This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2487274, IEEE Transactions on Software Engineering

A Probabilistic Analysis of the Efficiency of Automated Software Testing

Marcel Böhme and Soumya Paul

Abstract—We study the relative efficiencies of the random and systematic approaches to automated software testing. Using a simple but realistic set of assumptions, we propose a general model for software testing and define sampling strategies for random ($\mathcal{R}$) and systematic ($\mathcal{S}_0$) testing, where each sampling is associated with a sampling cost: 1 and c units of time, respectively. The two most important goals of software testing are: (i) achieving in minimal time a given degree of confidence x in a program's correctness and (ii) discovering a maximal number of errors within a given time bound $\hat{n}$. For both (i) and (ii), we show that there exists a bound on c beyond which $\mathcal{R}$ performs better than $\mathcal{S}_0$ on the average. Moreover for (i), this bound depends asymptotically only on x. We show that the efficiency of $\mathcal{R}$ can be fitted to the exponential curve. Using these results we design a hybrid strategy $\mathcal{H}$ that starts with $\mathcal{R}$ and switches to $\mathcal{S}_0$ *when* $\mathcal{S}_0$ is expected to discover more errors per unit time. In our experiments we find that $\mathcal{H}$ performs similarly or better than the most efficient of both and that $\mathcal{S}_0$ may need to be significantly faster than our bounds suggest to retain efficiency over $\mathcal{R}$.

Index Terms—Partition Testing, Random Testing, Error-based Partitioning, Efficient Testing, Testing Theory

1 Introduction

Efficiency is an important property of software testing; potentially even more important than effectiveness. Because complex software errors exist even in critical, widely distributed programs for many years [2], [3], developers are looking for automated techniques to gain confidence in their programs' correctness. The most effective way to inspire confidence in the program's correctness *for all inputs* is called program verification. However, due to state explosion and other problems, the applicability of verification remains limited to programs of a few hundred lines of code. Now, software testing trades this effectivness for efficiency. It allows one to gain confidence in the program's correctness with every test input that is executed. So, automated testing is an efficient way to inspire confidence in the program's correctness *for an increasing set of inputs*. Yet, most research of software testing has mainly focussed on *effectiveness*:

> *The* most effective *testing technique reveals a maximal number of errors and inspires a maximum degree of confidence in the correctness of a program.*

Only now are we starting to investigate its *efficiency*:

> *The* most efficient *testing technique i) generates a sufficiently effective test suite in minimal time or ii) generates the most effective test suite in the given time budget.*

Using a simple set of assumptions, we construct a general model of software testing, define testing strategies where each generated test input is subject to a cost, and cast our efficiency analysis as a problem in probability theory.

- *M. Böhme is with the Software Engineering Chair at Saarland University in Germany but conducted this research while he was PhD Candidate at the School of Computing, National University of Singapore in Singapore. E-mail: boehme@cs.uni-saarland.de*
- *S. Paul is with the School of Computing, National University of Singapore in Singapore.*

A conference version of this article was published at the 2014 ACM SIGSOFT International Symposium on the Foundations of Software Engineering [1].

We model the testing problem as an *exploration of error-based input partitions*. Suppose, for a program there exists a partitioning of its input space into homogeneous subdomains [4], [5]. For each subdomain, either all inputs reveal an error or none of the inputs reveal an error. The number and "size" of such error-based partitions can be arbitrary but must be bounded. Assuming that it is unknown *a-priori* whether or not a partition reveals an error, the problem of software testing is to sample each partition in a systematic fashion to gain confidence in the correctness of the program.

A testing technique samples the program's input space. We say that a partition D_i is *discovered* when D_i is sampled for the first time. The sampled test input shows whether or not partition D_i reveals an error. Effectively, the sampled test input becomes a witness for the error-revealing property of D_i. A testing technique achieves the degree of confidence x when at least $x\%$ of the program inputs reside in discovered partitions. Hence, if none of the discovered partitions reveals an error, we can be certain that the program works correctly at least for $x\%$ of its input.

For our efficiency analysis, we consider two strategies: random testing that is oblivious of error-based partitions and systematic testing that samples each partition exactly once. *Random testing* $\mathcal{R}$ samples the input space uniformly at random and might sample some partitions several times and some not at all. Specifically, we show that for $\mathcal{R}$ the number and size of partitions discovered decays exponentially over time.[1] *Systematic testing* samples each error-based partition exactly once and thus *strictly increases* the established degree of confidence. We model a systematic testing technique $\mathcal{S}_0$ that chooses the *order* in which partitions are discovered uniformly at random and show that number and size of partitions discovered grows linearly over time. Note that our hypothetical S_0 can proof correctness eventually.

1. Thus, to *predict* the efficiency of $\mathcal{R}$, e.g., in terms of errors exposed (or even paths exercised), one only needs to fit an exponential curve!

Weyuker and Jeng [4] observe that a technique that samples from error-based partitions, like $\mathcal{S}_0$, is *most effective*. However, realistic systematic testing techniques are much less effective [6]. In fact, a test suite – that is 100% statement- and branch-coverage adequate, kills all possible mutants, and executes successfully – does still not guarantee the correctness of the tested program [7]. By analyzing the program's specification, tools can automatically generate test inputs that cover corner-cases [8]. By analyzing the program's source code, tools can generate inputs that stress potentially faulty statements, branches, or paths by increasing the coverage of the code [9], [10], [11]. By generating and analyzing deliberately faulty versions [12], tools can generate even more effective test input. Generally, the *more comprehensive such analysis*, the *more effective* can the testing technique be. But, with increasing analysis time, what about the associated *reduction of efficiency*?

To analyze the efficiency of both techniques, we assume that each sampling takes time and call it the *sampling cost*. Random testing does not spend any time on program analysis. We say that $\mathcal{R}$ *takes one unit of time* to sample one test input. However, systematic testing inherently requires some time to analyze artifacts related to the program, such as source code, specifications, or faulty versions, to derive the error-based partitions. We say that $\mathcal{S}_0$ *takes* c *units of time* to sample one test input. Note that we give the sampling cost for $\mathcal{S}_0$ as a factor of the sampling cost of $\mathcal{R}$. This allows us to account for the time spent on the concrete sampling-related tasks that are common to both techniques. For instance, if $\mathcal{R}$ takes, on average, $5ms$ to generate and execute a valid, readable, and typical test case and check whether it passes or fails, then by definition $\mathcal{S}_0$ takes $(\mathsf{c} \cdot 5)ms$ which includes the same time spent on test generation, execution, and oracle checking *and the time spent on program analysis*.

We observe that the efficiency of systematic testing decreases as the time spent on analysis increases while the efficiency of random testing remains unchanged. In other words, as the sampling cost c for $\mathcal{S}_0$ increases, it takes more time to establish the same degree of confidence and discover the same number of errors. So, in order for $\mathcal{S}_0$ to maintain its efficiency over $\mathcal{R}$, c cannot exceed a certain value and is thus bounded above!

In this paper, we study the *maximum sampling cost* c_0 of $\mathcal{S}_0$ beyond which the systematic testing technique S_0 is expected to be less efficient than random testing $\mathcal{R}$. Thereby, we explore two notions of testing efficiency that may well be the main goals of automated software testing: i) to achieve a given degree of confidence in minimal time, and ii) to expose a maximal number of errors in a given time. Furthermore, for our probabilistic analysis we take the sampling cost c as a constant. However, we provide a discussion on implications for the more realistic case when c increases with time, program size, number of inputs sampled, or is inversely proportional to partition size.

We design a *more efficient* hybrid technique. Given any systematic testing technique $\mathcal{S}$ that discovers one partition for each input sampled, we introduce a hybrid technique $\mathcal{H}$ that starts with $\mathcal{R}$ and switches to $\mathcal{S}$ after a certain time. We discuss how to determine *when* $\mathcal{H}$ switches from $\mathcal{R}$ to $\mathcal{S}$ in expectation and in practice and show that $\mathcal{H}$ is more efficient than both its constituent techniques, on the average.

The most important **contributions** of the paper are as follows. We provide a uniform mathematical framework for modeling software testing which is elementary and intuitive. In this framework we show that even a highly effective systematic testing technique is inefficient compared with random testing if the time per sampling is relatively too high. More precisely, we show the following:

- **1st Problem of Efficient Testing**. Given a degree of confidence x, we show that the time taken by $\mathcal{S}_0$ to sample an input cannot exceed $(ex - ex^2)^{-1}$ times the time taken by $\mathcal{R}$ to sample an input. Otherwise, $\mathcal{R}$ *is expected to achieve* x *earlier*. For instance, let $\mathcal{R}$ take $10ms$ per test; to establish the confidence that *any* program works correctly for 90% of its input, $\mathcal{S}_0$ *must take less than* $41ms$ per test. In our experiments we find that $\mathcal{S}_0$ must take *signif. less time* than our bound suggests to be expected to achieve x earlier.
- **2nd Problem of Efficient Testing**. Given $\hat{n}$ time units, we show that the time taken by $\mathcal{S}_0$ per test cannot exceed $\frac{\hat{n}}{k} \cdot (1 - (1 - q_{\min})^{\hat{n}})^{-1}$ times the time taken by $\mathcal{R}$ per test, in order for $\mathcal{S}_0$ expose more errors in $\hat{n}$ time units — where k is the number of partitions and $q_{\min}$ the fractional size of the "smallest" error-revealing partition in the program's input space.
- **Exponential Decay**. We show that for $\mathcal{R}$ the number of errors discovered decays exponentially over time. In practice, this allows to *predict* the efficiency of $\mathcal{R}$ by fitting the exponential curve $h(n) = ae^{-\lambda n} + b$.
- **Hybrid Testing Technique**. Using the above insights, (the efficiency of $\mathcal{R}$ decays exponentially while that of $\mathcal{S}$ does not) we design a hybrid technique $\mathcal{H}$ which starts using $\mathcal{R}$ and switches to $\mathcal{S}$ *when* $\mathcal{S}$ is expected to discover more partitions per unit time than $\mathcal{R}$.
- **24,000 Simulation Experiments**. We observe that $\mathcal{H}$ performs similarly or better than the most efficient of both, and that the maximum cost c_0 of $\mathcal{S}_0$ can be significantly higher if the input space is partitioned such that there is a small number of huge and a very large number of very tiny partitions.

In summary, we present *strong, elementary, and theoretical results* about the efficiency of automated testing that hold for all programs and every systematic testing technique under the realistic assumptions stated in the following section.

2 Preliminaries

2.1 Background

In this work, we focus on automated testing techniques that seek to establish a certain degree of confidence in the correctness of the program or reveal a maximal number of errors. Interestingly, this eliminates inexhaustive, automated techniques that seek to generate just one failing test input as evidence of the incorrectness of the program. First, the search for a failing test input may never terminate due to the undecidability of the infeasible path problem [13]. Secondly, the absence of a failing test input throughout the search does not inspire any degree of confidence in the absence of errors. Instead, we shall focus on partition testing techniques, such as coverage, mutation, and specification based testing.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2487274, IEEE Transactions on Software Engineering

Partition testing [4], [7] comprises of testing techniques that 1) divide the program's input domain into classes whose points share the same property in some respect and then 2) test the program for at least one input from each class. Thus, the problem of systematic testing is reduced to finding a "good" partition strategy. For example, a *specficiation-based* partition strategy might divide the input domain into subdomains, each of which invokes one of several program features or satisfies the pre-condition of some predicate [8]. *Mutation-based* partition strategies may yield subdomains, each of which strongly kills a certain mutant of the program [12], [14]. A *differential* partition strategy yields subdomains, each of which either homogeneously exposes a semantic difference or homogeneously shows semantic equivalence [15]. Symbolic execution is a *path-based* partition strategy [11]. One may also consider an *assertion-based* partitioning strategy that divides the input space into classes where inputs do and others do not violate an assertion in the program. Such assertion-based partitioning would be fit to serve as practical counter-part of the hypothetical error-based partitioning where erroneous program behavior is explicitly encoded using assertions (or exceptions, etc.).

However, questioning its *effectiveness*, Hamlet and Taylor [7] find that "partition testing does not inspire confidence". Varying several parameters, the authors repeated the experiments of Duran and Ntafos [16] who presented a surprising result: The number of errors found by random and partition testing is very similar. Hamlet and Taylor came to much the same conclusion. The results universally favoured partition testing, but not by much. Weyuker and Jeng [4] found that the effectiveness of partition testing varies depending on the fault rate for each subdomain that is systematically sampled and concluded that a partitioning strategy that yields error-based (revealing) subdomains is the most effective. Subsequently, several authors discussed conditions under which partition testing is generally more effective than random testing (e.g., [17], [18]).

Arcuri et al. [19] study the *scalability* of random testing. In this work, scalability refers to the ability of exercising many "targets" in the program as the number of targets increases. Specifically, the authors show that random testing scales better than a *directed* testing technique that focuses on one target until it is "covered" before proceeding to the next. Intuitively, parallel search (here, random testing) scales better than sequential search (here, directed testing). In contrast, we assess the scalability of systematic testing relative to random testing by investigating the efficiency of both techniques as the program size increases. Thereby, we also consider systematic techniques that are not "directed".

Leaving the scope of our analysis are several practical concerns that are common to all automated testing techniques. i) Firstly, there is the *oracle problem* [20] which states that a mechanism deciding for every input whether the program computes the correct output is pragmatically unattainable and only approximate. Partial solutions include the automated encoding of common [21], [22], [23] and the manual encoding of custom error conditions as assertions [24], [25], [26]. ii) Secondly, there is the *typicality problem* which states that automatically generated test cases may not represent the "typical" input a user would provide or "valid" input that satisfies some pre-condition for the program to execute normally. Technically, both techniques could sample according to the operational distribution [27] or using symbolic grammars [28]. Then, both techniques receive the same ability to sample typical, valid inputs. We make no such assumptions. iii) Finally, we want to stress explicitly that for the purpose of this article the achieved *code coverage is only secondary*. For instance, suppose a branch somewhere in the program is exercised only if for some variable `i` we have `i == 780234`. Then this branch may (or may not) have a very low probability to be exercised randomly. Instead, the technique shall achieve confidence and expose errors. In our investigations, we also account for partitions that are relatively small, possibly containing only one input.

2.2 Definitions and Notations

We construct a general model of software testing that is based on three simple assumptions: i) the input space is bounded, ii) errors are deterministic, and iii) it is unknown a-priori whether or not some input reveals an error. These assumptions are stated explicitely and formally and may be relaxed in future work. Furthermore, we define error-based partitioning, the two problems of efficient software testing, and the two testing strategies, $\mathcal{R}$ and $\mathcal{S}_0$.

Given any program $\mathcal{P}$, the number of input variables to the program determine the *dimensionality* of the program's input space. The values for an input variable determines the values of the corresponding dimension in the program's input space. For instance, a program with two input variables of type integer has a two dimensional input space that can take any integer values. Regarding the input space, we make the following *assumptions*:

- **Bounded Dimensionality**. Given any program $\mathcal{P}$, the space of inputs to $\mathcal{P}$ has a bounded dimension. This assumption is realistic since the length of $\mathcal{P}$ is bounded, it can only manipulate a bounded number of variables.
- **Bounded Input Space**. Given any program $\mathcal{P}$, every input variable $\mathcal{P}$ can take only a bounded number of values from a finite domain. This assumption is also realistic since in practice the size of the registers where the variables are stored is bounded.

Given these assumptions, we see that given a program $\mathcal{P}$, its input space can be taken to be a finite, measurable metric space $\mathcal{D} = \prod_{i=1}^{d} A_i$ where d is the dimension of the input space of $\mathcal{P}$ and A_i is a finite set for every $1 \leq i \leq d$. In what follows, we fix a program $\mathcal{P}$ which in turn fixes the dimension d and the input space $\mathcal{D}$.

Definition 1 (*Error-based Partitioning*)

> *The input space $\mathcal{D}$ of a program $\mathcal{P}$ can be partitioned into k disjoint non-empty subdomains $\mathcal{D}_i$ where $1 \leq i \leq k$ with the following property: Either every input $t \in \mathcal{D}_i$ reveals the same error, or every input $t \in \mathcal{D}_i$ does not reveal an error. If every input of a partition $\mathcal{D}_i$ reveals an error then we call $\mathcal{D}_i$ an* error-revealing *partition.*

We notice that Def. 1 requires *determinism*: All executions of the same test input yield the same output. This is satisfied also if a model that *renders* an execution deterministic, like a specific thread schedule, is constituent of the test input.

Note that $|\mathcal{D}_i| > 0$ for all $1 \leq i \leq k$ where $|\cdot|$ denotes the cardinality of a set. Since $|\mathcal{D}|$ is finite, k is finite, too, and

$$|\mathcal{D}| = \sum_{i=1}^{k} |\mathcal{D}_i| \tag{1}$$

If we draw an input t uniformly at random from $\mathcal{D}$, for every partition $\mathcal{D}_i$ there is a probability that $t \in \mathcal{D}_i$. We denote this probability vector by $\mathbf{p} = \langle p_1, \cdots, p_k \rangle$. Note that for every $i : 1 \leq i \leq k$

$$p_i = \frac{|\mathcal{D}_i|}{|\mathcal{D}|} \quad \text{and} \tag{2}$$

$$\sum_{i=1}^{k} p_i = 1 \tag{3}$$

For every $i : 1 \leq i \leq k$, let θ_i be the indicator random variable which is 1 if partition $\mathcal{D}_i$ reveals an error and 0 otherwise.

A *testing technique* samples the input space of the program-under-test and discovers error-based partitions. We assume that the information whether a partition does or does not reveal an error is unknown a-priori. This is a fair assumption because otherwise there was no need for testing. Hence, each sampled test case becomes a *witness* of whether or not the corresponding partition is error-revealing.

Definition 2 (*Discovered Partitions*)

Given a testing technique $\mathcal{F}$ that samples the input space $\mathcal{D}$, we say that $\mathcal{F}$ discovers *partition D_i in n units of time if $\mathcal{F}$ samples from D_i after exactly n units of time and no test input has been sampled from D_i previously.*

While the goal of software verification is to show the correctness of the program for *all* inputs, the goal of software testing is to show the correctness of the program at least for *some* $x\%$ of the input. Arguably, this more modest goal may also be more practical and economical.

Definition 3 (*Achieving Confidence*)

Let $\mathbf{x} = \langle X_1, \cdots, X_k \rangle$ where X_i is the random variable indicating whether testing technique $\mathcal{F}$ has discovered partition D_i in n units of time, we say that $\mathcal{F}$ achieves the degree of confidence *x in n units of time if*

$$x|D| \leq \sum_{i=1}^{k} X_i |D_i|$$

In other words, a testing technique achieves the degree of confidence x when at least $x\%$ of the program inputs reside in discovered partitions.

In the following, we define two main goals of efficient software testing. The first goal is to achieve a certain degree of confidence x in minimal time.

Definition 4 (*The 1st Problem of Efficient Software Testing*)

Given two testing techniques, $\mathcal{F}_1$ and $\mathcal{F}_2$, and the degree of confidence x, let n_1 and n_2 be the units of time in which $\mathcal{F}_1$ and $\mathcal{F}_2$ are expected to achieve x. We say that $\mathcal{F}_1$ is expected to be more efficient *than $\mathcal{F}_2$ according to the* 1st Problem of Efficient Software Testing *iff $n_1 < n_2$.*

The second goal is to expose the most number of errors in a certain time budget $\hat{n}$ (see **E-measure** [19]).

Definition 5 (*The 2nd Problem of Efficient Software Testing*)

Given two testing techniques, $\mathcal{F}_1$ and $\mathcal{F}_2$, and the time budget $\hat{n}$, let d_1 and d_2 be the expected number of error-revealing partitions discovered by $\mathcal{F}_1$ and $\mathcal{F}_2$ in $\hat{n}$ units of time. We say that $\mathcal{F}_1$ is expected to be more efficient *than $\mathcal{F}_2$ according to the* 2nd Problem of Efficient Software Testing *iff $d_1 > d_2$.*

Now, we define two particular testing techniques, random testing $\mathcal{R}$ and the systematic testing technique $\mathcal{S}_0$. For each technique we assign a sampling cost that corresponds to the time that is required for sampling a test input. The *sampling* of a test input comprises of concrete tasks such as generating and executing the corresponding test case and checking the correctness of its outcome. The sampling cost is computed as the sum of the time it takes each sampling-related task.

Definition 6 (*Random Testing $\mathcal{R}$*)

Given a program $\mathcal{P}$, random testing R tests $\mathcal{P}$ by sampling at each iteration its input space $\mathcal{D}$ uniformly at random. The cost for each sampling is one *unit of time.*

Note that random testing $\mathcal{R}$ *samples with replacement*.

Definition 7 (*Systematic Testing Technique $\mathcal{S}_0$*)

Given a program $\mathcal{P}$, the systematic testing technique $\mathcal{S}_0$ tests $\mathcal{P}$ by sampling at each iteration exactly one undiscovered error-based partition uniformly at random. The sampled partition itself is also chosen uniformly at random from the remaining undiscovered error-based partitions. The cost for each sampling is c *units of time.*

Note that $\mathcal{S}_0$ samples exactly one input from each error-based partition. Eventually, $\mathcal{S}_0$ will have discovered all partitions and is thus *most effective*. The cost for each sampling of c unit of time includes the time to generate and execute the corresponding test case and verify the correctness of its output *and* the time it takes for the additional analysis. Hence, we call c the *analysis cost* of $\mathcal{S}_0$. Note that $\mathcal{S}_0$ discovers all of k partitions in ck units of time.

We note that both techniques can sample from a *reduced* input subdomain that contains only e.g., valid, readable, or typical test cases if such are concerns. However, we make no such assumptions.

We now delve into the technical details. In the following, we shall formalise relevant concepts of approximation and exponential decay.

Definition 8 (*Asymptotics*)

Let $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$ be real functions. We say

1) *$f \sim g$ if $\frac{f(n)}{g(n)} \to 1$ as $n \to \infty$. Thus, for every $\epsilon > 0$ there exists $n_0 \in \mathbb{R}^+$ such that for every $n > n_0$, $|f(n) - g(n)| < \epsilon$.*
2) *$f \lesssim g$ if there exist constants $c, n_0 \in \mathbb{R}^+$ such that $|f(n)| < c|g(n)|$ for all $n > n_0$.*
3) *$f \gtrsim g$ if there exist constants $c, n_0 \in \mathbb{R}^+$ such that $|f(n)| > c|g(n)|$ for all $n > n_0$.*

Note, if $f \lesssim g$ then $g \gtrsim f$ and conversely.

Definition 9 (*Exponential Decay*)

A function $f : \mathbb{R} \to \mathbb{R}$ has exponential decay *if it is differentiable at every $x \in \mathbb{R}$ and $\frac{df(x)}{dx} = -\lambda f(x)$ for some constant λ. In particular note that the function $ae^{-\lambda x}$ where a is a constant has exponential decay.*

3 The 1st Problem of Efficient Testing

Achieving a given degree of confidence x in minimal time is the 1st Problem of Efficient Software Testing (1st PoEST). In other words, an efficient testing technique establishes that the program works correctly at least for x% of its input in minimal time. Given a degree of confidence x, we compare the expected time it takes to achieve x by random testing $\mathcal{R}$ and by the systematic testing technique $\mathcal{S}_0$. After introducing the concepts and insights with an example, we investigate the efficiency of $\mathcal{S}_0$ and $\mathcal{R}$. For $\mathcal{S}_0$, we show that the expected degree of confidence achieved grows linearly with time. In contrast, for $\mathcal{R}$ we show exponential decay.

Given a degree of confidence x, we find that the sampling cost c of $\mathcal{S}_0$ must be below $(ex - ex^2)^{-1}$ units of time in order for $\mathcal{S}_0$ to remain more efficient than $\mathcal{R}$. For example, to establish that the program works correctly for 90% of its input, sampling one test systematically must take much less than *five* times the time it takes to sample one test randomly.

3.1 Efficiency Analysis of Individual Techniques

In this work, we define the confidence that is achieved wrt. the input space that is discovered (Def. 3). So, we give the expected size of input space discovered after n time units.

Lemma 1 (*Confidence – Efficiency of $\mathcal{S}_0$*)

For the systematic testing technique $\mathcal{S}_0$, the expected input space discovered after n time units is

$$f_s(n) = \frac{|\mathcal{D}|}{\mathsf{c}k} \cdot n$$

where c is units of time taken for sampling one test input.

Proof: By Definition 7, $\mathcal{S}_0$ discovers n/c partitions in n units of time. The order in which partitions are discovered is decided by choosing uniformly at random from the set of undiscovered partitions. Let X_i be the random variable indicating that partition $\mathcal{D}_i$ has been discovered after n units of time. Then,

$$\mathbb{E}[X_i] = \frac{n}{\mathsf{c}k} \tag{4}$$

Let the expected size of the input space discovered by $\mathcal{S}_0$ after n units of time be given by the function $f_s : \mathbb{N} \to \mathbb{R}$. We compute $f_s(n)$ as the expected value of the sum of the size of all discovered partitions.

$$f_s(n) = \mathbb{E}\left[\sum_{i=1}^{k} X_i|\mathcal{D}_i|\right] \tag{5}$$

$$= \sum_{i=1}^{k} |\mathcal{D}_i|\mathbb{E}[X_i] \qquad \text{[by lin. of exp.]} \tag{6}$$

$$= \sum_{i=1}^{k} |\mathcal{D}_i|\frac{n}{\mathsf{c}k} \qquad \text{[by Eqn. (4)]} \tag{7}$$

$$= \frac{|\mathcal{D}|}{\mathsf{c}k} \cdot n \qquad \text{[by Eqn. (1)]} \tag{8}$$

■

Thus, the expected size of the input space discovered grows linearly with the number of iterations. As the cost increases, the slope with the time-axis, $|\mathcal{D}|/(\mathsf{c}k)$, of $f_s(n)$ decreases.

Now, we look at the case for random testing.

Lemma 2 (*Confidence – Efficiency of $\mathcal{R}$*)

For random testing $\mathcal{R}$, the expected size of the input space discovered after n units of time is

$$f_r(n) = |\mathcal{D}|\left[1 - \sum_{i=1}^{k} p_i(1-p_i)^n\right]$$

$$\sim |\mathcal{D}|\left[1 - \sum_{i=1}^{k} p_i e^{-np_i}\right]$$

Proof: By Definition 6, $\mathcal{R}$ samples n tests in n units of time. By Eqn. (2), the probability that $\mathcal{R}$ discovers partition $\mathcal{D}_i$ in any trial is p_i. Let X_i be the random variable indicating that partition $\mathcal{D}_i$ has been discovered after n units of time. The probability that $\mathcal{D}_i$ has *not* been discovered after n units of time is $(1-p_i)^n$. Thus,

$$\mathbb{E}[X_i] = 1 - (1-p_i)^n \tag{9}$$

Let the expected size of the input space discovered by $\mathcal{R}$ after n units of time be given by the function $f_r : \mathbb{N} \to \mathbb{R}$. We compute $f_r(n)$ as the expected value of the sum of the size of all discovered partitions.

$$f_r(n) = \mathbb{E}\left[\sum_{i=1}^{n} X_i|\mathcal{D}_i|\right] \tag{10}$$

$$= \sum_{i=1}^{k} |\mathcal{D}_i|\mathbb{E}[X_i] \qquad \text{[by linearity of exp.]} \tag{11}$$

$$= \sum_{i=1}^{k} |\mathcal{D}_i|[1-(1-p_i)^n] \qquad \text{[by Eqn. (9)]} \tag{12}$$

$$= |\mathcal{D}|\sum_{i=1}^{k} p_i[1-(1-p_i)^n] \qquad \text{[by Eqn. (2)]} \tag{13}$$

$$= |\mathcal{D}|\left[1 - \sum_{i=1}^{k} p_i(1-p_i)^n\right] \qquad \text{[by Eqn. (3)]} \tag{14}$$

To approximate the above quantity, we cast the problem of achieving confidence into the problem of finding the *bonus sum* in the generalized coupon collectors problem [29]. Given $|\mathcal{D}|$ coupons with k different colours, there are $|\mathcal{D}_i|$ coupons of a colour i where $1 \le i \le k$ and each coupon has a bonus value of $|\mathcal{D}_i|$. Note that the probability to collect a coupon of colour i is $p_i = |\mathcal{D}_i|/|\mathcal{D}|$. Then the above quantity is nothing but the bonus sum of the coupons collected after a person collected n coupons when counting the bonus value of each colour only once. From the result of Rósen [29, Theorem 1] we have

$$f_r(n) \sim |\mathcal{D}|\left[1 - \sum_{i=1}^{k} p_i e^{-np_i}\right]$$

■

3.2 Example for Equal-Sized Partitions

We illustrate the main insights for the simplified case where the size of each partition is equal. In this setting, we demonstrate that the confidence achieved per unit of time decays exponentially for random testing $\mathcal{R}$ while it grows linearly for the systematic testing technique $\mathcal{S}_0$. Later, this result is generalized for partitions of arbitrary size.

First, we show a simple corollary of Lemma 2.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2487274, IEEE Transactions on Software Engineering

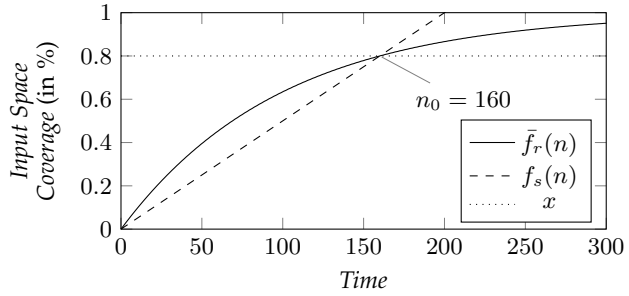

Fig. 1. On the average, $\mathcal{S}_0$ and $\mathcal{R}$ break even after approximately 80% of the input space was covered and 160 random test inputs were sampled (when $\mathsf{c} = 2$, $k = 100$, $p_i = \frac{1}{k}$).

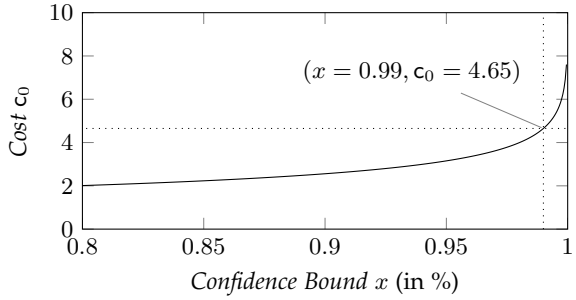

Fig. 2. If the average analysis cost of $\mathcal{S}_0$ exceeds c_0 for a given degree of confidence x, then $\mathcal{R}$ is expected to be more efficient than $\mathcal{S}_0$ (here for $p_i = \frac{1}{k}$).

Corollary 1

For random testing $\mathcal{R}$ where $p_i = \frac{1}{k}$ for all $i : 1 \leq i \leq k$, the expected size of input space discovered after n time units is

$$\begin{aligned}\bar{f}_r(n) &= |\mathcal{D}| \left[1 - \left(1 - \frac{1}{k}\right)^n\right] \\ &= |\mathcal{D}| - |\mathcal{D}| e^{-\lambda n}\end{aligned}$$

where $\lambda = \ln\left(\frac{k}{k-1}\right)$.

Proof: Setting $p_i = \frac{1}{k}$ for every $i : 1 \leq i \leq k$ in $f_r(n)$, we have

$$\bar{f}_r(n) = |\mathcal{D}| \left[1 - \sum_{i=1}^{k} \frac{1}{k} \left(1 - \frac{1}{k}\right)^n\right] \quad (15)$$

$$= |\mathcal{D}| \left[1 - \left(1 - \frac{1}{k}\right)^n\right] \quad (16)$$

$$= |\mathcal{D}| - |\mathcal{D}| \left(\frac{k}{k-1}\right)^{-n} \quad (17)$$

$$= |\mathcal{D}| - |\mathcal{D}| \left(e^{\ln\left(\frac{k}{k-1}\right)}\right)^{-n} \quad (18)$$

■

The corollary shows that $\bar{f}_r(n)$ has exponential decay as per Definition 9.

Figure 1 shows the expected size of input space discovered per unit of time for $\mathcal{R}$ and $\mathcal{S}_0$ when $k = 100$ and $\mathsf{c} = 2$. So, it takes $\mathcal{S}_0$ twice as long to sample a test input compared to $\mathcal{R}$. On the average, after 80 units of time, $\mathcal{S}_0$ discovered partitions in 40% of the input space while $\mathcal{R}$ discovered partitions in 55% of the program's input space. On the average, after 160 units of time both techniques break even, having discovered partitions in 80% of the input space.

There exists a time n_0 where $\bar{f}_r(n_0) = f_s(n_0)$ and $\mathcal{S}_0$ has discovered more of the input space than $\mathcal{R}$ for any $n > n_0$, on the average. To assess the *relative efficiency* of $\mathcal{S}_0$ we pose the following question: Given a degree of confidence x, what is the maximum cost c_0 for $\mathcal{S}_0$ such that $\mathcal{S}_0$ achieves x in time $n \leq n_0$? We give the answer by the following lemma.

Lemma 3

Given a degree of confidence x, let n_s and n_r be the time at which $\mathcal{S}_0$ and $\mathcal{R}$ are expected to achieve x, respectively. When $p_i = \frac{1}{k}$ for every $i : 1 \leq i \leq k$, the maximum cost c_0 of $\mathcal{S}_0$, such that $n_s \leq n_r$, is given as

$$\mathsf{c}_0 = \check{c} \cdot \frac{-\ln(1-x)}{x} \text{ for a constant } \check{c}.$$

Proof: First, we compute the time it takes $\mathcal{S}_0$ to achieve x depending on c_0 and k. Setting $f_s(n) = |\mathcal{D}|x$ gives

$$n = xk\mathsf{c}_0 \quad (19)$$

Then, we set the same time for $\mathcal{R}$ by substituting n. Setting $\bar{f}_r(n) = |\mathcal{D}|x$ yields

$$x = 1 - \left(1 - \frac{1}{k}\right)^n \quad (20)$$

$$= 1 - \left(1 - \frac{1}{k}\right)^{xk\mathsf{c}_0} \quad \text{[by Eqn. (19)]} \quad (21)$$

Solving for the maximum cost c_0 gives

$$1 - x = \left(1 - \frac{1}{k}\right)^{xk\mathsf{c}_0} \quad (22)$$

$$\ln(1-x) = xk\mathsf{c}_0 \ln\left(1 - \frac{1}{k}\right) \quad (23)$$

$$\frac{\ln(1-x)}{x} = -k\mathsf{c}_0 \ln\left(\frac{k-1}{k}\right) \quad (24)$$

$$\mathsf{c}_0 = \check{c} \cdot \frac{-\ln(1-x)}{x} \quad (25)$$

where

$$\check{c} = \left(k \ln\left(\frac{k}{k-1}\right)\right)^{-1} \quad (26)$$

■

Figure 2 shows for the segment from $x : 0.8 \leq x \leq 1$ the exact cost c_0 for $\mathcal{S}_0$ such that both techniques are expected to break even at a given degree of confidence. Giving the degree of confidence $x = 0.8$, $\mathcal{S}_0$ is expected to be more efficient than $\mathcal{R}$ according to the 1st PoEST only if the sampling cost of $\mathcal{S}_0$ is $\mathsf{c} < 2$. For $x = 0.99$, we see in Fig. 2 that the maximum sampling cost of $\mathcal{S}_0$ is $\mathsf{c}_0 = 4.65$ units of time so that $\mathcal{S}_0$ is expected to be more efficient than $\mathcal{R}$.

3.3 Bounds on the Expected Confidence Achieved by Random Testing

Under the simplified conditions of the example, where each partition has the same size, $|\mathcal{D}_1| = \cdots = |\mathcal{D}_k|$, we have shown that the confidence achieved per unit of time *decays exponentially* for random testing. In the following, we prove that this is the case for partitions of arbitrary sizes. Towards that, we define two quantities p_{min} and p_{max}.

$$p_{\max} = \max_{i=1}^{k}\{p_i\} \text{ and } p_{\min} = \min_{i=1}^{k}\{p_i\} \quad (27)$$

where the functions max and min compute the maximum and minimum number in a given set, respectively. Note that $p_{\max} \geq 1/k$ and $p_{\min} \leq 1/k$. We claim

Lemma 4 (*Approximate Bounds*)

$f_r(n)$ is bounded above and below approximately as

$$|\mathcal{D}|[1 - kp_{\min}e^{-np_{\min}}] \lesssim f_r(n) \lesssim |\mathcal{D}|[1 - kp_{\max}e^{-np_{\max}}]$$

Proof: Let $I_{\max} \subseteq \{1, 2, \ldots, k\}$ be the set of indices such that $p_{\max} \neq p_i$ iff $i \in I_{\max}$. For all $i \in I_{\max}$, let n_i be the point in time such that

$$n_i = \frac{\ln(p_{\max}) - \ln(p_i)}{p_{\max} - p_i} \tag{28}$$

This implies for all $n \geq n_i$

$$e^{np_{\max} - np_i} \geq e^{\ln(p_{\max}) - \ln(p_i)} \tag{29}$$

$$\frac{e^{-np_i}}{e^{-np_{\max}}} \geq \frac{p_{\max}}{p_i} \tag{30}$$

$$p_{\max}e^{-np_{\max}} \leq p_i e^{-np_i} \tag{31}$$

Let $n_{\max}$ be the point in time such that

$$n_{\max} = \max_{i \in I_{\max}}\{n_i\} \tag{32}$$

For all $n \geq n_{\max}$ we have

$$\begin{aligned}
\sum_{i=1}^{k} p_i e^{-np_i} &= \sum_{i \in I_{\max}} p_i e^{-np_i} + \sum_{i \notin I_{\max}} p_i e^{-np_i} \\
&= \sum_{i \in I_{\max}} p_i e^{-np_i} + \sum_{i \notin I_{\max}} p_{\max} e^{-np_{\max}} \\
&\qquad \text{[since } p_i = p_{\max} \text{ for } i \notin I_{\max}\text{]} \\
&\geq \sum_{i \in I_{\max}} p_{\max} e^{-np_{\max}} + \sum_{i \notin I_{\max}} p_{\max} e^{-np_{\max}} \\
&\qquad \text{[by Eqn. (31)]} \\
&= kp_{\max}e^{-np_{\max}}
\end{aligned}$$

Similarly, let $I_{\min} \subseteq \{1, 2, \ldots, k\}$ be the set of indices such that $p_i \neq p_{\min}$ iff $i \in I_{\min}$. Let $n_{\min}$ be the point in time such that

$$n_{\min} = \max_{i \in I_{\min}} \left\{ \frac{\ln(p_i) - \ln(p_{\min})}{p_i - p_{\min}} \right\} \tag{33}$$

We can show for all $n \geq n_{\min}$ that

$$\sum_{i=1}^{k} p_i e^{-np_i} \leq kp_{\min}e^{-np_{\min}} \tag{34}$$

So, for all $n \geq \max\{n_{\min}, n_{\max}\}$, we have

$$kp_{\max}e^{-np_{\max}} \leq \sum_{i=1}^{k} p_i e^{-np_i} \leq kp_{\min}e^{-np_{\min}} \tag{35}$$

Hence by Lemma 2 and Def. 8, we have

$$|\mathcal{D}|[1 - kp_{\min}e^{-np_{\min}}] \lesssim f_r(n) \lesssim |\mathcal{D}|[1 - kp_{\max}e^{-np_{\max}}] \tag{36}$$ ■

Thus $f_r(n)$ being asymptotically bounded above and below by functions having exponential decay also behaves like one.

3.4 Relative Efficiency of $\mathcal{S}_0$ in 1st PoEST

We evaluate the efficiency of the systematic testing technique $\mathcal{S}_0$ relative to that of random testing $\mathcal{R}$. Because of the additional analysis cost, sampling a test input using $\mathcal{S}_0$ takes c times longer than sampling a test input using $\mathcal{R}$. Since *in general* the achieved confidence per unit of time decays exponentially for $\mathcal{R}$ while it grows linearly for $\mathcal{S}_0$, there is a point where $\mathcal{S}_0$ and $\mathcal{R}$ are expected to break even. Its coordinates depend on the value of c.

Given a degree of confidence x, we compute the maximum cost c_0 such that the expected time it takes for $\mathcal{S}_0$ to achieve x is at most the same as the expected time it takes $\mathcal{R}$ to achieve x and $\mathcal{S}_0$ remains more efficient than $\mathcal{R}$.

Proposition 1

Given a degree of confidence $x : 1 - e^{-1} \leq x < 1$, let n_s and n_r be the units of time after which $\mathcal{S}_0$ and $\mathcal{R}$ are expected to achieve x, respectively. For all programs $\mathcal{P}$, the maximum cost c_0 of $\mathcal{S}_0$, such that $n_s \leq n_r$, is bounded above as

$$\mathsf{c}_0 \lesssim \frac{1}{ex - ex^2}$$

Proof: Fix a program $\mathcal{P}$ which in turn fixes the number of partitions k and also the probabilities p_i for all $i : 1 \leq i \leq k$. Let $\mathsf{c}_0^{\mathcal{P}}$ be the cost of $\mathcal{S}_0$, such that $n_s = n_r$ for $\mathcal{P}$. Now, setting $f_s(n_s) = |\mathcal{D}|x$ in Lemma 1 yields

$$n_s = n_r = xk\mathsf{c}_0^{\mathcal{P}} \tag{37}$$

Setting $f_r(n_r) = |\mathcal{D}|x$ in Lemma 2 gives

$$x \sim 1 - \sum_{i=1}^{k} p_i e^{-n_r p_i} \tag{38}$$

$$\gtrsim 1 - kp_{\min}e^{-n_r p_{\min}} \qquad \text{[by Lemma 4]} \tag{39}$$

$$\gtrsim 1 - \frac{kp_{\min}}{e^{xk\mathsf{c}_0^{\mathcal{P}} p_{\min}}} \qquad \text{[by Eqn. (37)]} \tag{40}$$

When solving for $\mathsf{c}_0^{\mathcal{P}}$ note that $0 < x < 1$ and $kp_{\min} > 0$,

$$e^{xk\mathsf{c}_0^{\mathcal{P}} p_{\min}} \lesssim \frac{kp_{\min}}{1 - x} \tag{41}$$

$$\mathsf{c}_0^{\mathcal{P}} \lesssim \frac{\ln\left(\frac{kp_{\min}}{1-x}\right)}{kxp_{\min}} \tag{42}$$

Let us denote $\frac{\ln\left(\frac{kp_{\min}}{1-x}\right)}{kxp_{\min}}$ as $h(k, p_{\min})$. From Eqn. (42),

$$\mathsf{c}_0 \leq \max_{\mathcal{P}}\{\mathsf{c}_0^{\mathcal{P}}\} \lesssim \max_{\mathcal{P}}\{h(k, p_{\min})\} \tag{43}$$

where $\max_{\mathcal{P}}$ denotes the maximum of the given quantity over all programs.

To find the value of $\max_{\mathcal{P}}\{h(k, p_{\min})\}$, we first relax the requirement that k takes integral values and allow k to range over the reals $\mathbb{R}$. By doing so we notice that $h(k, p_{\min})$ is a continuous function over $(\mathbb{R} \times [0, 1])$ which is differentiable everywhere. This allows us to use techniques from differential calculus to maximize $h(k, p_{\min})$ wrt $p_{\min}$ and k. [As we shall see below, $h(k, p_{\min})$ will have exactly one global extremum at some non-boundary point. Hence, the value of $\max_{\mathcal{P}}\{h(k, p_{\min})\}$, with the original requirement that k ranges over the discrete integral domain, will be attained at one of the two nearest integers.]

To derive all extrema of $h(k, p_{\min})$ wrt. $p_{\min}$, we set the partial derivative of $h(k, p_{\min})$ wrt $p_{\min}$ to 0.

$$\frac{\partial}{\partial p_{\min}} \frac{\ln\left(\frac{kp_{\min}}{1-x}\right)}{kxp_{\min}} = \frac{1 - \ln\left(\frac{kp_{\min}}{1-x}\right)}{kxp_{\min}^2} = 0 \quad (44)$$

This yields a critical point for $h(k, p_{\min})$ when

$$p_{\min} = \frac{e - ex}{k} \quad (45)$$

The second partial derivative of $h(k, p_{\min})$ wrt $p_{\min}$ is given by

$$\frac{\partial^2}{\partial p_{\min}^2} \frac{\ln\left(\frac{kp_{\min}}{1-x}\right)}{kxp_{\min}} = \frac{-3 + 2\ln\left(\frac{kp_{\min}}{1-x}\right)}{kxp_{\min}^3} \quad (46)$$

Hence for $h(k, p_{\min})$ to be maximal wrt $p_{\min}$ it must hold that

$$\frac{-3 + 2\ln\left(\frac{kp_{\min}}{1-x}\right)}{kxp_{\min}^3} < 0 \quad (47)$$

which yields

$$p_{\min} < \frac{e\sqrt{e}(1-x)}{k} \quad (48)$$

Since (45) satisfies (48) we have that $h(k, p_{\min})$ attains a maximum wrt $p_{\min}$ at $p_{\min} = \frac{e-ex}{k}$.

By a similar analysis we can demonstrate that $h(k, p_{\min})$ attains a maximum wrt k at $k = \frac{e-ex}{p_{\min}}$ which is the same as Eqn. (45). Plugging $p_{\min} = \frac{e-ex}{k}$ into $h(k, p_{\min})$ we get

$$\mathsf{c}_0 \lesssim \frac{1}{ex - ex^2} \quad (49)$$

Finally, to derive the bounds on the degree of confidence x for which the above inequality holds, note that it must also hold that $0 < p_{\min} \leq 1/k$ whence from Equation (45) we have

$$0 < \frac{e - ex}{k} \leq \frac{1}{k} \quad (50)$$

which gives

$$1 - e^{-1} \leq x < 1 \quad (51)$$

■

4 THE 2ND PROBLEM OF EFFICIENT TESTING

Exposing the most number of errors within a certain time budget is the 2nd Problem of Efficient Testing (2nd PoEST). So, given the same time budget $\hat{n}$, we compare the expected number of errors found by random testing $\mathcal{R}$ with the expected number of errors found by the systematic testing technique $\mathcal{S}_0$. After illustrating our main insights by an example, we investigate the efficiency of $\mathcal{S}_0$ and $\mathcal{R}$ w.r.t. the expected number of errors discovered. We show that the expected number of errors discovered per unit of time grows linearly for $\mathcal{S}_0$ while it decays exponentially for $\mathcal{R}$.

Note that Definition 1 of error-based partitioning states that failing inputs revealing *the same error* are grouped into the same error-revealing partition. This is reasonable because in practice several failing inputs may expose the same error. Thus, the number of error-revealing partitions discovered corresponds to the number of errors found.

Given a time bound $\hat{n}$, we find that the expected number of errors discovered by $\mathcal{R}$ within $\hat{n}$ time units is less than or equals that of $\mathcal{S}_0$ only if the analysis cost c incurred by $\mathcal{S}_0$ is less than $\frac{\hat{n}}{k} \cdot (1 - (1 - q_{\min})^{\hat{n}})^{-1}$, where k is the number of error-based partitions, and $q_{\min}$ is the fractional size of the *"smallest" error-revealing* partition.

Duran and Ntafos [16] define a quantity θ_i for every partition $\mathcal{D}_i$ which gives the probability of that partition to reveal an error. In our setting, θ_i can be defined as

$$\theta_i = \begin{cases} 1 & \text{if } \mathcal{D}_i \text{ is error-revealing} \\ 0 & \text{otherwise} \end{cases}$$

Then the total number of errors is given by $z = \sum_{i=1}^{k} \theta_i$.

4.1 Efficiency Analysis of Individual Techniques

First, we give the expected number of errors found per unit of time, i.e., the efficiency, for the systematic technique $\mathcal{S}_0$.

Lemma 5 (*Errors Found – Efficiency of $\mathcal{S}_0$*)

For the systematic testing technique $\mathcal{S}_0$, the expected number of errors discovered after n time units is

$$g_s(n) = \frac{z}{\mathsf{c}k} \cdot n$$

for $n : 0 \leq n \leq k$, where sampling one input takes c units of time.

Proof: By Definition 7, $\mathcal{S}_0$ performs n/c draws in n units of time. In this classical urn problem of sampling without replacement we shall call the discovery of an error(-revealing partition) a "success". The expected number of successes in n/c draws without replacement from a finite population k containing z successes is given by $\frac{z}{\mathsf{c}k} \cdot n$. ■

The expected number of errors discovered w.r.t the number of iterations grows linearly. As the cost c increases, the slope with the time-axis, $z/\mathsf{c}k$, of the line, $g_s(n)$, decreases.

Now, we look at the case for random testing.

Lemma 6 (*Errors Found – Efficiency of $\mathcal{R}$ [16]*)

For random testing $\mathcal{R}$, the expected number of errors discovered after n time units is

$$g_r(n) = k - \sum_{i=1}^{k} (1 - p_i\theta_i)^n$$

The proof is due to Duran and Ntafos [16]. By Definition 6, every iteration occurs in one unit of time.

4.2 Example for Equal-Sized Partitions

We illustrate the main insights for the simplified case where the size of each partition is equal, $|\mathcal{D}_1| = \cdots = |\mathcal{D}_k|$ and hence $p_i = \frac{1}{k}$ for all $1 \leq i \leq k$. In this setting, we demonstrate that the number of errors exposed decays exponentially for $\mathcal{R}$ while it grows linearly for $\mathcal{S}_0$. Later, this result is generalized for partitions of arbitrary size.

First, we derive the corollary of Lemma 6.

Corollary 2

For random testing $\mathcal{R}$ where $p_i = \frac{1}{k}$ for all $1 \leq i \leq k$, the expected number of errors found after n time units is

$$\begin{aligned} \bar{g}_r(n) &= z - z\,(1 - 1/k)^n \\ &= z - ze^{-\lambda n} \text{ where } \lambda = \ln\left([1 - 1/k]^{-1}\right) \end{aligned}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2487274, IEEE Transactions on Software Engineering

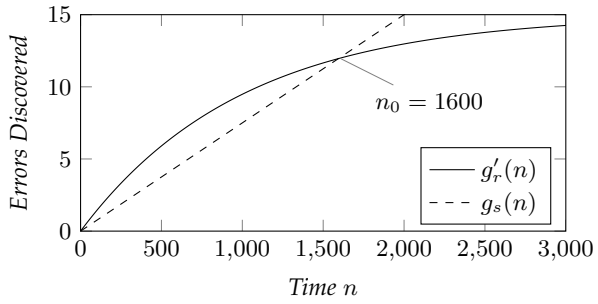

Fig. 3. On the average, $\mathcal{S}_0$ and $\mathcal{R}$ break even after 12 of 15 errors were discovered and 1600 random test inputs were sampled (when $\mathsf{c} = 2$, $z = 15$, $k = 1000$, $p_i = \frac{1}{k}$).

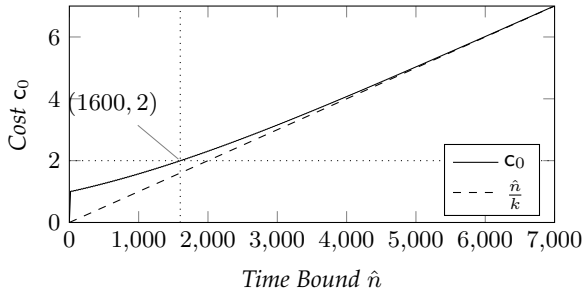

Fig. 4. The maximum cost c_0 increases approximately linearly as the given time bound $\hat{n}$ increases. If the average analysis cost of $\mathcal{S}_0$ exceeds c_0 for a given time bound $\hat{n}$, then $\mathcal{R}$ is generally more efficient than $\mathcal{S}_0$ (here for $p_i = \frac{1}{k}$ and $k = 1000$).

Proof: After setting $p_i = \frac{1}{k}$ in the formula of Lemma 6, we have for z number of partitions that $\theta_i = 1$ and for $k - z$ number of partitions that $\theta_i = 0$. Thus,

$$\bar{g}_r(n) = k - \sum_{i=1}^{k} \left(1 - \frac{\theta_i}{k}\right)^n \tag{52}$$

$$= k - \left((k - z) + z\left(1 - \frac{1}{k}\right)^n\right) \tag{53}$$

$$= z - z\left(1 - \frac{1}{k}\right)^n \tag{54}$$

$$= z - ze^{-\ln\left((1-\frac{1}{k})^{-1}\right)n} \tag{55}$$

■

The corollary shows that $\bar{g}_r(n)$ has exponential decay.

Figure 3 depicts the expected number of discovered errors per unit of time for random testing and $\mathcal{S}_0$ in our example configuration. As the cost c is 2, it takes $\mathcal{S}_0$ twice as long to sample a test input compared to $\mathcal{R}$. After 800 units of time, $\mathcal{S}_0$ discovered 6 of $z = 15$ errors on the average, while $\mathcal{R}$ discovered 2.2 errors *more*, on the average. After 1600 units of time, both techniques discovered 12 of $z = 15$ errors, on the average. This the point of time where both testing schemes, $\mathcal{S}_0$ and $\mathcal{R}$, are expected to break even.

There exists a time n_0 where $\bar{g}_r(n_0) = g_s(n_0)$ meet and $\mathcal{S}_0$ has discovered more errors than $\mathcal{R}$ for any $n > n_0$, on the average. To assess the *relative efficiency* of $\mathcal{S}_0$ we pose the following question: Given a time bound $\hat{n}$, what is the maximum cost c_0 for $\mathcal{S}_0$ such that $n_0 \leq \hat{n}$?

Lemma 7

In the case where $p_i = \frac{1}{k}$ for every $1 \leq i \leq k$, the maximum cost c_0 of the systematic testing technique $\mathcal{S}_0$ – such that the expected number of errors discovered by $\mathcal{S}_0$ is at least the same as the expected number of errors discovered by $\mathcal{R}$ in $\hat{n}$ units of time – is given as

$$\mathsf{c}_0 = \frac{\hat{n}}{k(1 - (1 - \frac{1}{k})^{\hat{n}})}$$

Proof: The proof follows directly from Lemma 5 and Corr. 2 when fixing n to $\hat{n}$ and setting $\bar{g}_r(\hat{n}) = g_s(\hat{n})$. ■

Notice that the maximum cost $\mathsf{c}_0 \sim \hat{n}/k$ as $\hat{n} \to \infty$.

Figure 4 depicts the exact cost c_0 for $\mathcal{S}_0$ such that both techniques are expected to break even at a given time $\hat{n}$. Giving a time bound of $\hat{n} = 1600$, the maximum cost is $\mathsf{c}_0 = 2$ and both techniques are expected to break even at $\hat{n}$ as shown in Figure 3. Increasing the time-bound $\hat{n}$, increases the maximum cost c_0 approximately proportionally.

4.3 Tight Bounds on the Expected Number of Errors Discovered for Random Testing

Under the simplified conditions of the example, where each partition has the same size, $|\mathcal{D}_1| = \cdots = |\mathcal{D}_k|$, we see that the efficiency of random testing *decays exponentially*. In the following, we show that this is the case for partitions of arbitrary sizes. Intuitively, random testing discovers many (error-revealing) partitions in the beginning and much less as the number of iterations increases.

Towards that, let $Q \subseteq \{p_1, \ldots, p_k\}$ be a set of probabilities such that $p_i \in Q$ iff $\theta_i = 1$ for all indices $1 \leq i \leq k$. Thus, Q is the set of p_i's corresponding to all the error-*revealing* partitions $\mathcal{D}_i$. We define two quantities

$$q_{\max} = \max\{q \mid q \in Q\} \text{ and } q_{\min} = \min\{q \mid q \in Q\} \tag{56}$$

where the functions max and min give the maximum and minimum elements in a given set, respectively. We have

Lemma 8 (*Tight bounds*)

Given a program $\mathcal{P}$, let k be the total number of partitions of the input space out of which z are error-revealing. Let

$$\lambda_{\min} = \ln\left(\frac{1}{1 - q_{\min}}\right) \text{ and } \lambda_{\max} = \ln\left(\frac{1}{1 - q_{\max}}\right)$$

Then, $z - ze^{-\lambda_{\min} n} \leq g_r(n) \leq z - ze^{-\lambda_{\max} n}$.

Proof:

$$g_r(n) = k - \sum_{i=1}^{k} (1 - \theta_i p_i)^n \tag{57}$$

$$= k - [\sum_{q_i \in Q} (1 - q_i)^n] - [\sum_{q_i \notin Q} 1] \tag{58}$$

$$= k - [\sum_{q_i \in Q} (1 - q_i)^n] - (k - z) \tag{59}$$

$$= z - \sum_{q_i \in Q} (1 - q_i)^n \tag{60}$$

Hence, we have

$$z - z(1 - q_{\min})^n \leq g_r(n) \leq z - z(1 - q_{\max})^n \tag{61}$$

$$z - ze^{-\lambda_{\min} n} \leq g_r(n) \leq z - ze^{-\lambda_{\max} n} \tag{62}$$

■

The function $g_r(n)$ being bounded above and below by exponentially decaying functions also behaves like one. That is, there exists a 3-tuple (a, b, λ) such that $g_r(n) = ae^{-\lambda n} + b$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2487274, IEEE Transactions on Software Engineering

4.4 Relative Efficiency of $\mathcal{S}_0$ in 2nd PoEST

We evaluate the efficiency of the systematic testing technique $\mathcal{S}_0$ relative to that of random testing $\mathcal{R}$. Because of the additional analysis cost, sampling a test input using $\mathcal{S}_0$ takes c times longer than sampling a test input using $\mathcal{R}$. Since *in general* the efficiency of $\mathcal{R}$, here w.r.t. discovering errors, decays exponentially while that of $\mathcal{S}_0$ grows linearly, there is a point in time where $\mathcal{S}_0$ and $\mathcal{R}$ are expected to break even. The coordinates of this point depend on the value of $\mathcal{S}_0$'s sampling cost c.

Given $\hat{n}$ units of time, we compute the maximum cost c_0 such that $\mathcal{S}_0$ remains more efficient than $\mathcal{R}$ according to the 2nd Problem of Efficient Software Testing. Specifically, we compute c_0 such that the expected number of errors discovered by $\mathcal{S}_0$ is at least the same as the expected number of errors discovered by $\mathcal{R}$ after the time budget of $\hat{n}$ time units is exhausted.

Proposition 2

Given a program $\mathcal{P}$, let k be the total number of error-based partitions out of which z are error-revealing. Given $\hat{n}$ units of time, let d_r and d_s be the expected number of error-revealing partitions discovered by the systematic testing technique $\mathcal{S}_0$ and random testing $\mathcal{R}$, respectively. Then, the maximum cost c_0 of $\mathcal{S}_0$, such that $d_r \leq d_s$, is given as

$$\mathsf{c}_0 \leq \frac{\hat{n}}{k} \cdot \left(1 - (1 - q_{\min})^{\hat{n}}\right)^{-1}$$

where $q_{\min}$ is defined as in Eqn. (56).

Proof: Setting $g_s(\hat{n}) = g_r(\hat{n})$ yields

$$\frac{z\hat{n}}{k\mathsf{c}_0} = k - \sum_{i=1}^{k}(1 - p_i\theta_i)^{\hat{n}} \tag{63}$$

$$\frac{z\hat{n}}{k\mathsf{c}_0} \geq z - z(1 - q_{\min})^{\hat{n}} \qquad \text{[By Lemma 8]} \tag{64}$$

Solving for c_0 having $\hat{n} > 0$, $k > 0$, and $z \geq 0$ gives

$$\mathsf{c}_0 \leq \frac{1}{k} \cdot \frac{\hat{n}}{1 - (1 - q_{\min})^{\hat{n}}} \tag{65}$$ ■

5 A Hybrid Testing Technique $\mathcal{H}$

Given a systematic technique $\mathcal{S}$ that discovers a partition with every input sampled, subject to the sampling cost c, there exists a hybrid testing technique $\mathcal{H}$ that, at *any* time, has discovered at least as many partitions as the random technique $\mathcal{R}$ *and* at least as many partitions as the systematic technique $\mathcal{S}$.[2] Since $\mathcal{S}$ is expected to discover all partitions eventually, while $\mathcal{R}$ is not, there *must* be a time when it is best to switch from $\mathcal{R}$ to $\mathcal{S}$ to gain optimal efficiency.

For simplicity, we assume i) that the sampling cost c of $\mathcal{S}$ is known and constant, and ii) that the switch itself takes no time at all. In practice, the sampling cost of $\mathcal{S}$ may be a function over time n. In that case, $\mathsf{c}(n)$ of $\mathcal{S}$ needs to be derived *empirically* by measuring the time it takes to generate test cases as compared to a random generator.[3] Also, in practice the cost of measuring the number of partitions that $\mathcal{R}$ has already discovered and the cost of the switch itself should be considered.

Algorithm 1 Hybrid Testing Technique $\mathcal{H}$

Require: Systematic Testing $\mathcal{S}$ with sampling cost c
Require: Random Testing $\mathcal{R}$ with sampling cost 1
Require: Program $\mathcal{P}$ with k partitions in input space $\mathcal{D}$
1: let *time*, *nDisc* := 0
2: **while** *nDisc* $< k$ **do**
3: let *time* := *time* + 1
4: sample t from $\mathcal{D}$ using $\mathcal{R}$
5: **if** t sampled from undiscovered partition **then**
6: let *nDisc* := *nDisc* + 1
7: let T_{nDisc} := *time*
8: let $\mathbb{E}[T_{nDisc+1}]$:= *regression*$(\{T_i \mid 1 \leq i \leq nDisc\})$
9: **if** $\mathbb{E}[T_{nDisc+1}] > \mathsf{c}$ **then** *break*; **end if**
10: **end if**
11: **end while**
12: **while** *nDisc* $< k$ **do**
13: let *nDisc* := *nDisc* + 1
14: sample t from $\mathcal{D}$ using $\mathcal{S}$
15: **end while**

In Algorithm 1, we define the hybrid technique $\mathcal{H}$ that tests the program using $\mathcal{R}$ until the time to discover the next partition exceeds c units of time and then switches to testing using $\mathcal{S}$. In Algorithm 1, $\mathcal{H}$ samples test input using $\mathcal{R}$ until the expected time it takes $\mathcal{R}$ to discover (not sample!) the next partition exceeds the expected time c it takes $\mathcal{S}$ to sample (and thus discover) the next partition. The expected time it takes $\mathcal{R}$ to discover the next partition is not difficult to predict, given sufficiently many previous random samples. From Lemma 8, we know that the expected number of partitions discovered decays exponentially over time. Hence, for each program there exists a 3-tuple (a, b, λ), such that $h(n) = ae^{-\lambda n} + b$ gives the expected number of partitions that $\mathcal{R}$ discovers over time n. In Alg. 1, the function *regression* in line 8 takes the vector of the previous points in time, when $\mathcal{R}$ discovered a new partition to fit to an exponential curve and predict the expected time-to-next-discovery. In line 9, $\mathcal{H}$ switches to $\mathcal{S}$.

The efficiency of the hybrid technique is intuitively explained in Figure 5. The hybrid technique $\mathcal{H}$ switches from $\mathcal{R}$ to $\mathcal{S}$ at that precise moment when $\mathcal{S}$ is expected to discover the most number of partitions *per unit time*.

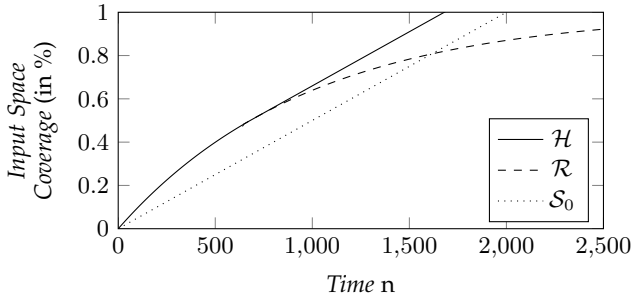

Fig. 5. $\mathcal{H}$ performs better than $\mathcal{S}_0$ and $\mathcal{R}$ ($\mathsf{c} = 2$, $k = 1000$, $p_i = \frac{1}{k}$).

2. Notice that $\mathcal{S}_0$ as defined in Def. 7 is an instance of $\mathcal{S}$ where the *order* in which the partitions are sampled is chosen at random.

3. Note that the partitioning need not be *error-based* for $\mathcal{H}$ to discover at least as many partitions as $\mathcal{R}$ or $\mathcal{S}$; a partition could also correspond to the set of inputs exercising the same path [11].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2487274, IEEE Transactions on Software Engineering

5.1 Expected Time To Next Discovery

Given a systematic testing technique $\mathcal{S}$ and a program $\mathcal{P}$, we can compute the number of partitions that have to be discovered using random testing $\mathcal{R}$ until the expected time to the *next* discovery exceeds the sampling cost c of $\mathcal{S}$. In the following, we discuss a simple example when we are given that $p_i = 1/k$ for all $i : 1 \leq i \leq k$.

Lemma 9 (*Expected Time to Switch* $(\mathcal{S}_0, p_i = \frac{1}{k})$)

Given sampling cost c *of* $\mathcal{S}$ *and a program* $\mathcal{P}$ *where* $p_i = \frac{1}{k}$ *for all* $i : 1 \leq i \leq k$*, the expected time to the next discovered partition exceeds* c *after* $k(1 - 1/\mathsf{c})$ *partitions have been discovered using* $\mathcal{R}$*.*

Proof: Let the random variables $T_j,\ 1 \leq j \leq k$ denote the time units taken from the discovery of the $j-1$-th partition to the discovery of the j-th partition. Moreover, let p denote $\frac{1}{k}$. The following are easy to observe.

$$\mathbb{E}[T_1] = 1 \tag{66}$$

$$\mathbb{E}[T_2] = 1(1-p) + 2p(1-p) + 3p^2(1-p) + \ldots = 1/(1-p) \tag{67}$$

$$\mathbb{E}[T_3] = 1(1-2p) + 2 \cdot 2p(1-2p) + 3(2p)^2(1-2p) + \ldots = 1/(1-2p) \tag{68}$$

$$\ldots$$

$$\mathbb{E}[T_j] = 1/(1-(j-1)p) \qquad \text{where } 1 \leq j \leq k \tag{69}$$

Let j_0 be the number of partitions that have to be discovered using $\mathcal{R}$ until the expected time to discover the next partition exceeds c. Then,

$$\mathbb{E}[T_{j_0+1}] > \mathsf{c} \tag{70}$$

Using Eqn. (69) and substituting back the value of p

$$j_0 > k\,(1 - 1/\mathsf{c}) \tag{71}$$

■

Note, from the above proof that for $\mathcal{R}$, $\mathbb{E}[T_j]$ increases strictly with j. We show that this is the case for non-equi-sized partitions, too.

Lemma 10 (*Monotonicity of* $\mathcal{R}$)

Let the random variables $T_j,\ 1 \leq j \leq k$ *denote the time units taken from the discovery of the* $j-1$*-th partition to the discovery of the* j*-th partition for* $\mathcal{R}$*. Then for all* $i, j : 1 \leq i < j \leq k$*, we have* $\mathbb{E}[T_i] < \mathbb{E}[T_j]$*. That is,* $\mathbb{E}[T_j]$ *increases strictly with* j*.*

Proof: Let $d_j,\ 1 \leq j \leq k$ denote the probability that $\mathcal{R}$ discovers a partition after the discovery of $j-1$ partitions. Note that $d_1 = 1$. Because with every discovery, the size of the space of undiscovered partition decreases and hence the probability to sample from that space also decreases, we have

$$d_j < d_i \text{ for all } i : 1 \leq i < j \leq k \tag{72}$$

So, the expected time for the j-th discovery is given as

$$\mathbb{E}[T_j] = 1 \cdot d_j + 2(1-d_j)d_j + 3(1-d_j)^2 d_j + \ldots \tag{73}$$

$$= 1/d_j \tag{74}$$

Thus from (72) we have

$$\mathbb{E}[T_i] < \mathbb{E}[T_j] \text{ for all } i : 1 \leq i < j \leq k \tag{75}$$

■

5.2 Efficiency of $\mathcal{H}$ over $\mathcal{R}$ and $\mathcal{S}$

We can show that the hybrid testing technique $\mathcal{H}$, at any point in time n, has discovered at least as many partitions as both its constituent techniques $\mathcal{R}$ and $\mathcal{S}$ in expectation.

Proposition 3

Let $1 \leq j \leq k$ *and suppose* $n_r(j), n_s(j)$ *and* $n_h(j)$ *are random variables denoting the respective times taken by* $\mathcal{R}, \mathcal{S}$ *and* $\mathcal{H}$ *to discover* j *partitions. Then*

$$\mathbb{E}[n_h(j)] \leq \mathbb{E}[n_r(j)] \text{ and } \mathbb{E}[n_h(j)] \leq \mathbb{E}[n_s(j)]$$

Proof: By construction, $\mathcal{H}$ employs $\mathcal{R}$ and switches to $\mathcal{S}$ when the cost to discover the next partition using $\mathcal{R}$ exceeds c. Given a program $\mathcal{P}$, let j_0 be the expected number of partitions discovered before $\mathcal{H}$ switches from $\mathcal{R}$ to $\mathcal{S}$. Let T_j^h, T_j^r, T_j^s for all $j : 1 \leq j \leq k$ be the random variables denoting the time units taken from the discovery of the $j-1$-th partition to the discovery of the j-th partition, by $\mathcal{H}$, $\mathcal{R}$, and $\mathcal{S}$, respectively. Note, that for all $j : 1 \leq j \leq k$

$$\mathbb{E}[T_j^s] = \mathsf{c} \tag{76}$$

We distinguish two cases: (i) $j \leq j_0$ and (ii) $j > j_0$. **If (i)** $j \leq j_0$, according to Alg. 1, $\mathbb{E}[T_j^h] = \mathbb{E}[T_j^r]$, and since $\mathcal{H}$ hasn't made the switch from $\mathcal{R}$ to $\mathcal{S}_0$ we have

$$\mathbb{E}[T_j^h] < \mathsf{c} \tag{77}$$

$$\mathbb{E}[T_j^h] < \mathbb{E}[T_j^s] \qquad \text{[by Eqn. (76)]} \tag{78}$$

If (ii) $j > j_0$, according to Alg. 1, $\mathbb{E}[T_j^h] = \mathbb{E}[T_j^s]$. From Lemma 10, we know that $\mathbb{E}[T_j^r]$ strictly increases with j. Since, $\mathbb{E}[T_{j_0+1}^r] > \mathsf{c}$, we know that for all $j : j_0 < j \leq k$

$$\mathsf{c} < \mathbb{E}[T_j^r] \tag{79}$$

$$\mathbb{E}[T_j^s] < \mathbb{E}[T_j^r] \qquad \text{[by Eqn. (76)]} \tag{80}$$

$$\mathbb{E}[T_j^h] < \mathbb{E}[T_j^r] \qquad \text{[by Alg. 1]} \tag{81}$$

In both cases (i) and (ii), we have shown that for all $j : 1 \leq j \leq k$

$$\mathbb{E}[T_j^h] \leq \mathbb{E}[T_j^r] \text{ and} \tag{82}$$

$$\mathbb{E}[T_j^h] \leq \mathbb{E}[T_j^s]. \tag{83}$$

Thus,

$$\sum_{i=1}^{j} \mathbb{E}[T_j^h] \leq \sum_{i=1}^{j} \mathbb{E}[T_j^r] \tag{84}$$

$$\mathbb{E}\left[\sum_{i=1}^{j} T_j^h\right] \leq \mathbb{E}\left[\sum_{i=1}^{j} T_j^r\right] \qquad \text{[by lin. of exp.]} \tag{85}$$

$$\mathbb{E}[n_h(j)] \leq \mathbb{E}[n_r(j)] \tag{86}$$

Similarly, we can show that $\mathbb{E}[n_h(j)] \leq \mathbb{E}[n_s(j)]$. ■

6 Simulation Experiments

While the efficiency of the systematic technique $\mathcal{S}_0$ is independent of the distribution of partition size,[4] the random technique $\mathcal{R}$ performs differently as $\mathbf{p}$ varies. Intuitively, $\mathcal{R}$ is likely to discover bigger partitions earlier than smaller ones. Using simulation, we study the impact of different distributions of $\mathbf{p}$ on the efficiency of $\mathcal{R}$ and the maximum cost c_0 of $\mathcal{S}_0$ such that $\mathcal{S}_0$ remains at least as efficient as $\mathcal{R}$.

4. The efficiency of $\mathcal{S}_0$ is independent of the distribution of partition size since i) we prove a *linear increase* of errors discovered / confidence achieved over time and ii) all partitions are discovered in $k\mathsf{c}$ time units.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2487274, IEEE Transactions on Software Engineering

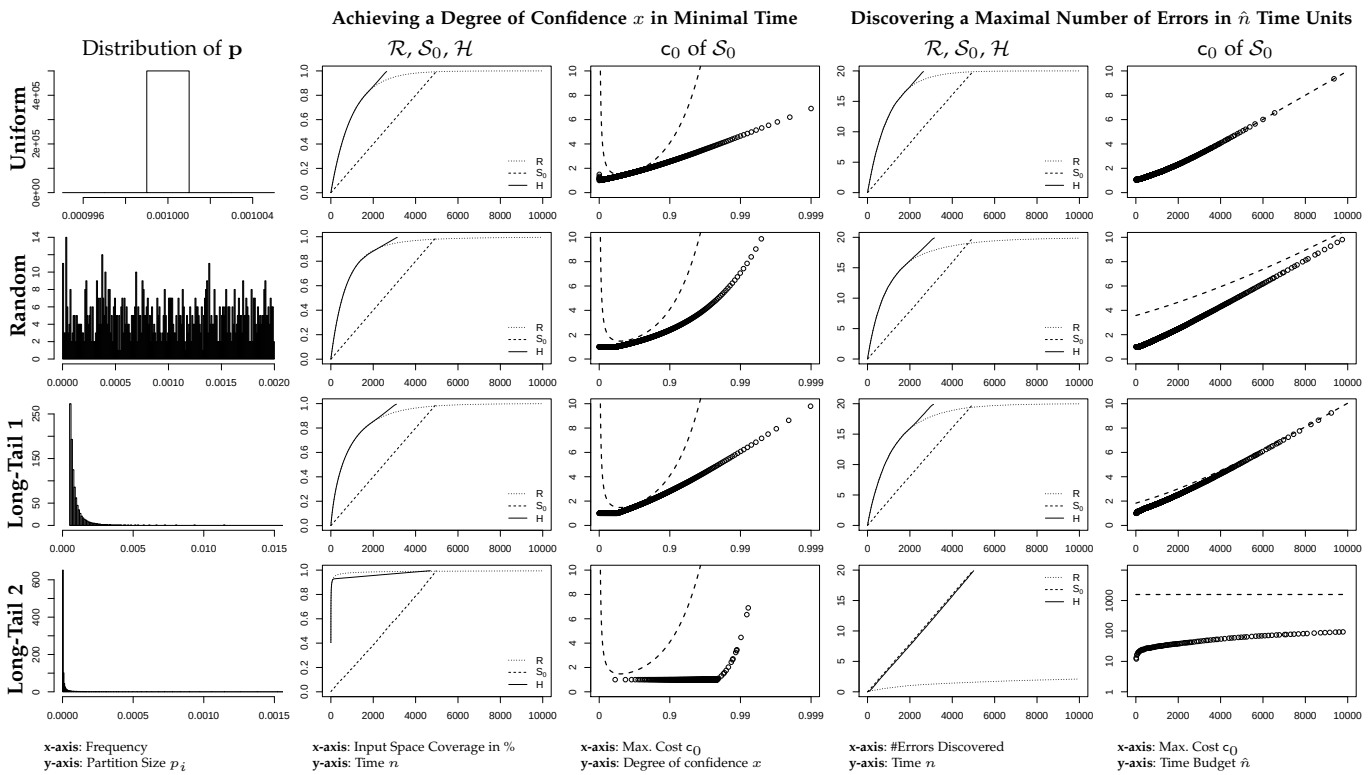

Fig. 6. *Efficiency plots for* $\mathcal{R}$, $\mathcal{S}_0$, *and* $\mathcal{H}$. For each distribution (Col. 1), we show the degree of confidence achieved over time (Col. 2) and the maximum sampling cost c_0 of $\mathcal{S}_0$ (Col. 3) to remain more efficient than $\mathcal{R}$, given a certain degree of confidence x. We also show the number of errors exposed over time (Col. 4) and the maximum sampling cost c_0 of $\mathcal{S}_0$ (Col. 5) to remain more efficient than $\mathcal{R}$, given a certain time budget $\hat{n}$. In Col. 3 and Col. 5, the maximum cost c_0 of $\mathcal{S}_0$ is shown as points while the *upper bound* on c_0 (see Prop. 1 & Prop. 2) is shown as dashed line.

Setup. All simulations were conducted in `R` on a MacBook Pro with 16GB of memory and a 2.3GHz i7 CPU. We compute the mean of *1000 repetitions* of each experiment. The number of partitions was fixed at $k = 1000$. In total, we performed $24{,}000$ simulation experiments (2 testing goals, 3 testing techniques, 4 distributions, and 1000 repetitions).

Testing Techniques. We implemented the three techniques discussed in this article. $\mathcal{R}$ samples *with replacement* bigger partitions *more likely* than smaller partitions taking 1 time unit per sampling. $\mathcal{S}_0$ samples *without replacement* bigger partitions *as likely* as smaller partitions. In Col. 2 and 4, $\mathcal{S}_0$ takes c=5 time units per sampling. $\mathcal{H}$ works similar as in Alg. 1. However, it switches when the *actual* time-since-last-discovery exceeds $\mathsf{c} \cdot \mathsf{c}$, which might be slightly after the *expected* time when time-to-next-discovery exceeds c.

Distributions of p_i. We chose the uniform, a random, and two long-tail distributions for the size of the partitions ($|D_i| = p_i|\mathcal{D}|$). The histogram featuring the frequencies of partition sizes is shown for each distribution in the first row of Figure 6. The *uniform* distribution is computed as $p_i = 1/k$ for every $i : 1 \leq i \leq k$. The *random* distribution assigns each partition a random size. The *long-tail* distributions are instances of the Zipf distribution (for $s = 0.5$ and $s = 2$). Intuitively, Zipf yields a very large number of very small partitions and a very small number of very large partitions.

Distributions of θ_i. There are a total of 20 error-revealing partitions that are selected without replacement from the set of all $k = 1000$ partitions where a partition $\mathcal{D}_i$ is selected with probability $1-p_i$. In other words, the smaller partitions are more likely to be error-revealing. The 980 remaining partitions do not reveal an error.

Results. Figure 6 shows for each distribution (Col. 1), the efficiency of $\mathcal{R}$, $\mathcal{S}_0$, $\mathcal{H}$ and the maximum cost of $\mathcal{S}_0$ if the goal is to achieve a given degree of confidence x in minimal time (Col. 2-3), and the efficiency of $\mathcal{R}$, $\mathcal{S}_0$, $\mathcal{H}$ and the maximum cost of $\mathcal{S}_0$ if the goal is to expose a maximal number of errors within a given time budget $\hat{n}$ (Col. 4-5). We observe:

- **(O1) The hybrid testing technique $\mathcal{H}$ has a *similar efficiency* than the most efficient of both, $\mathcal{R}$ or $\mathcal{S}_0$.** *Column 2*: For all distributions, $\mathcal{H}$ can establish the degree of confidence $x = 1.0$ *significantly earlier* than $\mathcal{S}_0$. Except for `Long-tail 2`, $\mathcal{H}$ is always *more efficient* than both its constituent techniques, $\mathcal{R}$ and $\mathcal{S}_0$, in terms of achieving a degree of confidence x in minimal time. Only for `Long-tail 2` and for some period of time does $\mathcal{H}$ achieve *slightly less* confidence than the most efficient of both $\mathcal{R}$ or $\mathcal{S}_0$. *Column 4*: For all distributions except `Long-tail 2`, $\mathcal{H}$ is *more efficient* than both its constituent techniques in terms of revealing a maximal number of errors in a given time. For `Long-tail 2`, at any time $\mathcal{H}$ has a *similar efficiency* than the most efficient of $\mathcal{R}$ or $\mathcal{S}_0$.
- **(O2) The asymptotic bound on c_0 is *not very tight* if the goal is to achieve confidence x in minimal time.** For instance, given degree of confidence $x = 0.99$, for all of the distributions the actual maximum cost c_0 of $\mathcal{S}_0$ never exceeds 7 units of time while our upper bound on c_0 gives about 37 units of time. In other words, for $x > 0.99$ our asymptotic bound allows $\mathcal{S}_0$ to be more than five times slower than it actually should be before it guarantees $\mathcal{R}$ to be more efficient than $\mathcal{S}_0$. Consequently, our upper bound is not tight.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2487274, IEEE Transactions on Software Engineering

- **(O3) c_0 increases with the *skewness* of the dist. of p_i if the goal is to expose max. errors in $\hat{n}$ time units.** Specifically, random testing $\mathcal{R}$ performs poorest for the `Long-Tail 2` distribution, where the majority of the 10^{-3} partitions cover less than 10^{-6}-th of the input space. $\mathcal{R}$ exposes less than 3 of 20 errors after 10,000 sampled test inputs, on average (Row 4, Col. 4). This allows $\mathcal{S}_0$ to take up to $\mathsf{c}_0 = 100$ times longer to sample a test input than $\mathcal{R}$ while still exposing at least the same number of errors in $\hat{n} = 10{,}000$ time units (Row 4, Col. 5).[5] Our theoretical bound is *magnitudes* higher than the actual value of c_0 for $\mathcal{S}_0$.

7 Practical Implications

To analyze the efficiency of automated testing in general, we construct a mathematical, probabilistic model of automated testing that hinges upon assumptions made about the real world. After repeating these assumptions, we discuss (A) their validity for realistic testing techniques, and (I) the implications of our theoretical results on the real world.

- **(A1) $\mathcal{S}_0$ can *prove* the absence of errors eventually**. In practice, realistic systematic testing techniques $\mathcal{T}_0$ are much less effective than our hypothetical, ideal technique $\mathcal{S}_0$. For example, consider a (high quality) test suite that is 100% branch coverage, MCDC coverage, path coverage and mutation-adequate, and also executes successfully on the program. Can we conclude that the program is correct? – No, because the absence of a failing test case does not imply the absence of errors in the program [7]. This is because complete certainty about the "true" error-based partitioning is unattainable [20]. Consequently, $\mathcal{T}_0$ with some degree of uncertainty samples some partitions several times and others not at all. The degree of uncertainty depends directly on the analysis cost. The more comprehensive the analysis, the more effective is the testing technique. It follows that

> **(I1)** ***The maximum sampling cost for realistic techniques $\mathcal{T}_0$ is likely less than the maximum sampling cost c_0 that we give for $\mathcal{S}_0$.*** *In practice, to approach the effectiveness of $\mathcal{S}_0$, we need to increase the analysis cost which in turn decreases the efficiency of the testing technique!*

- **(A2) $\mathcal{S}_0$ takes *constant time* c to sample one test input.** In practice, the sampling cost for realistic systematic techniques $\mathcal{T}_0$ may be a function that increases with testing time or program size. For example, consider coverage-based testing. It requires almost no analysis to sample an initial set of inputs that cover much of the source code. However, it becomes increasingly difficult to cover the remaining few uncovered code elements [30], [31]. So, the sampling of the test inputs takes increasingly longer. However, the average sampling cost for $\mathcal{T}_0$ must remain below c_0 for $\mathcal{S}_0$!

> **(I2)** ***Given the same sampling cost for the first test input, the maximum sampling cost for realistic techniques $\mathcal{T}_0$ is likely less than c_0 for $\mathcal{S}_0$.*** *The time to sample a test input for $\mathcal{T}_0$ likely increases as a function on time, number of tests generated, or the size of the program. In that case, $\mathcal{T}_0$ becomes less efficient over time while $\mathcal{S}_0$ remains just as efficient.*

5. Recall that we choose *smaller partitions to be more likely to be error-revealing* so that $\mathcal{R}$ may perform better otherwise (Dist. of θ_i).

- **(A3) Input partitioning into *error-based* subdomains.** In Def. 1, we define error-based partitioning to set up our investigations of testing efficiency in terms of *errors revealed* and the confidence achieved in the program's *correctness*. However, there is no reason why the partitioning should not be target-based, path-based, or differential, for example. *Target-based partitioning* yields subdomains for which all inputs either do or do not reach a certain target in the source. *Differential partitions* [15] are difference- and equivalence-revealing subdomains in the context of regression testing. *Path-based partitioning* [11], [32] groups all inputs into one partition that exercise the same path.

> **(I3)** ***The bounds on c_0 for $\mathcal{S}_0$ hold for disjoint input subdomains that are homogeneous w.r.t. other properties,*** *for instance, if the goal is to cover a maximal number of paths within a given time budget:*
>
> **Question:** We have a program with $k = z = 10^6$ paths where the path with the least probability to be exercised is of fractional size $q_{\min} = 10^{-8}$. We have two testing tools: a symbolic execution tool $\mathcal{S}'$ that exercises each path – one at a time, chosen uniformly at random from paths not exercised – and a random testing tool $\mathcal{R}$ that takes $10ms$ to generate and execute a test case. Finally, we only have one hour ($\hat{n} = 1h$) to exercise as many paths as possible. Which technique should we choose, $\mathcal{R}$ or $\mathcal{S}'$?
>
> **Answer:** We choose $\mathcal{S}'$ only if generating and executing one test case takes, on the average, less than about $1s$!

In practice, finding $q_{\min}$ while possible may not be viable; e.g., using symbolic execution and model counting the number of inputs exercising a certain path can be computed [32].

- **(A4) $\mathcal{S}_0$ samples error-based partitions in *random order*.** In Definition 7, we define testing technique $\mathcal{S}_0$ such that it samples each partition exactly once (cf. (A1)). However, we also specify that the partition that is sampled next is chosen uniformly at random. This assumption holds for instance for symbolic execution tools that exercise each path one at a time, chosen uniformly at random from paths not exercised. This assumption may not hold for other testing techniques that discover large partitions earlier than small partitions.

> **(I4)** ***The bounds on the maximum sampling cost hold for realistic testing techniques $\mathcal{T}_0$ with a similar sampling scheme than $\mathcal{S}_0$,*** *i.e., those that choose uniformly at random from the set of undiscovered parititons which partition is to be sampled next.*

- **(A5) $\mathcal{R}$ samples from input space *uniformly at random*.** In our probabilistic analysis, we assume that $\mathcal{R}$ chooses an input uniformly at random from the set of *all* program inputs. In practice, it is unlikely that any existing random test generator satisfies this assumption [19]. For instance, there may be bias towards producing small inputs, or dependence among the sampled tests such that new inputs are produced from previous valid ones in a feedback-directed manner.

- **(A6) Input space is *bounded*; errors are *deterministic*.** *Boundedness*: In practice, no program can take infinite input. Hence, our assumption that the program's input space is bounded is realistic. The input domain can be arbitrarily large with an arbitrarily large number of error-based partitions that may never all be discovered in any practical time. Yet, our bounds are applicable since k and $|\mathcal{D}|$ are finite.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2487274, IEEE Transactions on Software Engineering

Determinism: We assume that executing a test case that failed once, does always fail for the tested (unmodified) program. This is also satisfied if a model that *renders* a test execution deterministic, like a specific thread schedule, is constituent of the test case (and input space, respectively). However, for many test generators indeterminism is an open problem.

- **(A7) Works correctly for $x\%$ of its valid, typical input.** Consider a program that takes XML files as input. Then, $99.99\cdots99\%$ of random strings are effectively *invalid input*. It may seem that sampling even one test input using $\mathcal{R}$ achieves degree of confidence $x > 0.999$ suggesting that the program works correctly for more than 99.9% of its input. However, as long as no program analysis is involved we can give both test generators the *same power* while retaining the validity of our efficiency analysis: If we assume $\mathcal{S}_0$ to generate only valid input, then we should assume $\mathcal{R}$ to generate only valid input, too. After all, we have c of $\mathcal{S}_0$ represent the *additional* time for *program analysis* and defined to be a *factor* of the time it takes $\mathcal{R}$ to sample a (valid, typical) input. Thus, our bound holds even if we want to establish that the program works correctly for $x\%$ of its *valid, typical input*.

(I5) If we want to establish whether any program works correctly for $x = 99\%$ of its input, we can compute a bound c_0' on the time that a realistic technique $\mathcal{T}_0$ takes on average to generate and execute a test case and check its outcome such that some random test generator tool $\mathcal{R}$ is expected to achieve x earlier than $\mathcal{T}_0$* if *$\mathcal{T}_0$ exceeds $\mathsf{c}_0' = 37$ times the time that $\mathcal{R}$ takes on average to generate and execute a test case and check its outcome.* $\mathcal{T}_0$ has the same sampling scheme as $\mathcal{S}_0$ but may be less effective.

The „class of nines" for a degree of confidence x is directly proportional to the magnitude of the maximum analysis cost. The class of nines for degree of confidence x is computed as $\lfloor -\log_{10}(1-x) \rfloor$, where $\lfloor . \rfloor$ is the floor function.

confidence x	class of nines	bound on c_0
90%	1 nine	$\mathsf{c}_0 < 4.1 * 10^0$
99%	2 nines	$\mathsf{c}_0 < 4 * 10^1$
99.99%	4 nines	$\mathsf{c}_0 < 4 * 10^3$
99.9999%	6 nines	$\mathsf{c}_0 < 4 * 10^5$

8 Conclusion

In this paper we presented strong, elementary, theoretical results about the efficiency of automated software testing. For thirty years [16], we have struggled to understand how automated random testing and systematic testing seem to be almost *on par* [4], [5], [7], [17], [18], [33], [34].

Researchers in Software Engineering have spent much time and effort developing *highly effective* testing techniques; in fact, so effective that we can use testing even to *prove* the correctness of a program [26], [35]. In practice however, companies develop very large programs and have only limited time for testing. Given the choice of two testing tools, the developer would choose that which produces good results *faster*. Efficiency is key for testing tools.

Instead of seeking to increase the effectiveness of automated software testing, we should take time limitations into account and increase the efficiency of automated software testing.

In this work, we have provided a uniform mathematical framework for modeling the efficiency of software testing which is elementary and intuitive. In this framework, we showed that even a highly effective systematic testing technique is inefficient compared with random testing if the time for program analysis and test generation/execution is relatively too high. We explored two notions of *testing efficiency* that may be the main goals of automated software testing: i) to show in minimal time the correctness of a program for a given percentage of the program's input domain (Sec. 3) and ii) to discover a maximal number of errors within a given time bound (Sec. 4).

We defined a systematic testing technique $\mathcal{S}_0$ that is most effective in terms of both the above notions. Subsequently, we explored the efficiency of $\mathcal{S}_0$ again in terms of both the above notions. We also discussed how these results *generalize*, e.g., if the goal is to reach many targets, exercise many paths, or expose many differences, and how these results *apply* to realistic testing techniques (Sec. 7): Since realistic techniques with the same sampling scheme and cost as $\mathcal{S}_0$ are certainly less effective, they are trivially also less efficient. We believe that our work can also provide the formal framework to explore the efficiency of testing techniques other than $\mathcal{S}_0$.

For both goals of efficient software testing, we showed that there exists a bound on the time that $\mathcal{S}_0$ can take per test case beyond which $\mathcal{R}$ performs better than $\mathcal{S}_0$ on the average. Moreover, if the goal is to achieve degree of confidence x in minimal time, this bound *depends asymptotically only on* x. This has implications on the *scalability* of $\mathcal{S}_0$: If the time c to analyze the program increases with program size, for any testing technique there exists a program large enough that $\mathcal{R}$ is always expected to achieve x earlier.

Using insights from the above, we designed a hybrid testing technique $\mathcal{H}$ that starts with $\mathcal{R}$ but switches to $\mathcal{S}_0$ at that precise moment when $\mathcal{S}$ is expected to discover the most number of partitions per unit time. It is different from earlier seeding techniques [36], [37] (e.g., run $\mathcal{R}$ for 60sec, then run $\mathcal{S}$) in that $\mathcal{H}$ is clearly more systematic about *when* to switch to achieve optimal efficiency. We showed that $\mathcal{H}$ performs similarly or better than the most efficient of both. That $\mathcal{H}$ can be instantiated with techniques other than $\mathcal{S}_0$ demonstrates that the technique is robust and generic.

Finally, we conducted 24,000 simulation experiments with varying parameters. We observed that i) $\mathcal{H}$ has a similar efficiency than the most efficient of both, $\mathcal{R}$ or $\mathcal{S}_0$, ii) the asymptotic bound on c_0 is not very tight if the goal is to achieve confidence x in minimal time, and iii) c_0 can be significantly larger if the input space is partitioned such that there is a small number of huge and a very large number of very tiny partitions if the goal is to expose a maximal number of errors in $\hat{n}$ time units.

Acknowledgments

We would like to thank our colleagues Abhijeet Banerjee and Dr. Konstantin Rubinov for the engaging discussions about this paper. We also thank the anonymous reviewers for their valuable feedback. This work was partially supported by Singapore's Ministry of Education research grants MOE2010-T2-2-073 and MOE-2011-T2-2-012. The first author is funded by an ERC advanced grant 'SPECMATE'.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSE.2015.2487274, IEEE Transactions on Software Engineering

REFERENCES

[1] M. Böhme and S. Paul, "On the efficiency of automated testing," in *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2014, 2014, pp. 632–642.

[2] M. Böhme, B. C. d. S. Oliveira, and A. Roychoudhury, "Regression tests to expose change interaction errors," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2013, 2013, pp. 334–344.

[3] M. Böhme and A. Roychoudhury, "Corebench: Studying complexity of regression errors," in *Proceedings of the 23rd ACM/SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA, 2014, pp. 398–408.

[4] E. J. Weyuker and B. Jeng, "Analyzing partition testing strategies," *IEEE Transactions on Software Engineering*, vol. 17, pp. 703–711, July 1991.

[5] E. Weyuker and T. Ostrand, "Theories of program testing and the application of revealing subdomains," *IEEE Transactions on Software Engineering*, vol. SE-6, no. 3, pp. 236–246, May 1980.

[6] E. W. Dijkstra, "Notes on Structured Programming," Apr. 1970, circulated privately. [Online]. Available: http://www.cs.utexas.edu/users/EWD/ewd02xx/EWD249.PDF

[7] D. Hamlet and R. Taylor, "Partition testing does not inspire confidence (program testing)," *Transactions on Software Engineering*, vol. 16, pp. 1402–1411, 1990.

[8] C. Boyapati, S. Khurshid, and D. Marinov, "Korat: Automated testing based on java predicates," in *Proceedings of the 2002 ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA '02, 2002, pp. 123–133.

[9] G. Fraser and A. Arcuri, "Evosuite: Automatic test suite generation for object-oriented software," in *ESEC/FSE '11*. New York, NY, USA: ACM, 2011, pp. 416–419.

[10] C. Cadar, D. Dunbar, and D. R. Engler, "Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs," in *8th USENIX Symposium on Operating Systems Design and Implementation*, ser. OSDI'08, 2008, pp. 209–224.

[11] P. Godefroid, N. Klarlund, and K. Sen, "Dart: Directed automated random testing," in *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '05, 2005, pp. 213–223.

[12] L. J. Morell, "A theory of fault-based testing," *IEEE Transactions on Software Engineering*, vol. 16, no. 8, pp. 844–857, Aug. 1990.

[13] A. Goldberg, T. C. Wang, and D. Zimmerman, "Applications of feasible path analysis to program testing," in *Proceedings of the 1994 ACM SIGSOFT international symposium on Software testing and analysis*, ser. ISSTA '94, 1994, pp. 80–94.

[14] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Transactions on Software Engineering*, vol. 37, no. 5, pp. 649–678, Sept 2011.

[15] M. Böhme, B. C. d. S. Oliveira, and A. Roychoudhury, "Partition-based regression verification," in *35th International Conference on Software Engineering*, ser. ICSE'13, 2013, pp. 302–311.

[16] J. W. Duran and S. C. Ntafos, "An evaluation of random testing," *IEEE Transactions on Software Engineering*, vol. 10, no. 4, pp. 438–444, Jul. 1984.

[17] W. Gutjahr, "Partition testing vs. random testing: the influence of uncertainty," *Transactions on Software Engineering*, vol. 25, no. 5, pp. 661–674, Sep 1999.

[18] T. Y. Chen and Y.-T. Yu, "On the expected number of failures detected by subdomain testing and random testing." *IEEE Transactions on Software Engineering*, vol. 22, no. 2, pp. 109–119, 1996.

[19] A. Arcuri, M. Iqbal, and L. Briand, "Random testing: Theoretical results and practical implications," *IEEE Transactions on Software Engineering*, vol. 38, no. 2, pp. 258–277, March 2012.

[20] E. J. Weyuker, "On Testing Non-Testable Programs," *The Computer Journal*, vol. 25, no. 4, pp. 465–470, Nov. 1982.

[21] B. Korel, "Automated software test data generation," *IEEE Transactions on Software Engineering*, vol. 16, no. 8, pp. 870–879, 1990.

[22] C. Cadar, V. Ganesh, P. M. Pawlowski, D. L. Dill, and D. R. Engler, "Exe: Automatically generating inputs of death," *ACM Transactions on Information and System Security*, vol. 12, no. 2, 2008.

[23] N. Tracey, J. Clark, and K. Mander, "Automated program flaw finding using simulated annealing," in *Proceedings of the 1998 ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA '98, 1998, pp. 73–81.

[24] D. Rosenblum, "A practical approach to programming with assertions," *IEEE Transactions on Software Engineering*, vol. 21, no. 1, pp. 19–31, Jan 1995.

[25] B. Korel and A. M. Al-Yami, "Assertion-oriented automated test data generation," in *Proceedings of the 18th International Conference on Software Engineering*, ser. ICSE '96, 1996, pp. 71–80.

[26] P. Godefroid, A. V. Nori, S. K. Rajamani, and S. D. Tetali, "Compositional may-must program analysis: Unleashing the power of alternation," in *Proceedings of the 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '10. New York, NY, USA: ACM, 2010, pp. 43–56.

[27] P. Thévenod-Fosse and H. Waeselynck, "An investigation of statistical software testing," *Software Testing, Verification & Reliability*, vol. 1, no. 2, pp. 5–25, 1991.

[28] R. Majumdar and R.-G. Xu, "Directed test generation using symbolic grammars," in *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '07, 2007, pp. 134–143.

[29] B. Rosén, "Asymptotic normality in a coupon collector's problem," *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, vol. 13, no. 3-4, pp. 256–279, 1969.

[30] Q. Yang, J. J. Li, and D. Weiss, "A survey of coverage based testing tools," in *International Workshop on Automation of Software Test*, 2006, pp. 99–103.

[31] T. Williams, M. Mercer, J. Mucha, and R. Kapur, "Code coverage, what does it mean in terms of quality?" in *Proceedings of the Reliability and Maintainability Symposium*, 2001, pp. 420–424.

[32] J. Geldenhuys, M. B. Dwyer, and W. Visser, "Probabilistic symbolic execution," in *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, ser. ISSTA 2012, 2012, pp. 166–176.

[33] M. Staats, G. Gay, M. W. Whalen, and M. P. E. Heimdahl, "On the danger of coverage directed test case generation," in *15th International Conference on Fundamental Approaches to Software Engineering*, ser. FASE'12, 2012, pp. 409–424.

[34] R. Sharma, M. Gligoric, A. Arcuri, G. Fraser, and D. Marinov, "Testing container classes: Random or systematic?" in *Proceedings of the 14th International Conference on Fundamental Approaches to Software Engineering*, ser. FASE'11, 2011, pp. 262–277.

[35] N. E. Beckman, A. V. Nori, S. K. Rajamani, and R. J. Simmons, "Proofs from tests," in *Proceedings of the 2008 International Symposium on Software Testing and Analysis*, ser. ISSTA '08, 2008, pp. 3–14.

[36] M. Miraz, P. L. Lanzi, and L. Baresi, "Improving evolutionary testing by means of efficiency enhancement techniques," in *Proceedings of the IEEE Congress on Evolutionary Computation*, ser. CEC '10, 2010, pp. 1–8.

[37] P. D. Marinescu and C. Cadar, "Make test-zesti: A symbolic execution solution for improving regression testing," in *Proceedings of the 34th International Conference on Software Engineering*, ser. ICSE '12, 2012, pp. 716–726.

Marcel Böhme is currently Postdoctoral Fellow with Prof. Andreas Zeller at Saarland University. Marcel completed his PhD at the School of Computing, National University of Singapore advised by Prof. Abhik Roychoudhury and received his Dipl.-Inf. (cf. M.Sc.) from Technische Universität Dresden, Germany in 2014 and 2009, respectively. His research is on testing, debugging, and repair of evolving programs – where he seeks to understand and elucidate intrinsic properties such as the efficiency of automated testing, the complexity of realistic software errors, and the interaction of faults and changes, amongst others. Generally, his work is driven towards establishing and extending the formal foundations of Software Engineering.

Soumya Paul is currently Postdoctoral Fellow with Prof. P.S. Thiagarajan at the School of Computing, National University of Singapore. Soumya received his PhD from the Institute of Mathematical Sciences, Chennai, India under the guidance of Prof. R. Ramanujam and his MS in Theoretical Computer Science also from the same institution prior to that. His research interest is broadly the foundations of Computer Science with focus on Logic, Automata, Games and Formal Verification. He has also worked on the modeling, approximation and model-checking of Hybrid Systems.