SEKE 2014) Marek Reformat (ed.)*, pp. 126–131., 2014.
[33] A. Tappenden and J. Miller, "A novel evolutionary approach for adaptive random testing," *Reliability, IEEE Transactions on*, vol. 58, pp. 619–633, Dec 2009.
[34] S. Anand and et al., "An orchestrated survey on automated software test case generation," *Journal of Systems and Software, Antonia Bertolino, Jenny Li, Hong Zhu (Editor/Orchestrators)*, 2013.
[35] R. V. Binder, *Testing Object-oriented Systems: Models, Patterns, and Tools*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.
[36] 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)*, vol. 24, no. 2, pp. 482–491, 1962.
[37] C. Sarbu, A. Johansson, N. Suri, and N. Nagappan, "Profiling the operational behavior of OS device drivers," in *19th Int. Symposium on Software Reliability Engineering (ISSRE)*, pp. 127–136, Nov 2008.
[38] C. Trammell, "Quantifying the reliability of software: statistical testing based on a usage model," in *2nd IEEE Int. Software Engineering Standards Symposium, Experience and Practice, ISESS'95*, pp. 208–218, Aug 1995.
[39] M. Riebisch, I. Philippow, and M. Gotze, "UML-based statistical test case generation," in *Int. Conference NetObjectDays on Objects, Components, Architectures, Services, and Applications for a Networked World*, pp. 394–411, Springer-Verlag, 2003.
[40] R. Pietrantuono, S. Russo, and K. Trivedi, "Software reliability and testing time allocation: An architecture-based approach," *Software Engineering, IEEE Transactions on*, vol. 36, pp. 323–337, May 2010.
[41] O. Silva, A. Crespo, M. Chaim, and M. Jino, "Sensitivity of two coverage-based software reliability models to variations in the operational profile," in *Fourth Int. Conference on Secure Software Integration and Reliability Improvement (SSIRI)*, pp. 113–120, June 2010.
[42] D. Cotroneo, R. Pietrantuono, and S. Russo, "Combining operational and debug testing for improving reliability," *IEEE Trans. on Reliability*, vol. 62, pp. 408–423, June 2013.
[43] B. Zachariah and R. N. Rattihalli, "Failure size proportional models and an analysis of failure detection abilities of software testing strategies," *IEEE Trans. on Reliability*, vol. 56, pp. 246–253, June 2007.
[44] K.Y.Cai, *Software defect and operational profile modeling*. Kluwer Academic Publishers, 1998.
[45] K.-Y. Cai, "Towards a conceptual framework of software run reliability modeling," *Information Sciences*, vol. 126, no. 1–4, pp. 137 – 163, 2000.
[46] K. Trivedi, *Probability and statistics with reliability, queuing and computer science applications (2nd ed.)*. John Wiley and Sons Ltd., Chichester, UK, 2001.
[47] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.
[48] M. Sridharan and A. Namin, "Prioritizing mutation operators based on importance sampling," in *21st Int. Symposium on Software Reliability Engineering (ISSRE)*, pp. 378–387, Nov 2010.
[49] D. Cotroneo, R. Pietrantuono, and S. Russo, "A learning-based method for combining testing techniques," in *35th Int. Conference on Software Engineering (ICSE)*, (Piscataway, NJ, USA), pp. 142–151, IEEE Press, 2013.
[50] D. G. Horvitz and D. J. Thompson, "A generalization of sampling without replacement from a finite universe," *Journal of the American Statistical Association*, vol. 47, no. 260, pp. pp. 663–685, 1952.
[51] D. Fox, "Adapting the sample size in particle filters through KLD-sampling," *Int. Journal of Robotics Research*, vol. 22, p. 2003, 2003.
[52] "Sir: Software-artifact infrastructure repository: http://sir.unl.edu/portal/index.html.."
[53] T. J. Ostrand and M. J. Balcer, "The category-partition method for specifying and generating fuctional tests," *Commun. ACM*, vol. 31, pp. 676–686, June 1988.
[54] R. Natella, D. Cotroneo, J. Duraes, and H. Madeira, "On fault representativeness of software fault injection," *Software Engineering, IEEE Transactions on*, vol. 39, no. 1, pp. 80–96, 2013.
[55] J. Duraes and H. Madeira, "Emulation of software faults: a field data study and a practical approach," *IEEE Transactions on Software Engineering*, vol. 32, no. 11, pp. 849–867, 2006.
[56] D. Cotroneo, R. Pietrantuono, and S. Russo, "Testing techniques selection based on odc fault types and software metrics," *Journal of Systems and Software*, vol. 86, pp. 1613–1637, June 2013.
[57] R. Chillarege, I. Bhandari, J. Chaar, M. Halliday, D. Moebus, B. Ray, and M.-Y. Wong, "Orthogonal defect classification-a concept for in-process measurements," *IEEE Trans. on Software Engineering*, vol. 18, pp. 943–956, Nov 1992.
[58] "Software fault emulation tool: http://www.mobilab.unina.it/sfi.htm."
[59] J. Mayer and C. Schneckenburger, "An empirical analysis and comparison of random testing techniques," in *Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering*, ISESE '06, (New York, NY, USA), pp. 105–114, ACM, 2006.
[60] K.-Y. Cai, "Optimal software testing and adaptive software testing in the context of software cybernetics," *Information and Software Technology*, vol. 44, no. 14, pp. 841 – 855, 2002.
[61] R. Hamlet, *Random Testing. Encyclopedia of Software Engineering.* John Wiley and Sons, 2002.
[62] P. B. Nemenyi, *Distribution-free multiple comparisons*. PhD thesis, Princeton University, 1963.
[63] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine Learning Research*, vol. 7, pp. 1–30, Dec. 2006.
[64] M. Jain, T. Manjula, and T. R. Gulati, "Software reliability growth model (srgm) with imperfect debugging, fault reduction factor and multiple change-point," in *International Conference on Soft Computing for Problem Solving*, pp. 1027–1037, 2011.
[65] M. Cinque, C. Gaiani, D. D. Stradis, A. Pecchia, R. Pietrantuono, and S. Russo, "On the impact of debugging on software reliability growth analysis: A case study," in *Computational Science and Its Applications – ICCSA 2014*, vol. Volume 8583 of *Lecture Notes in Computer Science*, pp. 461–475, Springer International Publishing, 2014.

Domenico Cotroneo received the PhD degree from the University of Naples, Italy, in 2001. He is currently an associate professor at the University of Naples. His main interests include software fault injection, dependability assessment techniques, and field-based measurements techniques. He is serving/has served as a pro- gram committee member for several dependability conferences, including DSN, EDCC, ISSRE, SRDS, and LADC.

Roberto Pietrantuono , Ph.D., IEEE Member, is currently a post-doc at Federico II University of Naples, Italy. He received his PhD degree (2009) in Computer and Automation Engineering from the same university. He collaborates with several companies of the Finmeccanica group, in the field of critical software system development. His research interests are in the area of software reliability engineering, particularly in the software verification of critical systems, software testing, and software reliability analysis.

Stefano Russo is Professor of Computer Engineering at the Federico II University of Naples, teaching Software Engineering and Distributed Systems, and leading the distributed and mobile systems research group. He co-authored over 130 papers in the areas of distributed software engineering, middleware technologies, software dependability, mobile computing. He is Associate Editor of the IEEE Transactions on Services Computing.