Developing and Evaluating Objective Termination Criteria for Random Testing

PORFIRIO TRAMONTANA, DOMENICO AMALFITANO, and NICOLA AMATUCCI,
Department of Electrical Engineering and Information Technologies, University of Naples Federico II, Italy
ATIF MEMON, Department of Computer Science, University of Maryland, USA
ANNA RITA FASOLINO, Department of Electrical Engineering and Information Technologies, University of Naples Federico II, Italy

Random testing is a software testing technique through which programs are tested by generating and executing random inputs. Because of its unstructured nature, it is difficult to determine when to stop a random testing process. Faults may be missed if the process is stopped prematurely, and resources may be wasted if the process is run too long. In this article, we propose two promising termination criteria, "All Equivalent" (AEQ) and "All Included in One" (AIO), applicable to random testing. These criteria stop random testing once the process has reached a code-coverage-based saturation point after which additional testing effort is unlikely to provide additional effectiveness. We model and implement them in the context of a general random testing process composed of independent random testing sessions. Thirty-six experiments involving GUI testing and unit testing of Java applications have demonstrated that the AEQ criteria is generally able to stop the process when a code coverage equal or very near to the saturation level is reached, while AIO is able to stop the process earlier in cases it reaches the saturation level of coverage. In addition, the performance of the two criteria has been compared against other termination criteria adopted in the literature.

CCS Concepts: • **General and reference → Verification**; • **Software and its engineering → Software testing and debugging**;

Additional Key Words and Phrases: Random testing, saturation effect, code coverage

ACM Reference format:
Porfirio Tramontana, Domenico Amalfitano, Nicola Amatucci, Atif Memon, and Anna Rita Fasolino. 2019. Developing and Evaluating Objective Termination Criteria for Random Testing. *ACM Trans. Softw. Eng. Methodol.* 28, 3, Article 17 (July 2019), 52 pages.
https://doi.org/10.1145/3339836

1 INTRODUCTION

According to the ISO 29119 Standard on Software Testing [1], random testing is a testing technique where input values are randomly selected from a set of possible values, using a specific input distribution for the generation of random input values. Compared to other testing techniques,

Authors' addresses: P. Tramontana, D. Amalfitano, N. Amatucci, and A. R. Fasolino, Department of Electrical Engineering and Information Technologies, University of Naples Federico II, Naples, Italy; emails: {porfirio.tramontana, domenico.amalfitano, nicola.amatucci, fasolino}@unina.it; A. Memon, Department of Computer Science, University of Maryland, Via Claudio, 21, College Park, Maryland, USA; email: atif@cs.umd.edu.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2019 Association for Computing Machinery.
1049-331X/2019/07-ART17 $15.00
https://doi.org/10.1145/3339836

random testing has traditionally enjoyed popularity among practitioners in industry because of its simplicity of use, fully automatic nature, and quick deployment with little personnel training and upfront investment.

Over the years, we have seen several flavors of random testing, including "classical" random testing in which test inputs are generated randomly with a uniform probability [13] to more recent "smarter" approaches such as Adaptive Random Testing [27] and Guided Random Testing [71] that actively steer the random generation process toward a more desirable sampling of the input space. Multiple studies have empirically demonstrated the usefulness of random testing in many different contexts [30, 33, 34, 56, 57, 68, 85, 91, 100, 107], and have compared random testing with other approaches such as partition testing [12] and combinatorial testing [9]. Because of its popularity and demonstrated effectiveness, there is little doubt that random testing is here to stay.

Despite the deployment of random testing techniques, there are still open issues regarding their general use, key among them is the lack of a well-established stopping criterion [11, 31, 32, 40, 47, 61, 92]. Stop random testing prematurely and one risks missing faults; run it for too long and one wastes resources. Both cases are non-optimal; today, a tester has to guess at when to stop random testing.

Researchers have used a number of heuristics as a guide to stop the random testing process, the most popular among them stopping after testing resources, including time, are exhausted [11, 31, 32, 40]. A few researchers have proposed using heuristics such as stopping after reaching a fixed coverage value [47, 61, 92] or a fixed number of found faults [61, 92], or stopping after a fixed time interval since the last increase in code coverage or faults detected [47].

In an earlier report of this work [3], we addressed the stopping criterion problem of random testing by developing two complementary criteria based on the *saturation effect* of testing. The saturation effect is a phenomenon where the rate of convergence of a test suite toward a specific test adequacy criterion decreases as the amount of test execution increases [69]. According to the first proposed termination criterion, testing is stopped when all the executed testing sessions have covered identical sets of testing targets, whereas, according to the second criterion, it is stopped when there is a testing session that has covered a set of testing targets that is the union of the sets of testing targets covered by all the performed testing sessions. We applied these termination criteria to testing processes involving the execution of 12 independent testing sessions composed of randomly selected GUI events sent to an Android app. A preliminary experiment carried out on 18 Android apps showed that the adoption of these termination criteria provided a cost-effective stop point for almost all the considered applications.

We extend our previous work by further investigating the theoretical soundness of the approach, its usefulness in different testing contexts, and the comparing to other existing termination criteria.

We provide the following main contributions:

- a theoretical background of the existence of the saturation effect in random testing processes, the relationship between the saturation effect, and the predictability property of random testing and the consequent theoretical soundness of termination criteria based on the saturation effect;
- the definition of two objective termination criteria based on the identification of the saturation effect in the context of a generic *Multiple Random Testing Process* based on the execution of independent random testing sessions;
- an experimental evaluation of the proposed criteria in two different contexts involving random GUI testing of Java applications and random unit testing of Java applications and libraries. The main objectives of these experiments are as follows: (1) the evaluation of the capability of the proposed termination criteria of identifying the saturation effect when different numbers of testing sessions are executed; (2) the cost-effectiveness of the processes

based on the proposed termination criteria when different number of testing sessions are executed; and (3) a comparison between the performance of testing processes adopting the proposed stopping criteria against the ones of processes adopting other termination criteria proposed in literature.

The rest of the article is organized as follows: Section 2 reports related work on random testing focusing on random testing termination criteria; in Section 3 a theoretical background on random testing processes is presented by focusing on the properties of saturation and predictability and by investigating the relationship between them; in Section 4 the termination criteria are formally defined in the context of a generic random testing process based on multiple independent testing sessions. The results of the experiments involving system testing of Java GUI-based applications are reported and discussed in Section 5 while Section 6 reports the results obtained by another experiment involving unit testing of Java applications and libraries. The main threats to the validity of both the experiments are reported in Section 7. Finally, Section 8 contains concluding remarks and ideas for future work.

2 RELATED WORK

This section presents related work on random testing techniques and reports the metrics that are generally used to evaluate their performance. Moreover, a review of the termination criteria most commonly adopted in random testing processes is presented.

2.1 Random Testing Techniques

Mariani et al. in a recent survey on Automatic Black Box techniques [79] classify black box random testing techniques in four categories: Pure Random testing techniques, Adaptive Random techniques, Guided Random techniques, and Fuzz techniques.

In Pure Random testing techniques, test inputs are randomly generated with a uniform probability. Examples of Pure Random testing tools are monkey tools [13], which are able to solicit the application under test by sending it random input sequences that in some ways can be imagined as replications of the behavior of a monkey that "uses" the application. Monkey tools are relatively simple to be implemented and are very popular in many contexts. For example, the *UI/Application Exerciser Monkey* included in the Android Development Toolkit (ADT) is widely adopted in Android applications testing. It is a command line executable testing tool that implements a technique that fires events belonging to different classes of user and system events.

Adaptive Random Testing (ART) techniques first introduced by Chen et al. in 2001 [29] improve the capability of Pure Random testing techniques in fault finding by taking into account the diversity between the generated test cases. Many contributions in the literature have been proposed in this field to improve the effectiveness and the efficiency of ART techniques [67, 68]. Several studies demonstrated that Adaptive Random testing techniques are more effective than Pure Random testing techniques [27, 28]. However, Arcuri and Briand highlighted that the empirical studies in the literature may be sometimes misleading about the real efficiency of ART techniques, because the applications involved in the experiments are usually more faulty than real systems [8].

Guided Random testing techniques take into account pieces of information about the software under test extracted via static or dynamic analysis. Examples of Guided Random testing techniques are the ones proposed by Wei et al. [105], Groce et al. [50], Hofer et al. [59], Godefroid et al. [46], Bauersfeld et al. [14], and Pacheco et al. [89, 90]. In particular, many contributions in the last years regard the field of mobile applications testing; some recent tools implementing Guided Random testing techniques on mobile applications are Android Ripper [4], Dynodroid [72], PUMA [58], and Vanarsena [96].

Finally, Fuzz testing techniques are based on the random mutation of valid inputs [86]. An example of fuzz testing technique is the one proposed by Godefroid et al. [45], that exploits symbolic execution and constraint solving to generate random input values for unit test cases. More recent examples of works involving fuzz testing techniques are the ones of Tsankov et al. [103] in the context of security protocols testing and of Martignoni et al. [80] in the context of virtual machines testing.

An evolution of random testing techniques is represented by Search Based testing techniques founded on genetic algorithms aiming at the optimization of pure random testing techniques [51, 52, 73, 95]. In particular, the Evosuite tool [38, 40] is able to generate unit test cases by optimizing their ability in automatically killing generated mutant versions of the program under test by means of a genetic algorithm.

The performance of random testing techniques has been object of many studies in the literature, both by a theoretical and by an empirical point of view, to assess their testing adequacy, their efficiency, their predictability and to compare them with respect to the ones of other testing activities.

As regards testing adequacy of random testing techniques, it is usually measured in terms of faults found or in terms of covered code. Several empirical studies in the literature demonstrated the existence of a positive correlation between code coverage and fault detection capability [16, 20, 24, 37, 61, 70, 84, 93, 99]. Moreover, many authors found correlation between code coverage and software reliability [25, 43, 48, 53, 62–64, 74, 75, 94, 104]. On the contrary, Wei et al. [106] demonstrated that code coverage is not always sufficiently correlated with fault detection capability, in the context of Eiffel programs and design by contract development.

Some works try, instead, to propose theoretical models for modeling random testing techniques performance. Arcuri et al. [9, 12] investigated the capability of random testing techniques in covering testing targets. They modeled the problem of target coverage as a Coupon Collector problem [83] in the hypothesis that each test case covers exactly one testing target (they applied this model to crash testing in which each test case cannot found more than one application crash). They found that a pure mathematical analysis for such techniques is unfeasible due to the complex interactions among the sets of possible random inputs and to the variable length of test sequences. However, they demonstrated the existence of a lower bound for the expected number of random test cases needed to cover a set of test targets, and showed that the best performance is achieved when all the test targets have the same probability of being covered. Moreover, they demonstrated how the random testing techniques can scale up better than Partition Testing techniques. These theoretical results have been confirmed by the work of Boehme and Paul [15]. They have also discussed in [19] theoretical aspects of the efficiency of random testing in the hypothesis that all the testing targets have the same probability to be executed by each test case. Their results cannot be generalized to the context of testing processes where the values of probability of each testing target to be covered by a test case are unknown and different between them, and each test case is able to cover zero or more testing targets (e.g., when testing targets are represented by LOCs or branches).

2.2 Random Testing Termination Criteria

Effectiveness, efficiency, and costs of random testing processes are influenced by the termination criteria used to stop the testing process. The termination criteria proposed in the literature can be grouped in four distinct categories:

- Termination criteria related to the reaching of an upper bound in testing time or resources;
- Termination criteria related to the achievement of a fixed testing adequacy level;
- Termination criteria based on predictions given by estimation models;

- Termination criteria related to the evaluation of predictability and saturation properties of the testing process.

Fixing an upper bound related to the cost of the testing process, such as the number of executed test cases, the total number of generated input sequences or the testing time, is a commonly adopted termination criterion for a random testing process. This kind of practical criteria is often used when the process has constraints related to its cost. These criteria generally do not give any guarantee about the testing adequacy of the generated test suite. The testing process can be terminated too early, probably resulting in a scarce effectiveness, or too late, wasting time and resources in the redundant execution of identical test cases. These criteria are largely used in works in the literature and in industrial practice due to their simplicity. For example, in References [36, 72, 82] the testing process termination criteria is related to the execution of a fixed number of events. In References [90, 100], instead, the process is terminated after the execution of a fixed maximum number of function calls. Moreover, in search-based testing techniques the process terminates after the execution of a maximum number of iterations (e.g., References [2, 44, 65]). Arcuri and Fraser [11, 40] have executed search-based unit testing techniques on Java classes belonging to applications and libraries with the aim to optimize branch coverage. They fixed a maximum testing time value that is 10 minutes for each tested class in Reference [40] and, successively, they reduced this time to 2 minutes per class (a reasonable time according to some testers interviewed by the authors [11]). Ciupa et al. [31, 32], instead, performed experiments involving classes belonging to Eiffel libraries with the aim to obtain a predictable test adequacy. They have found that the results become almost independent on randomness with testing sessions lasting 90 minutes for each class.

Termination criteria may also be related to the reaching of a predefined level of test adequacy (for example, the reaching of a fixed level of coverage, the discovery of the injected faults, the occurrence of a fixed number of distinct failures). Due to the possible presence of infeasible testing targets, these termination criteria are rarely adopted (e.g., in Reference [92] and in Reference [61]). In some cases, the time needed to find the first fault is fixed as a termination criterion able to evaluate the efficiency of the testing technique (e.g., in References [27, 29, 67, 68]).

As regards the termination criteria based on predictions derived by estimation techniques, they rely on the adoption of statistical models. These models are mainly related to the growth of a testing adequacy measure with respect to the increase of the number of the executed test cases. For example, Arcuri et al. [12] and Howden [60] model the coverage growth according to a binomial distribution. Other works in the literature, instead, model it by means of Poisson processes [54, 55, 76, 99], logarithmic series distributions [42, 49] or Bayesian predictive distributions [66, 98].

Several other works in literature attempt to model the operational usage of the system under test and to derive the corresponding trend of its reliability to propose optimal testing termination criteria. In most cases the operational usage is modeled with Markov chains (e.g., in References [23, 26, 108]) or structural models [102]. Unfortunately, these models are generally based on hypotheses related to the statistical independence of the coverage of the testing targets or to the independence of the quantity of time occurring between two failures. These hypotheses can not be verified when code coverage is used to measure test adequacy, because the coverage of a single line of code is usually strongly related to the coverage of other lines of code (e.g., to the ones belonging to the same function or block). In general, termination criteria based on statistical models have a strong theoretical relevance, but they are not often applied in the practice for the difficulties in obtaining the required statistical models.

The trend of the reliability of applications that are randomly tested with the increase of the number of the executed test cases has been taken into account, too, to formulate termination criteria. Some criteria consist in stopping the testing process after that a fixed number of consecutive

tests have been executed without any increase in test adequacy [47]. For instance, Bueno and Jino [17] propose to stop an iterative process of testing when the two following assertions become true:

$$c[i] - c[i-1] \leq \delta \; \forall i, j : j \leq i \leq j + NL, \tag{1}$$

$$c[j + NL] - c[j] \leq \Delta, \tag{2}$$

where $c[i]$ is the value of a monotonically growing testing adequacy metric (e.g., the code coverage) after i iterations of the testing process, NL is the size of the subset of process iterations that is considered by the termination criterion, δ is the maximum difference in adequacy between two consecutive iterations that is considered negligible and Δ is the maximum difference in adequacy between NL consecutive iterations that is considered negligible.

If $\delta = \Delta = 0$, then this termination criterion stops the testing process after NL consecutive iterations of the testing process in which no increases in testing adequacy have been observed.

Termination criteria based on this metric are usually adopted in the context of search-based testing techniques [17, 18, 77]. For example, Mansour et al. [77] have stopped their testing processes after 50 consecutive iterations without any improvement of the fitness metric ($NL = 50$ and $\delta = \Delta = 0$).

Oriol [87] and Furia et al. [42] searched for the empirical laws able to relate the number of executed random test cases and the number of faults and failures found in Eiffel and Java applications. Their results are promising in terms of measured correlations but can not be easily generalized to code coverage problems.

Finally, some works in the literature explicitly noted that the trend of the test adequacy of random testing techniques tends to be predictable by increasing the number of executed tests. For example, Ciupa et al. [31, 32] found that the variance of the number of faults detected by different random testing sessions tends to decrease as the number of executed test cases increases; they concluded that the fault detection capability of random testing techniques is predictable if a sufficient number of test cases are executed. More specifically, Sherman et al. [101] exploited the existence of a saturation effect occurring when the increase in coverage over a window of test runs (of a given size) is less than a fixed threshold; they used this effect to define new adequacy criteria in concurrency testing. They argued the existence of a saturation point at which the maximum possible testing adequacy has been reached and further tests are not able to improve the testing adequacy. The termination criteria exploited in this article and proposed in our previous work [3] are designed to automatically identify the reaching of a saturation point. These criteria will be defined and explained in the next section.

3 MODELING THE RANDOM TESTING PROCESS, ITS SATURATION EFFECT, AND EXAMINING ITS PREDICTABILITY

We now present a general treatment of random testing, which we then enhance with a coverage-based termination criterion, setting the stage for defining the saturation effect, which we later show to be predictable with respect to random testing.

3.1 A Generic Treatment of Random Testing

For a given *A*pplication *U*nder *T*est (*AUT*), we define *RTT* as a random testing technique that can select random test cases $tc \subseteq TC$, where TC is the set—possibly infinite—of test cases that can be executed on the AUT. The pseudocode describing an iterative random testing process is shown in Algorithm 1.

At each iteration of this process, a test case *tc* is randomly selected according to the random testing technique *RTT*, and executed on the *AUT*. Note that we have used a very generic termination criterion, which we evolve next.

ALGORITHM 1: Generic Random Testing Process

Procedure *Random Testing Process (RTT)*
  repeat
    $tc \leftarrow SelectTestCase(RTT)$;
    $Execute(tc)$;
  until $TerminationCriterion$;

3.2 Random Testing Process with a Coverage-based Termination Criterion

Let us assume that our AUT contains a finite set—of size n—of unique "testing targets," $TT = \{t_1, \ldots, t_n\}$ that may be "covered" by test cases. Examples of finite testing targets are program statements, branches, methods, classes, and so on. Hence, each of our test cases c from Algorithm 1 may cover one or more elements of TT.

The cumulative coverage achieved by the execution of a set $s = \{c_1, \ldots, c_k\}$ composed of k test cases each one covering a set of testing targets C_i, can be evaluated as the ratio between the number of testing targets belonging to at least one of the coverage sets C_i and the total number of existing testing targets $|TT|$:

$$Coverage = \frac{|\bigcup_{i=1}^{k} C_i|}{|TT|}. \tag{3}$$

Our original pseudocode is now evolved to terminate once all the testing targets have been covered. The new code is shown in Algorithm 2.

ALGORITHM 2: Random Testing Process with Coverage-based Termination

Procedure *RandomTestingProcess(RTT)*
  repeat
    $tc \leftarrow SelectTestCase(RTT)$;
    $c \leftarrow Execute(tc)$;
    $Coverage \leftarrow UpdateCoverage(c)$;
  until $Coverage == 1$;

The test case tc is randomly selected according to the random testing technique RTT, is executed, and the corresponding coverage set c is returned, then the cumulative coverage achieved by the testing process is updated. The testing process terminates when all the testing targets have been covered at least once (i.e., $Coverage = 1$).

3.3 Saturation of the Random Testing Process

As was done by Lyu et al. [69], we define the *Saturation Level* as the maximum level of coverage achievable by executing the random testing process adopting the random testing technique RTT with respect to the application AUT.

Consider the following two conditions:

- $C1$ Each testing target is covered by at least one test case $tc \in TC$ that can be selected by RTT.
- $C2$ RTT independently selects test cases, i.e., the previously selected test cases do not influence the selection of the next test case.

If $C1$ holds, then the Saturation Level of the testing process reported in Algorithm 2 is equal to 1, because all the testing targets can be covered by at least a test case that can be selected by RTT.

With respect to the given set of testing targets $TT = \{t_1, \ldots t_n\}$, we define $P = \{p_1, \ldots, p_n\}$ as the set of corresponding values representing the probability to reach each testing target with a test case selected by *RTT*. Because $C1$ holds, it follows that all the testing targets are feasible, i.e., can be covered. That is,

$$0 < p_i \leq 1, \forall p_i \in P. \tag{4}$$

Let $p_{i,k}$ represent the probability of covering—at least once—the testing target t_i after the execution of k iterations of the testing process reported in Algorithm 2. According to $C2$, all the test cases are independently selected and consequently $p_{i,k}$ is given by:

$$p_{i,k} = 1 - (1 - p_i)^k. \tag{5}$$

In fact, $(1 - p_i)$ is the probability of not covering t_i with a single test case, while $(1 - p_i)^k$ is the probability of not covering p_i with k independent and randomly selected test cases. Consequently, $1 - (1 - p_i)^k$ is the probability of covering t_i at least once after the execution of k test cases.

For each testing target t_i, the probability of covering it asymptotically tends to 1 for growing values of k:

$$\lim_{k \to \infty} p_{i,k} = 1, \forall t_i \in TT. \tag{6}$$

Hence, the execution of the random testing process shown in Algorithm 2 asymptotically reaches the Saturation Level, i.e., the coverage of all feasible testing targets.

3.4 Predictability of Two Runs of the Random Testing Process

Condition $C1$ often does not hold in practical testing situations due to two reasons. First, not all the testing targets are feasible; second, the set of targets that can be covered by test cases that can be selected by *RTT* is unknown. In these cases, termination criteria based either on 100% coverage of all the testing targets or of all the feasible testing targets are not useful, since the tester does not know if this termination condition could ever be achieved.

We want to investigate a more practical alternative, i.e., to propose termination criteria that are able to stop the process at the Saturation Level of coverage even if $C1$ does not hold and the Saturation Level is unknown at the start of the testing process.

Let us consider:

- two testing sessions V_1 and V_2 corresponding to two independent executions of a sequence of k iterations of the testing process reported in Algorithm 2 on an AUT having n testing targets;
- the coverage vectors C_1 and C_2 corresponding to the testing sessions V_1 and V_2, each composed of n binary values. The generic values $C_i[j]$ is 1 if the testing target j has been covered by at least a test case of V_i, 0 elsewhere.

The difference in coverage $D(V_1, V_2)$ between the two testing sessions V_1 and V_2 can be defined as:

$$D(V_1, V_2) = \sum_{i=1}^{n} |C_1[i] - C_2[i]|. \tag{7}$$

This difference corresponds to the sum of the number of testing targets that are covered by the execution of V_1 but are not covered by the execution of V_2 and of the number of testing targets that are covered by the execution of V_2 but are not covered by the execution of V_1.

It has been demonstrated in earlier reported work [12] that the average value of D after the execution of k test cases is as follows:

$$E[D] = \sum_{i=1}^{n} 2 \cdot (1 - p_i)^k \cdot (1 - (1 - p_i)^k), \tag{8}$$

where p_i is the probability to cover the testing target t_i with a test case. $E[D]$ is upper bounded by $n/2$ [12] and asymptotically tends to zero for growing values of k:

$$\lim_{k \to \infty} E[D] = 0. \tag{9}$$

Arcuri et al. [12] have defined the *predictability* property of a random testing process in terms of the average difference in coverage D between two executions of the process, each composed of k test cases. They stated that the testing process is predictable after the execution of k test cases if the average coverage difference between two testing sessions is zero:

$$E[D] = 0. \tag{10}$$

For the purpose of our work, this means that the testing process is predictable if the cumulative coverage achieved after k test cases is independent on randomness. Property 9 guarantees that Property 10 is verified for $k \to \infty$ and, in the same conditions, the achieved coverage tends to be equal to the Saturation Level for Property 6. Property 9 indicates that there is an asymptotic equivalence between the predictability and the reaching of the saturation level. However, the reaching of the same level of coverage for different testing sessions generally does not imply that such a level represents the saturation level for the considered testing process.

To further investigate the relationship between the predictability condition and the reaching of the saturation level, we have studied the trend of *E[D]* and of the cumulative coverage for growing values of k with respect to three example probability distributions of testing targets for an abstract application having 100 testing targets. We summarize the results of our investigation in Figure 1, which shows on the left side the trends of $E[D]$ with respect to k for each of the three proposed distributions (evaluated with Formula 8) and on the right side the corresponding average values of coverage, evaluated by measuring the average cumulative coverage achieved by 1,000 pairs of simulated testing sessions.

PD_1 is a uniform probability distribution, where the probability of each testing target to be executed by a test case is as follows:

$$PD_1 : p_i = 1/100, \forall i \in \{1..100\}. \tag{11}$$

In the second probability distribution PD_2 it is possible to distinguish between 90 "easy" testing targets and 10 "difficult" targets. The "easy" targets have all the same probability to be executed (equal to 0.11111), whereas the "difficult" targets have all a probability of 10^{-6} to be executed by a single test case,

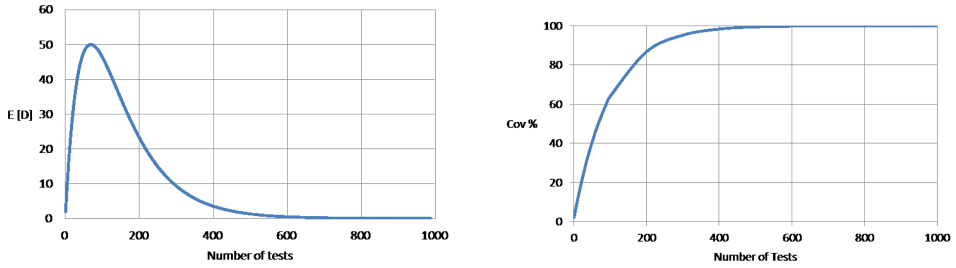
$$PD_2 : p_i = \begin{cases} 0.11111, \forall i \in \{1..90\} \\ 10^{-6}, \forall i \in \{91..100\} \end{cases}. \tag{12}$$

In PD_3 the distribution of probability of the testing targets is a geometric distribution, in the sense that:

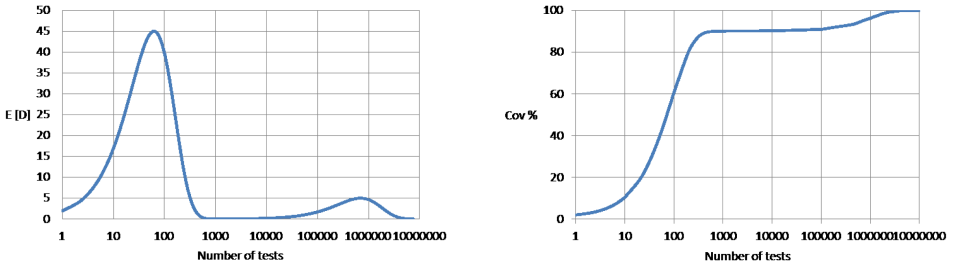
$$PD_3 : \begin{cases} p_1 = 0.099 \\ p_i = 1.11 * p_{i-1}, \forall i \in \{2..100\} \end{cases}. \tag{13}$$

We can verify that the probability of the most difficult target is $p_{100} \cong 3.23 \times 10^{-6}$.

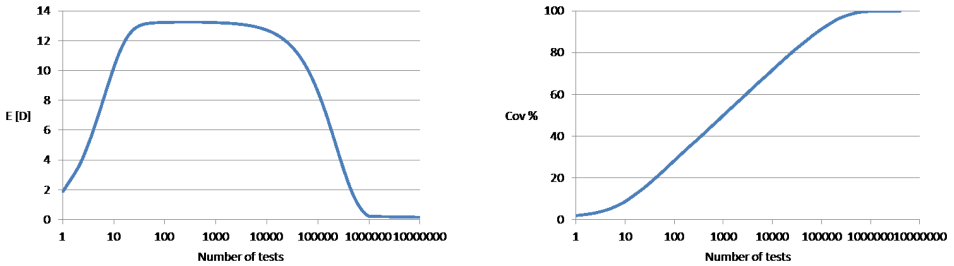
We can observe that all the three trends of *E[D]* are not monotonous for growing values of k. In particular, the trend of *E[D]* with the first distribution PD_1 starts with growing values of *E[D]*



(a) Average Value of D and Coverage with Uniform probability distribution PD_1 of the testing targets



(b) Average Value of D and Coverage with Two Levels probability distribution PD_2 of the testing targets



(c) Average Value of D and Coverage with Geometric probability distribution PD_3 of the testing targets and different numbers of testing sessions

Fig. 1. Average values of D and coverage with different probability distributions of the testing targets.

until the maximum possible value ($n/2 = 50$) is reached, then the values rapidly go toward zero while the coverage goes toward the Saturation Level (100/100).

The *E[D]* values obtained with the second distribution PD_2, instead, have initially a trend similar to the first distribution and go toward zero after about 1,000 test cases. At this point, we can observe on the right side of the second subfigure of Figure 1 that the average value of coverage is about 90/100, i.e., only all the "easy" testing targets have been covered. Furtherly, *E[D]* grows, reaches a local maximum and finally returns toward zero after about 10^7 test cases, when the coverage goes toward the Saturation Level of 100/100. Finally, the *E[D]* values measured for the third distribution PD_3 reach a maximum value (that is less than 14) and slowly tend to zero without any local minimum value. Values near zero are reached after about 10^7 test cases and, at the same time, the coverage reaches values near the saturation level.

Table 1. Coverage Achieved by 1,000 Simulations of the Random Testing Process Terminating at the Reaching of a Null Difference between the Two Testing Sessions

	Levels of Coverage Achieved When $D(V_1, V_2) = 0$
PD_1	100/100 (998 times), 99/100 (2 times)
PD_2	100/100 (11 times), 90/100 (986 times), 89/100 (2 times), 91/100 (1 time)
PD_3	100/100 (936 times), 99/100 (47 times), less than 99/100 (17 times)

In all the three cases, it is possible to observe the phenomenon called *saturation effect* [69] for which the rate of increment of the cumulative coverage of the random testing process decreases as the amount of executed test cases increases.

On the basis of these examples, we can observe that the predictability condition 10 does not imply the reaching of the Saturation Level. However, it may be interesting to assess how frequently the reaching of a null value of the distance $D(V_1, V_2)$ occurs together with the reaching of the saturation level of coverage.

The levels of coverage measured when the distance $D(V_1, V_2)$ becomes zero for each of the 1,000 pairs of simulated testing sessions are reported in Table 1. It shows that for the first distribution PD_1 the saturation level of coverage is reached almost always (998 times of 1,000), whereas with PD_2, most of the processes terminate when only the 90 easy testing targets have been covered (986 times of 1,000) and, with PD_3, 936 times of 1,000 the saturation level of coverage is reached when the difference becomes zero. However, lower levels of coverage or (in the second case) higher levels of coverage sometime can be measured, due to the intrinsic randomness of the testing process.

More in general, we can conclude that in the first and in the third case, there is a high probability that the reaching of a null difference $D(V_1, V_2)$ coincides with the reaching of the saturation level of coverage, while in the second case, there is a high probability to reach a level of coverage consisting in the coverage of all the "easy" testing targets.

On the basis of this data, we can assume that the termination criterion stopping the testing process when the difference $D(V_1, V_2)$ between two testing sessions V_1 and V_2 is zero may represent a cost-effective solution to the termination problem, at least with respect to the three considered cases. Indeed, in the first and in the third case, further test cases that may be executed generally would not provide any increase in coverage, while in the second case they would have provided an increase only after a very large number of executed test cases.

Nevertheless, if we consider the wider context of real software applications, then nothing could be said about the relation between the reaching of a null value of D and the reaching of the saturation level, since the probability distribution of testing targets in real software applications is usually not known. However, the results we observed in our previous study encourage us to further investigate how often the two properties are simultaneously satisfied in random testing of real applications.

To this aim, in the following section, we present two termination criteria based on the comparison between the coverage achieved by two or more independent random testing sessions. The capability to find the saturation level of coverage and the cost-effectiveness of the testing processes based on these criteria will be empirically assessed with respect to different random testing techniques and tools and different applications under test.

4 THE CONSIDERED TERMINATION CRITERIA

We now present two termination criteria that generalize the ones previously proposed in Reference [3]. They are applicable to iterative random testing processes consisting of the execution of

multiple testing sessions. Both these termination criteria are based on the comparison between the cumulative coverage provided by the executed testing sessions.

4.1 A Generic Multiple Random Testing Process

We first present a general treatment of a generic random testing process *TP* based on multiple testing sessions. The process is defined as a 5-tuple (*AUT*, *AUTPr*, *RTT*, *NTS*, *TerC*), where

- *AUT*: the *A*pplication *U*nder *T*est;
- *AUTPr*: a set of *AUT* preconditions that are restored before the execution of each test case;
- *RTT*: the *R*andom *T*esting *T*echnique used to select the test cases to be executed on the *AUT*;
- *NTS*: the *N*umber of *T*esting *S*essions executed on the AUT;
- *TerC*: a *Ter*mination *C*riterion adopted to stop the testing process.

The process is iterative and requires the evaluation of the cumulative coverage achieved by each of the *NTS* testing sessions. The outputs of the process are

- *terP*: an integer variable measuring the number of test cases executed on each session up to the termination of the process;
- *terL*: a real number representing the percentage of testing targets that have been covered by at least a test case of a testing session at the termination of the process.

The process is described by the pseudo code in Algorithm 3. It uses the following variables and methods.

Variables.

- *terCon*: Boolean variable assuming the value of the termination condition corresponding to the selected termination criterion *TerC*;
- *k*: integer variable counting the number of iterations of the testing process;
- *i*: integer variable indicating the number of the testing session (between 1 and *NTS*);
- *tc*: test case randomly selected by *RTT* and executable on the *AUT*;

ALGORITHM 3: Generic Multiple Random Testing Process

```
Procedure MultipleRandomTestingProcess(AUT, AUTPr, RTT, NTS, TerC)
    k ← 0 ;
    terCon ← FALSE ;
    while (terCon = FALSE) do
        k ← k + 1 ;
        foreach i ∈ {1, .., NTS} do
            start(AUT, AUTPr) ;
            tc ← selectTestCase(AUT, RTT) ;
            covTc ← executeTestCase(AUT, tc) ;
            stop(AUT) ;
            SCCS[i] ← updateCov(SCCS[i], covTc) ;
        end
        CCS ← evalCCov(SCCS[]) ;
        terCon ← evalTerCon(TerC, CCS, SCCS[], k) ;
    end
    terP ← k ;
    terL ← evalPercCov(CCS, AUT) ;
```

- *covTc*: set including all the testing targets covered by the test case *tc*;
- *SCCS[]*: array of coverage sets. The *i*th element of this array represents at each iteration of the process the set of testing targets of *AUT* that has been covered by the *i*th testing session after *k* executed test cases;
- *CCS*: set including all the testing targets that have been covered by at least one of the *NTS* sessions after *k* executed test cases.

Methods.

- *start(AUT, AUTPr)*: launches the execution of an instance of the *AUT* with the preconditions *AUTPr*;
- *selectTestCase(AUT, RTT)*: selects a random test case *tc* according to the *RTT* technique;
- *executeTestCase(AUT, tc)*: executes the test case *tc* on the *AUT* and returns the set *covTc* of testing targets covered during the execution of this test case;
- *stop(AUT)*: stops the execution of the *AUT*;
- *updateCov(SCCS[i],covTc)*: updates the cumulative set of testing targets covered by the *i*th testing session after *k* iterations by evaluating the union between the set of testing targets previously covered by the first *k-1* test cases of the session and the set *covTc* of testing targets covered by the last executed test case;
- *evalCCov(SCCS[])*: evaluates the cumulative set of testing targets *CCS* that have been covered by the *NTS* testing sessions;
- *evalTerCon(TerC, CCS, SCCS[], k)*: evaluates the Boolean value of the termination condition *terCon* with respect to the fixed termination criteria *TerC*. Depending on the selected termination criteria, it may take into account the achieved cumulative coverage set *CCS*, the array of coverage sets of all the testing sessions *SCCS[]* and the number of executed test cases *k*;
- *evalPercCov(AUT,CSSC)*: evaluates the percentage of *AUT* testing targets that have been cumulatively covered by the executed testing sessions at the end of the process execution.

The above described process represents an extension of the Process 2 having multiple random testing sessions instead that a single testing session. Also in this process the test cases are independently selected if the hypothesis *C2* is verified and the test cases are executed on the same application starting from the same starting state corresponding to the fixed preconditions *AUTPr*.

4.2 The Proposed Termination Criteria

With respect to this testing process, we define the following two termination conditions:

AEQ (All EQuivalent). The *AEQ* condition is a predicate that is true after *k* iterations of the testing process 3 if each testing session $S_i \in S$ has provided a coverage set $SCCS(S_i)$ that is equal to the union *CCS* of all the coverage sets of all the testing sessions. It is evaluated by means of Equation (14),

$$\begin{aligned} &AEQ(S,k) = TRUE \iff \\ &CCS(S) = SCCS(S_i)\ \forall\ S_i\ \in\ S. \end{aligned} \tag{14}$$

This predicate is true when all the sessions have actually covered the same set of testing targets of the *AUT*. It can be also evaluated by comparing the difference in coverage between each testing session: If there are no differences in coverage between any pair of testing sessions, then the condition is true.

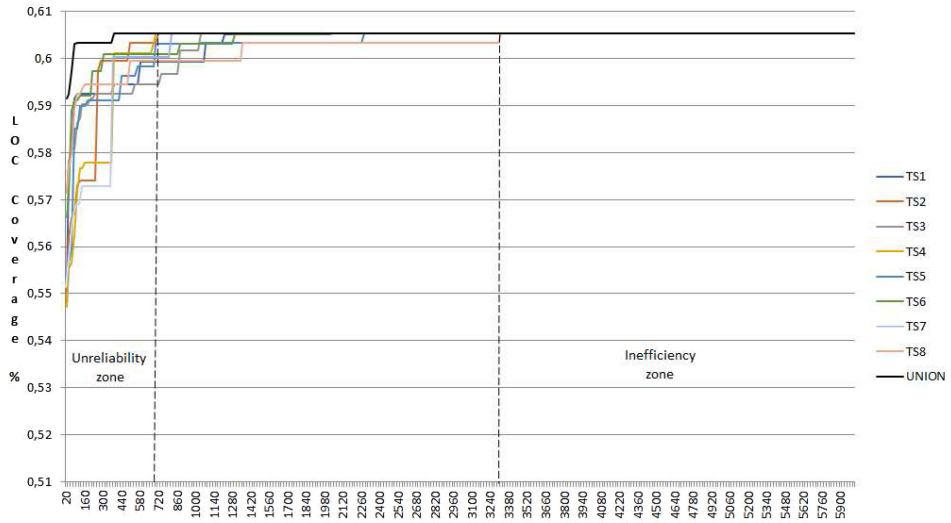


Fig. 2. Code coverage achieved by eight random testing sessions and their cumulative coverage.

All Included in One (AIO). It is a predicate that is true after k iterations of the testing process 3 if it exists a session $S_i \in S$ that has obtained a coverage set that is equal to the union CCS of the coverage sets of all the testing sessions. It is evaluated by means of Equation (15):

$$\begin{aligned} &AIO(S,k) = true \iff \\ &\exists\ S_i\ \in S \mid SCCS(S_i) = CCS(S). \end{aligned} \tag{15}$$

This predicate is true when it exists at least a testing session having a coverage set including the ones of all the other testing sessions. It can be also evaluated by comparing the difference in coverage between the cumulative coverage of all the testing sessions and the coverage of each testing session: If it exists a testing session having no difference in coverage with the cumulative coverage, then the predicate is true.

It is possible to observe that the condition AEQ is equivalent to the condition $D(V_1, V_2) = 0$ when $NTS = 2$. However, the condition AIO implies that the coverage of V_1 includes the one of V_2 or vice versa.

4.3 An Example of Execution of the Multiple Random Testing Process

Figure 2 shows the results of the application of the proposed testing process 3 in terms of code coverage.

The figure reports the code coverage percentages achieved by eight testing sessions (TS1 to TS8) executed by the Multiple Random Testing Process 3 on the SaxPath[1] Java library by using the Random Testing Technique *RTT* implemented by the Randoop tool[2] [89, 90]. In this process, at each iteration the AUT has been tested for 20s for each testing session.

As the figure shows, there is a transient unreliability zone (instability phase) of the process (between 0 and about 700s) where the testing sessions achieve different and floating coverage results, while the coverage of each testing session tends to continuously increase. After 700s the testing session TS4 is the first one that reaches the coverage value of 60.53% of LOCs; this value

[1] https://sourceforge.net/projects/saxpath/.
[2] https://randoop.github.io/randoop/.

will be not improved in the following by any of the testing sessions. After 3,320s, all the eight testing sessions reach this coverage level. If the process continues by running for more than 3,320s (inefficiency zone), then no further improvements in code coverage can be observed from that point for any testing sessions, resulting in both a waste of time and computing resources. This is due to the saturation effect and, we can assume that 60.53% is the saturation level for this testing process.

With respect to this example, the AIO termination condition becomes true after 700s, when the coverage set of TS4 becomes equal to the union of the coverage of all the testing sessions and it includes 60.53% of LOCs, whereas the AEQ termination condition becomes true after 3,320s, when all the coverage sets of all the testing sessions are equal between them and they include 60.53% of LOCs, too. In this example, we can observe that the saturation level of coverage has been reached both at the point when AIO is true and when AEQ is true, so both the two termination criteria have been able to stop the testing process at the reaching of the saturation level of coverage and they both represents cost-effective solutions to the termination problem.

5 FIRST EXPERIMENT ON GUI TESTING OF JAVA APPLICATIONS

The experiment presented in our previous work [3] concluded that random testing processes adopting the two proposed termination criteria AEQ and AIO are both generally able to terminate the testing process at the saturation point. Moreover, it was observed that the adoption of the AIO criterion reduces the number of tests needed to reach the saturation point with respect to the adoption of the AEQ criterion. These conclusions were obtained on the basis of the results of an experiment involving 18 open source Android applications; these apps were tested once in the context of a random testing process with 12 testing sessions and adopting the random testing technique implemented by the Android GUI Ripper tool.[3]

To extend the validity of the experimental results of that preliminary study, we performed a further experiment. In this experiment, we moved to the context of Java GUI-based desktop applications and decided to investigate the performance of the proposed termination criteria applied to the multiple random testing process in this different context. In particular, we wanted to evaluate if the proposed termination criteria AEQ and AIO are able to stop the Multiple Random Testing processes at the Saturation Level of code coverage and what is the testing effort needed to reach that level. In addition, we wanted to study how the number of executed testing sessions *NTS* affects the performance of Multiple Random Testing processes adopting the AEQ and AIO termination criteria. Finally, we aimed at comparing the performance of random testing processes adopting the two proposed termination criteria AEQ and AIO and other termination criteria presented in the literature.

More precisely, the experiment addressed the following research questions:

- RQ1 How does the capability of random testing processes based on AEQ and AIO to reach the saturation level at the termination point vary with the adopted termination criterion and the number of executed sessions?
- RQ2 How do the termination points of the testing processes based on the AEQ and AIO vary with the adopted termination criterion and the number of executed sessions?
- RQ3 How do the termination points and the capability to reach the saturation level of the testing processes adopting AEQ and AIO compare with the ones achieved using other termination criteria presented in the literature?

[3] https://github.com/reverse-unina/AndroidRipper.

5.1 Experimental Objects

In this experiment, we selected a sample of Java GUI-based applications satisfying a set of inclusion and exclusion criteria. These criteria were designed to obtain applications representative of real-world small applications and that could be tested by our testing infrastructure and according to the experimental design.

According to the first inclusion criterion, we considered only open source applications. Although the proposed termination technique based on code coverage evaluation could be applied also on applications for which only the bytecode is available, we required the availability of the source code to allow the detailed analysis of the covered code and the manual evaluation of the feasibility of the uncovered LOCs.

The second inclusion criterion required that the selected applications presented interactive GUIs, since we wanted to test them by a random GUI-based testing technique.

Third, we required that the GUIs of the selected apps used the standard widgets provided by the Java AWT and Swing libraries.

Fourth, we required that the applications belonged to different popular categories, including games (that could involve randomness in their execution), text editors and utility applications.

Some exclusion criteria were also applied. We discarded applications having strong dependencies on other applications or external libraries, since we wanted to test the applications in isolation and did not want to implement too invasive changes of the application code.

In addition, due to limitations in our testing resources, we fixed a maximum testing time per session and discarded the applications for which the proposed termination criteria were not verified within a testing time of one week per session.

We selected 18 Java GUI desktop applications, having different numbers of GUI interfaces, handlers and widgets. They were obtained either from open repositories such as Sourceforge and Github, or by free Google searches. In particular, some of them are programming examples presented in Java programming books. Some other ones were found on personal pages of independent programmers. In particular, one of them (Universal Password Manager, UPM) had been used also in the experiment reported in Reference [78] involving black box system testing of GUI-based applications.

Table 2 reports a description of the 18 applications. For each application the table lists an identifier, the name, a description, and some structural metrics. The number of classes, methods, blocks, and LOCs have been automatically computed by using the *Emma* library[4] during the code instrumentation needed for code coverage measurement. Some other metrics have been measured by means of automatic and manual inspection of the source code of the selected applications. They include the number of interactive widgets, the number of listeners implemented in the source code, the number of input text fields, dialog windows, and clickable menu items. Interactive widgets include several different types of widgets such as buttons, text fields, check buttons, combo boxes, lists, and so on.

As to the size of the considered applications, most of them are small sized (with a number of executable LOCs varying from 268 to 795), whereas applications A9 and A18 are larger than the other ones, having, respectively, 2,367 and 2,388 executable LOCs. A9 and A18 have almost the same number of LOCs but they differ as to the number of menu items (47 in A18 and only 16 in A9) and the number of text edit fields (16 in A9 and only 3 in A18). They represent examples of two common types of interactive applications, since A9 is a form-based application, whereas A18 has a main GUI interface and a large number of clickable buttons and menu items. Figures 3 and 4, respectively, show the main GUI interfaces of the applications A9 and A18.

[4] http://emma.sourceforge.net/.

Table 2. Characteristics of the AUTs Considered in the First Experiment

Application	Name	Brief Description	Number of Interactive Widgets	Number of Listeners	Number of Text Fields	Number of Dialogs	Number of Menu Items	Number of Classes	Number of Methods	Number of Blocks	LOCs
A1	Remember Password	Management of password lists	5	5	5	1	0	8	36	1402	294
A2	Scientific Calculator	Pocket Calculator with scientific functions	16	17	0	0	10	19	51	1187	268
A3	Address Book	Management of personal addresses	16	13	8	0	0	3	17	1586	363
A4	Dietetics	Calculator of dietological indexes	8	12	4	5	5	15	37	2031	448
A5	Minesweeper	Java porting of the minesweeper game	484	482	0	1	0	10	61	1927	386
A6	SimpleLibrary	Application managing lists of books	16	8	6	6	8	24	146	2200	570
A7	Quiz	15 questions quiz game	10	12	1	7	5	12	45	2785	438
A8	BlackJack	Java porting of the black jack card game	13	12	1	1	0	21	102	3882	762
A9	UPM	Manager of personal data and passwords	31	69	16	8	16	72	342	10291	2367
A10	Jnotepad	Text Editor	7	22	3	4	15	25	111	2525	574
A11	Simple Calculator	Pocket Calculator with scientific functions	24	24	0	1	2	5	24	1247	302
A12	Movie Catalog	Manager of lists of movies	15	11	5	2	6	25	136	2713	692
A13	Bibtex Manager	Editor of bibtex entries	12	17	4	2	9	46	182	3616	795
A14	QRCode Generator	Generator of QR code from input text	7	21	17	0	0	27	98	3009	662
A15	Shopping List Manager	Manager of shopping carts	8	7	2	0	0	19	78	1497	362
A16	HangMan	Java porting of the hangman word game	30	30	0	2	0	5	32	1434	324
A17	Unit Converter	Converter between physical units	4	4	59	1	0	14	74	2420	537
A18	Beagtex	Latex text editor	32	58	3	7	47	81	248	9955	2388

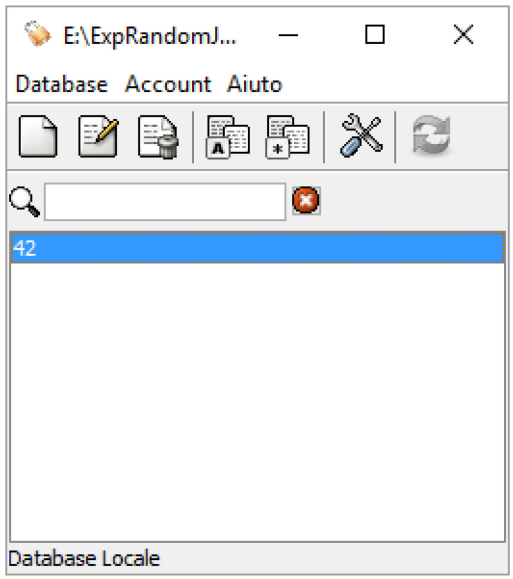


Fig. 3. An instance of the GUI of the UPM application (A9).

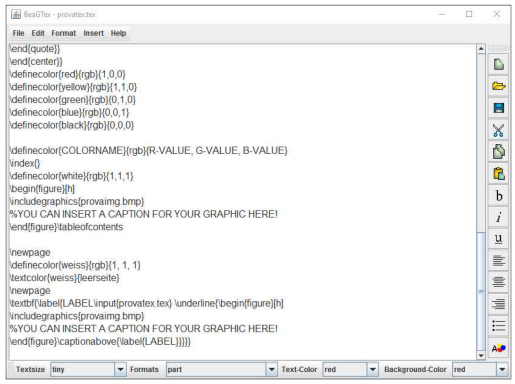


Fig. 4. An instance of the GUI of the BeagTex application (A18).

As to the category, A5, A7, A8, and A16 are simple games, and the execution of three of them depends on randomness (the positions of mines in A5–Minesweeper, the randomly extracted cards in A8–BlackJack, the word to be guessed in A16–HangMan). A10, A13, and A18 are text editors, whereas all the others are utility applications. In particular, A1, A3, A6, A9, A12, and A15 are somewhat similar between them, because they are applications for the management of lists of items (such as books, movies, addresses, passwords).

Eleven of 18 applications included calls to external Java libraries dialog windows (A1, A3, A6, A7, A9, A10, A12, A13, A14, A15, A18). Since in our process we did not evaluate the coverage of external libraries code and, we wanted to limit the file system resources accessed by the AUTs during the random testing, we modified the code of these applications by introducing stubs. In particular, the calls to dialogs used to select the files to be opened or saved, the printer, or a color have been substituted by calls to stubs returning predefined values.

The application A5 (Minesweeper) has a large number of buttons and event handlers, because it dynamically instances a matrix of clickable buttons for each possible cell of the game. The modified source code of each application under test is available at http://reverse.dieti.unina.it/index.php/27-expsaturation.

5.2 Variables

This experiment has two independent variables: the termination criterion *TerC* and the number of testing sessions *NTS*. The dependent variables are the level of coverage *terL* reached by the testing process and the testing time *terP* needed to achieve this value.

As regards the termination criterion *TerC*, we considered seven different values, associated with seven termination criteria. The first two considered criteria are the AEQ and AIO proposed and defined in Section 4.2. The other five criteria had been derived from the ones proposed in literature and discussed in Section 2.2.

Three criteria are time based, according to which the testing process is stopped after a predefined testing time. To establish a testing time value to be used in the experiment, we considered the works of Arcuri and Fraser [11, 40] and the ones of Ciupa et al. [31, 32] that fixed testing time values proportional to the number of classes of the AUTs. In References [11] and [40] a testing timeout value of 2 minutes per class was fixed. The applications tested with this timeout in those articles are similar in typology to the ones involved in our experiment, but they are generally composed of more complex classes. They have a density of LOCs per class that is about double than the density of the AUTs involved in our experiment. For this reason, in this study we defined the *t1* criterion terminating the process after a testing time in minutes for testing session that is equal to the number of classes of the application under test, and the *t2* criterion according to which the testing process will be stopped after a number of minutes for testing session that is equal to the double of the number of classes of the application under test. In References [31] and [32], a testing time of 90 minutes per class had been experimented. Since the libraries under test in those experiments are composed of complex classes having a density of LOCs per class that is about 13 times the one of the AUTs in this experiment, we have defined a *t7* criterion stopping the testing process after a number of minutes for testing session that is seven times the number of classes of the application under test.

Another termination criterion that we considered is related to software reliability properties and is coherent with the one proposed by Bueno and Jino [17] and reported in Section 2.2. A termination criterion called *t50* was derived by fixing the following values in the Assertions 1 and 2 reported in Reference [17]:

$$\delta = \Delta = 0; \tag{16}$$

$$NL = i/2. \tag{17}$$

In other words, the *t50* termination criterion stops a testing process when the last 50% of iterations have been spent without observing any increase in the cumulative coverage obtained by the executed testing sessions. The values of parameters δ and Δ were fixed to zero, since we expect that the proposed testing process terminates at the exact reaching of the saturation level of coverage. As to the number of iterations without coverage improvements, the chosen NL value is not constant but it is defined in terms of the number i of performed process iterations. In that sense, the proposed criterion is different from other ones in the literature, such as the one adopted by Mansour et al. [77] that stops the search-based testing process after 50 consecutive iterations without any improvement of the fitness metric.

Finally, we considered a further termination criterion (called *tm*) that stops the testing process when it reaches the maximum value of cumulative coverage obtained using all the other

Table 3. Definitions of the Termination Criteria Considered in the Experiment

TerC	Termination Criterion Description
AEQ	It is true when all the testing sessions have covered the same set of testing targets.
AIO	It is true when it exists at least a testing session having a coverage set including the ones of all the other testing sessions.
t1	It is true after a testing time in minutes for testing session that is equal to the number of classes of the application under test
t2	It is true after a testing time in minutes for testing session that is 2 times the number of classes of the application under test
t7	It is true after a testing time in minutes for testing session that is 7 times the number of classes of the application under test
t50	It is true when the last 50% of iterations have been spent without observing any increase in the cumulative coverage obtained by the testing process
tm	It is true when the maximum value of cumulative coverage obtained using the other considered criteria has been reached.

considered criteria. It represents an optimal criterion, because the corresponding testing process always reaches the maximum coverage and terminates exactly when this coverage is reached. Of course, it can be considered only in the context of a comparative experiment, since in a real testing process it is not possible to know when the maximum coverage value is reached.

The definitions of the seven termination criteria considered in the experiment are summarized in Table 3.

As regards the number of testing sessions independent variable *NTS*, in the experiment presented in Reference [3] 12 testing sessions were considered. To evaluate if a reduced number of testing sessions may produce the same results with a reduced amount of testing time, other values of *NTS* were considered. In particular, we considered four possible values, i.e., $NTS \in \{2, 4, 8, 12\}$ testing sessions.

As regards the dependent variables *terP* and *terL*, we used the following metrics to evaluate them:

- the *Termination Point* ($terP$) is measured by the number of test cases per session after which the termination criterion $TerC$ of the testing process TP is verified for the first time;
- the *Termination Level* ($terL$) is measured by the cumulative statement coverage percentage reached by the testing process TP when the termination criterion $TerC$ is verified for the first time.

5.3 Experimental Materials and Procedure

To execute the experiment, we had to instantiate the testing process described by the Procedure 3 providing values to its five parameters: *AUT*, *AUTPr*, *RTT*, *NTS*, and *TerC*. In the following, we report the selected values.

As to the AUT parameter, we considered 18 different values, each one associated to one of the applications named A1, . . . , A18 listed in Section 5.1.

As regards the preconditions *AUTPr*, for each *AUT* we had to establish a specific precondition and each test was executed from this starting point.

In 6 of 18 applications (i.e., A2, A4, A5, A8, A11, A17) no preconditions were set. In 9 applications presenting features for opening and saving files (i.e., A1, A3, A6, A7, A9, A10, A12, A15, and A18), the selected precondition is that no previously saved files exist before each test. In other two

Table 4. Testing Process Parameters Values

AUT	18 AUTS: {A1, . . . , A18}
AUTPr	A fixed precondition for each of the 18 AUTs
RTT	Java Random Ripper tool
NTS	{2, 4, 8, 12}
TerC	{AEQ, AIO, t1, t2, t7, t50, tm }

applications (i.e., A13 and A14) the fixed precondition is that there exists an input file that can be loaded by the AUT. This precondition was selected, because these applications have no features allowing to save files. Finally, in the application A16 a constant word to be guessed was fixed.

As regards the random testing technique *RTT*, we considered a random testing technique that we implemented in a testing tool called *Java Random Ripper*. This tool generates and executes random sequences of GUI events on the AUT. Each event is chosen at random from the ones executable on the current GUI of the AUT, assigning each event the same probability to be selected.

Each test case executed by this *RTT* consists of a sequence of 200 GUI events. To choose this length of the sequence, the observations reported in several related work in literature were taken into account (e.g., the ones of Andrews et al. [6], Fraser and Gargantini [41], and Arcuri [7]). In particular, Arcuri demonstrated that sequences of 100 consecutive function calls test more code than sequences of 7 consecutive function calls, while Andrews et al. [6] found the best fault detection capability when testing sequences of 200 function calls. The testing tool was configured to inject always the same constant value in each text field.

The *Java Random Ripper* tool has been implemented by exploiting the FEST (Fixtures for Easy Software Testing)[5] and the Java Robot libraries[6] to emulate user interactions with the GUI of the application. Currently only AWT and Swing libraries are supported by the tool. To evaluate the code coverage achieved by the random testing process, the Emma library had been used to measure the code coverage. An executable copy of the tool allowing the replication of the presented experiments is available at http://reverse.dieti.unina.it/index.php/27-expsaturation.

As regards the number of testing sessions (*NTS*), we considered the values of 2, 4, 8, and 12.

As to the TerC parameter of the Procedure 3, we considered the seven termination criteria *terC* (i.e., AEQ, AIO, t1, t2, t7, t50, tm) reported in the previous subsection.

The testing process parameter values used in the experiment are summarized in Table 4.

As to the experimental procedure, in each testing process we executed the selected number of testing sessions *NTS*. In each session, we ran test cases made of sequences of 200 GUI events. Before the execution of each test case, we reset the same preconditions and set a different random seed value. We iteratively evaluated the coverage levels and the termination criteria at a fixed step of 100 events. When a considered termination criterion was met, we evaluated the corresponding termination level *terL* and the termination point *terP*. Moreover, at the end of the process, for each application we evaluated its saturation level, that is the maximum coverage set that could be reached when the considered testing process is executed with the fixed preconditions, technique, and execution environment. To this aim, we manually analyzed the obtained coverage sets to evaluate whether the residual uncovered code could be exercised during the considered process.

The testing processes were carried out on PCs equipped with an Intel I7 processor with a 3GHz clock, 8GB of RAM, having a Solid State Disk (SSD) and using the Windows 7 operating system (64 bits) and Java Runtime Environment 7.

[5] https://code.google.com/archive/p/fest/.
[6] https://docs.oracle.com/javase/7/docs/api/java/awt/Robot.html.

Table 5. Median Values of the Termination Levels *TerL* (Measured in Terms of Code Coverage Percentage) of Testing Processes Adopting AEQ and AIO, Involving 12, 8, 4, and 2 Testing Sessions

NTS	AEQ				AIO				Sat Level %
	12	8	4	2	12	8	4	2	
A1	**97.3% (10)**	**97.3% (10)**	**97.3% (10)**	**97.3% (10)**	**97.3% (10)**	**97.3% (10)**	**97.3% (10)**	**97.3% (10)**	97.3%
A2	**87.1% (10)**	**87.1% (10)**	**87.1% (10)**	**87.1% (7)**	**87.1% (10)**	**87.1% (9)**	**87.1% (7)**	84.9% (2)	87.1%
A3	**92.3% (10)**	**92.3% (10)**	**92.3% (8)**	90.4% (4)	90.4% (4)	90.4% (2)	90.4% (1)	85.1% (0)	92.3%
A4	**84.2% (10)**	**84.2% (10)**	**84.2% (10)**	**84.2% (9)**	**84.2% (10)**	**84.2% (8)**	84.0% (3)	83.1% (1)	84.2%
A5	93.7% (0)	93.7% (0)	93.7% (0)	91.6% (0)	92.6% (0)	92.6% (0)	92.6% (0)	91.6% (0)	96.2%
A6	**89.0% (10)**	**89.0% (10)**	**89.0% (10)**	**89.0% (8)**	**89.0% (10)**	**89.0% (9)**	**89.0% (7)**	88.9% (4)	89.0%
A7	77.5% (0)	77.5% (0)	77.5% (0)	77.0% (0)	77.5% (0)	77.5% (0)	76.1% (0)	76.1% (0)	85.3%
A8	**94.0% (10)**	**94.0% (10)**	**94.0% (8)**	93.4 (4)	93.8% (3)	93.8% (3)	93.4% (1)	92.7% (0)	94.0%
A9	**72.1% (10)**	**72.1% (10)**	**72.1% (10)**	**72.1% (9)**	**72.1% (10)**	**72.1% (10)**	**72.1% (8)**	71.9% (4)	72.1%
A10	**89.5% (10)**	**89.5% (10)**	**89.5% (10)**	**89.5% (10)**	**89.5% (10)**	**89.5% (10)**	**89.5% (10)**	**89.5% (10)**	89.5%
A11	98.3% (0)	98.3% (0)	98.3% (0)	98.3% (0)	98.3% (0)	98.3% (0)	98.3% (0)	97.6% (0)	98.3%
A12	**88.0% (10)**	**88.0% (10)**	**88.0% (8)**	**88.0% (6)**	**88.0% (9)**	**88.0% (8)**	**88.0% (6)**	87.7% (3)	88.0%
A13	**80.5% (10)**	**80.5% (10)**	**80.5% (9)**	80.4% (4)	80.4% (4)	80.4% (2)	79.8% (0)	79.8% (0)	80.5%
A14	**78.5% (10)**	**78.5% (10)**	**78.5% (10)**	**78.5% (10)**	**78.5% (10)**	**78.5% (10)**	**78.5% (10)**	**78.5% (10)**	78.5%
A15	**83.0% (10)**	**83.0% (10)**	**83.0% (10)**	**83.0% (10)**	**83.0% (10)**	**83.0% (10)**	**83.0% (10)**	**83.0% (10)**	83.0%
A16	**95.1% (10)**	**95.1% (10)**	**95.1% (10)**	**95.1% (10)**	**95.1% (10)**	**95.1% (10)**	**95.1% (10)**	**95.1% (10)**	95.1%
A17	**51.8% (10)**	**51.8% (10)**	**51.8% (10)**	**51.8% (10)**	**51.8% (10)**	**51.8% (10)**	**51.8% (10)**	**51.8% (10)**	51.8%
A18	**64.4% (10)**	**64.4% (10)**	**64.4% (8)**	63.6% (2)	**64.4% (8)**	**64.4% (6)**	63.6% (3)	63.2% (1)	64.4%
fSat	150/180	150/180	141/180	113/180	128/180	117/180	96/180	75/180	

To mitigate the threat that the obtained experimental results were due to randomness, we performed 10 independent repetitions of each considered testing process. In each repetition, we used the same testing pre-conditions and configuration of the application, but different starting seed values. All the dependent variables have been measured once for each experiment repetition.

To compare the saturation capability of different testing techniques, we used the non-parametric Fisher's exact test [35], as suggested by Arcuri and Briand [10]. To compare the costs of different testing techniques, we used the Shapiro-Wilk test to assess whether the costs were normally distributed. Since these tests showed that the hypotheses related to the normal distribution could be always rejected, we used the non-parametric Mann–Whitney–Wilcoxon test (as suggested by Arcuri and Briand [10]) to evaluate null hypotheses related to their comparisons. All the statistical tests were calculated by using the R tool.

5.4 Results

At the end of the execution of the testing processes, we obtained the values of the *terP* termination points and of the *terL* termination levels.

Table 5 reports the median values of the termination levels *terL* (in percentage of covered LOCs) of the processes adopting the AEQ and the AIO termination criteria and 12, 8, 4, or 2 testing sessions, measured on the set of 10 experiment repetitions. The last table column lists the *SatLevel* saturation levels of coverage of each application. In the table, the median values of the termination levels that are equal to the corresponding saturation levels have been highlighted in bold. Of course, since the termination levels cannot exceed the saturation levels, we can observe that the median of the termination levels is equal to the saturation level if the majority of the executed

Table 6. *TerP* Median Values of Termination Points (in Terms of Number of Events per Session) of Processes Adopting AEQ and AIO and Involving 12, 8, 4, and 2 Testing Sessions

NTS	AEQ				AIO			
	12	8	4	2	12	8	4	2
A1	300	300	200	200	100	100	100	100
A2	2700	1900	1500	700	200	200	200	100
A3	7700	3100	1500	400	300	400	400	200
A4	21200	18300	13200	11600	4000	4200	2400	700
A5	18800	15400	14300	100	800	1800	2600	100
A6	1800	1600	1300	1100	300	400	600	200
A7	16700	15900	14900	4000	1600	1900	800	600
A8	102500	78900	62700	12100	12700	15300	9800	200
A9	9400	6600	5400	3500	1100	1300	1400	600
A10	900	700	500	400	200	200	300	400
A11	2800	2400	2100	1600	200	300	900	1100
A12	2200	1400	600	500	300	400	500	200
A13	63000	54100	47200	4400	1500	1800	1000	900
A14	800	600	400	300	100	200	200	300
A15	1600	900	700	400	100	200	300	400
A16	700	500	300	200	100	100	200	200
A17	800	700	500	200	100	200	200	300
A18	4538700	3259000	2356300	763700	135600	156200	178800	123200

processes repetitions adopting the considered testing termination criterion exactly reaches the saturation level of coverage. In the last row, the frequency of Saturation *fSat* is also reported. It is evaluated as the fraction between the total number of executed testing processes that reached the saturation level of coverage and the total number of executed testing processes. The data reported in Table 5 show that the random testing processes have covered a large part of the source code of the AUTs (84% in average with a minimum of 51.8% for application A17 and a maximum of 98.3% for A11).

Table 6 shows the median values of the termination points *terP* expressed by the number of executed events per testing session of the same considered processes.

5.4.1 Data Analysis 1: Evaluating the Capability of Reaching the Saturation. To study the capability of the processes to reach the saturation level at the termination point, we first analyzed the results achieved by processes adopting the AEQ criterion and a variable number of sessions, which are reported in the first four columns in Table 5. As the data show, the saturation level was reached in the majority of the experiment replications in 15 of 18 applications when 12, 8, or 4 sessions were executed. When the number of sessions decreased to 2, the saturation was reached in 12 of 18 cases. In particular, it is possible to observe that the testing processes based on 12 or 8 testing sessions had exactly the same capability to reach the saturation level of coverage at their termination.

The three applications for which the Saturation Level was never reached at the termination point using any of the considered termination criteria are A5 (Minesweeper), A7 (Quiz), and A11 (Simple Calculator). The residual uncovered code, respectively, consists of 10 LOCs for A5, 34 LOCs for A7, and 0.2 LOCs for A11. In the following, we explain the reasons why the saturation level was not reached in these three cases.

As regards the Simple Calculator application (A11), the uncovered code is related to the condition included in a *if* statement, corresponding to the case that the user has set as input a number with more than 20 digits. Since digits can be inputted by clicking on the 10 buttons corresponding to decimal digits, and the input operation is interrupted by a click on each of the other 13 buttons of the interface (corresponding to arithmetic operations and mathematical functions), the probability to execute a sequence of 21 clicks on digit buttons can be estimated under 10^{-7}. By applying the Property A.1 in the Appendix, it is possible to estimate that the number of events needed to achieve a probability of 50% to cover this testing target is not less than:

$$k^* = \frac{log(1-0.5)}{log(1-10^{-7})} \cong 693,147. \tag{18}$$

For this reason, it is not surprising that this testing target has not been covered after the execution of testing sessions having at least 2,800 events.

The other two applications, Minesweeper (A5) and Quiz (A7) both belonged to the category of games and the uncovered code is related to the winning of a game. In Minesweeper (A5) the player wins a game when all the cells without mines have been selected and no cells with mines have been selected, whereas in the Quiz application the player wins when he correctly provides answers to 15 consecutive questions each having 4 possible answers and one right answer only. In both cases, the sequence of events for winning of a game has a very low probability to be executed with the selected random testing technique. It is not surprising that even none of the longest testing sessions (having about 20,000 events) have covered it, coherently with the Property A.1 reported in the Appendix. These cases are very similar to the case of applications having the distribution PD_2 (Section 3.4), for which at the termination of the testing process, the most "difficult" targets remained uncovered.

As regards the termination condition AIO, the median values reported in the five rightmost columns in Table 5 show that the processes executing either 12 or 8 sessions often had the same capability of covering the source code and reaching the saturation. In fact, the saturation level is reached by the majority of the testing repetitions in 12 of 18 applications. When the processes with 4 sessions are considered instead, the capability of reaching the saturation decreased, being it reached in majority only in 10 of 18 applications. Even worst results were obtained when 2 sessions were executed, being the saturation reached in the majority of the replications only in 6 cases of 18.

The results discussed thus far have been based on the analysis of the median values of the 10 repetitions of each process execution. We now want to determine whether the observed differences between processes with different numbers of sessions are statistically significant. To this aim, we used the Fisher exact test to verify the null hypotheses reported in the first column of Table 7. The Fisher exact test was performed on the basis of the data reported in the last row of Table 5.

With respect to processes adopting AEQ, the results of the tests show that the processes having 12 or 8 testing sessions have exactly the same capability of reaching the saturation (we cannot reject the first null hypothesis, since the p-value is equal to 1). Processes having 8 testing sessions more often reach the saturation level than processes having 4 testing sessions but we cannot reject the corresponding null hypothesis (the p-value is 0.1420). However, we can accept the alternative hypothesis that processes having 4 testing sessions have a better capability of reaching the saturation level than the ones having 2 testing sessions (the p-value is $0.8628 * 10^{-3}$).

With respect to processes adopting AIO, the results show that processes with 12 testing sessions more often reach the saturation level with respect to the ones having 8 testing sessions, but we cannot reject the corresponding null hypothesis (the p-value is 0.1291). However, processes having 8 testing sessions have a better capability of reaching the saturation than processes having 4 testing

Table 7. Comparisons between the Capability of Reaching the Saturation with Different Numbers of Testing Sessions: Null Hypotheses and Corresponding p-values Evaluated with the Fisher Test

Null Hypothesis H_0	p-value
Processes adopting AEQ and NTS=12 have the same capability of reaching the saturation with respect to processes adopting AEQ and NTS=8	1
Processes adopting AEQ and NTS=8 have a worst capability of reaching the saturation with respect to processes adopting AEQ and NTS=4	0.1420
Processes adopting AEQ and NTS=4 have a worst capability of reaching the saturation with respect to processes adopting AEQ and NTS=2	$\mathbf{0.8628^{*}10^{-3}}$
Processes adopting AIO and NTS=12 have a worst capability of reaching the saturation with respect to processes adopting AIO and NTS=8	0.1291
Processes adopting AIO and NTS=8 have a worst capability of reaching the saturation with respect to processes adopting AIO and NTS=4	**0.0159**
Processes adopting AIO and NTS=4 have a worst capability of reaching the saturation with respect to processes adopting AIO and NTS=2	**0.0173**

sessions (p-value = 0.0159), that have a better capability of the ones having 2 testing sessions (p-value = 0.0173).

5.4.2 Data Analysis 2: Comparing the Capability of Reaching the Saturation at the Same Number of Sessions. We now want to compare the capability of reaching the saturation of processes using different termination criteria, but the same number of sessions.

The frequency of Saturation (*fSat*) values reported in Table 5 show that processes adopting AEQ more frequently reached the saturation than processes adopting AIO. In particular, the *fSat* values of processes using AEQ were 150, 150, 141, and 113 with 12, 8, 4, and 2 sessions, respectively, whereas the values in the analogous processes using AIO were 128, 117, 96, and 75.

The greatest differences in coverage were observed in four applications, named A3, A5, A8, and A13. To explain these differences, we analyzed the code of the applications and concluded that they were due to the existence of code portions that could be covered only when very specific sequences of events were triggered.

As an example, in A3 the coverage sets with AEQ and AIO and 12 sessions differed of 26 lines of code. These lines of code could be covered only if the specific sub-sequence of events that first insert a new address in the address book and, successively, find the same address using the search functionality was executed. In A5, instead, the uncovered code could be activated only when the user sets a flag on a randomly selected cell containing a mine and successively re-selects just this cell and removes the flag.

These examples confirm that these portions of code were not covered, since these specific sequences of events had a low probability to be executed by a uniform random testing technique. This probability was even lower in processes using AIO, since this criterion usually requires a less stringent condition to end the process.

These data indicate that the capability of reaching the saturation of processes using different termination criteria, but the same number of sessions, is different, with AEQ outperforming AIO.

To assess whether this difference is statistically significant, we performed the Fisher test to evaluate the null hypotheses reported in Table 8. The Fisher test relied on the *fSat* values reported in the last row of Table 5. Table 8 reports in the first column the proposed null hypotheses and in the latter column the corresponding Fisher test p-values.

Table 8. Comparisons between Capabilites of Reaching the Saturation: Null Hypotheses and Corresponding p-values Evaluated with the Fisher Test

Null Hypothesis H_0	p-value
Processes adopting AIO and NTS=12 have a better capability of reaching the saturation with respect to processes adopting AEQ and NTS=12	$\mathbf{0.4040^{*}10^{-2}}$
Processes adopting AIO and NTS=8 have a better capability of reaching the saturation with respect to processes adopting AEQ and NTS=8	$\mathbf{0.5188^{*}10^{-4}}$
Processes adopting AIO and NTS=4 have a better capability of reaching the saturation with respect to processes adopting AEQ and NTS=4	$\mathbf{0.4140*10^{-6}}$
Processes adopting AIO and NTS=2 have a better capability of reaching the saturation with respect to processes adopting AEQ and NTS=2	$\mathbf{0.4480*10^{-4}}$

All the p-values are under the threshold $\alpha = 0.05$. Consequently, all the null hypotheses could be rejected and the corresponding alternative hypotheses could be accepted. In other words, the processes adopting AEQ have always a better capability of reaching the saturation level of coverage than the processes adopting AIO, if the same number of testing sessions is considered.

5.4.3 Answer to RQ1. On the basis of the performed analysis, we could answer to RQ1 and conclude that:

- the capability of processes based on AEQ of reaching the saturation improves when the number of the executed testing sessions grows from 2 to 8;
- processes based on AEQ and executing 8 or 12 testing sessions have a very similar capability of reaching the saturation;
- if we exclude a few testing targets having very low probability to be covered by random testing, processes based on AEQ and executing 8 or 12 testing sessions always reach the saturation at the termination point.
- the capability of processes based on AIO of reaching the saturation improves when the number of sessions grows up to 12;
- the capability of processes based on AIO of reaching the saturation is always lower than the one of processes adopting AEQ, for equal number of sessions.

5.4.4 Data Analysis 3: Evaluating the Termination Points. To investigate the cost of processes adopting AEQ or AIO as a termination criterion, we analyzed and compared the termination point values that were collected in the different process executions. An overview of these values is provided in Table 6 that reports the median values of the observed termination points.

As the data in Table 6 show, the median values of the termination points varied considerably with the applications and with the number of sessions. The minimum number of executed events per session varied between 200 events for AEQ-based processes, and 100 events for the ones adopting AIO. It must be remarked that these values are strongly influenced by the circumstance that the termination criteria are evaluated with a sampling step of 100 events. The maximum observed values were of the order of magnitude of millions of events, and were observed in a single application, A18, that is one of the two greatest applications of the data set. However, if we consider the termination points of the other greatest application, i.e., A9, we do not observe similar large values of the termination points. Indeed, the termination points of testing processes involving A9 had a median value of only 1,200 events (with AIO) and 5,800 events (with AEQ).

To explain this large difference, we analyzed the GUIs and the code of the two applications and the trends of the coverage achieved by each random testing process. The GUIs of A9 (the

Table 9. Comparisons between Number of Executed Events: Null Hypotheses and Corresponding p-values Evaluated with the Mann–Whitney–Wilcoxon Test

Null Hypothesis H_0	p-value
Processes adopting AEQ and NTS=12 reach the termination point earlier than processes adopting AEQ and NTS=8	0.2003
Processes adopting AEQ and NTS=8 reach the termination point earlier than processes adopting AEQ and NTS=4	$\mathbf{0.3571*10^{-7}}$
Processes adopting AEQ and NTS=4 reach the termination point earlier than processes adopting AEQ and NTS=2	$\mathbf{0.2223*10^{-14}}$
Processes adopting AIO and NTS=12 reach the termination point later than processes adopting AIO and NTS=8	**0.0109**
Processes adopting AIO and NTS=8 reach the termination point later than processes adopting AIO and NTS=4	0.4582
Processes adopting AIO and NTS=4 reach the termination point earlier than processes adopting AIO and NTS=2	0.0914

UPM password manager utility tool) present only few buttons and menu items. The execution of large part of source code depends on input string values. Since the Java Random Ripper tool has been designed to provide prefixed string values, it is not able to explore parts of code that can be reached only when specific input values are provided [5]. This explains why the termination has been reached after a relatively small number of events, similar to the ones needed for reaching the termination in smaller applications. In other words, the behavior of the application can be considered similar to the one exhibited by applications having a probability distribution of testing targets coherent with the PD_1 distribution described in Section 3.

The code coverage of A18 (the Beagtex Latex editor), instead, increases very slowly with the number of executed events. The reason of this behavior is that there is a large set of possible events that can be activated on the main interface (i.e., events related to menu items and to the visible buttons) and there are many portions of code activated only by specific sequences of events (e.g., code related to the execution of a specific action just after the selection of some text in the editor). For this reason, there are many portions of code having a low probability to be covered by random events, and their coverage proceeds continuously but very slowly. This behavior can be considered similar to the one exhibited by applications having a testing targets probability distribution coherent with the distribution PD_3 described in Section 3.

On the basis of this analysis, we had to exclude that the observed termination point trends depended on the size of the applications. Rather, this trend seemed more likely to depend on the probability distribution of the testing targets, which is usually unknown before the testing process execution.

Therefore, we considered the data of processes adopting the same termination criterion but different number of sessions. The data reported in Table 6 show that with AEQ the median values of the termination points generally grow with the number of sessions. The greater the number of sessions, the greater the number of events needed to reach the termination. On the contrary, this behavior was not observed in processes adopting AIO.

We used statistical analysis to investigate the relationship between termination points of processes adopting the same criterion, but different number of sessions. We made the null hypotheses reported in Table 9 and tested them using the Mann–Whitney–Wilcoxon test (using the termination point values observed in all the process repetitions). Table 9 shows the resulting p-values in its second column.

Table 10. Comparisons between Number of Executed Events: Null Hypotheses and Corresponding p-values Evaluated with the Mann–Whitney–Wilcoxon Test

Null Hypothesis H_0	p-value
Processes adopting AEQ and NTS=12 reach the termination point earlier than processes adopting AIO and NTS=12	$<0.22^{*}10^{-15}$
Processes adopting AEQ and NTS=8 reach the termination point earlier than processes adopting AIO and NTS=8	$<0.22^{*}10^{-15}$
Processes adopting AEQ and NTS=4 reach the termination point earlier than processes adopting AIO and NTS=4	$<0.22^{*}10^{-15}$
Processes adopting AEQ and NTS=2 reach the termination point earlier than processes adopting AIO and NTS=2	$<0.22^{*}10^{-15}$

As regards AEQ, we can observe that we can reject two of three null hypotheses, so that we can affirm that the number of events to reach the termination points grows when the number of testing sessions grows from 2 to 4 (p-value = $0.2223{*}10^{-14}$) and from 4 to 8 sessions (p-value = $0.3571{*}10^{-7}$). However, we cannot reject that the processes with 12 testing sessions may terminate earlier than processes with 8 testing sessions. The measured p-value (0.2003) only indicates that processes having 12 testing sessions are often (but not definitely) longer than processes having 8 testing sessions. The general growth of the termination points with the number of testing sessions can be explained by considering that the termination condition AEQ is reached when all the testing sessions have reached the same coverage level. Intuitively, larger the number of testing sessions, larger is the time needed to reach this condition.

As to the processes adopting AIO, the unique hypothesis that can be rejected is that processes with 12 testing sessions terminate later than processes with 8 testing sessions (p-value = 0.0109). In other words, the test indicates that processes with more sessions may terminate earlier. We can explain this behavior by considering that processes adopting AIO can be terminated when at least one of the sessions reaches the cumulative coverage. The greater the number of sessions, the greater the likelihood that at least one of the sessions early reaches the cumulative coverage of the remaining sessions.

However, we cannot reject any hypotheses involving processes with four or two testing sessions. The reason is that often the coverage at the termination point does not reach the saturation by processes with four or two testing sessions (as shown in the previous subsection). In these cases, the termination points are reached earlier.

5.4.5 Data Analysis 4: Comparing the Termination Points at the Same Number of Sessions. In our study, we also wanted to investigate the relation between the termination points of processes adopting different termination criteria and the same number of sessions. To compare processes adopting AEQ and processes adopting AIO, for each application we evaluated the ratio between the average value of the termination points of all processes adopting AEQ and the average value of the termination points of all processes adopting AIO. Therefore, we averaged the values of this ratio among all the applications. The obtained result was that the termination points of processes adopting AEQ are about 10 times greater than the ones of processes using AIO on average. This datum could suggest that the AIO criterion allows testing time saving with respect to AEQ.

To have a statistical confirmation that the testing processes adopting AIO are able to terminate earlier than the processes adopting AEQ, we compared the termination points measured for testing processes having the same number of executed testing sessions. Table 10 shows in the first column the proposed null hypotheses and in the latter column the corresponding p-values for the

Mann–Whitney–Wilcoxon test. All the p-values are less than $0.22 * 10^{-15}$, i.e., they are practically equal to zero. As a consequence, all the proposed hypotheses can be rejected and, we can conclude that testing processes adopting AIO terminates earlier than processes adopting AEQ, for equal number of sessions.

5.4.6 Answer to RQ2. On the basis of the performed analysis, we could answer to RQ2 and conclude that:

- The termination points of processes adopting AEQ grow when the number of sessions grows from 2 up to 8 sessions;
- The termination points of processes adopting AEQ and executing 12 testing sessions are often (but not definitely) greater than the ones of processes executing 8 testing sessions;
- The termination points of processes adopting AIO and 8 sessions are generally smaller than the ones of processes executing 12 testing sessions;
- The termination points of processes adopting AIO are generally smaller that the ones of processes adopting AEQ, for equal number of sessions.

5.4.7 Data Analysis 5: Evaluating the Termination Levels of Processes Adopting AEQ, AIO, t1, t2, t7, t50, and tm. The results presented in the previous subsections show that the choice of 8 testing sessions represents a good solution for both processes adopting AEQ and AIO, both in terms of capability of reaching the saturation level of coverage and in terms of saving the testing cost. For this reason, in this subsection we will compare the testing processes adopting the proposed termination criteria and executing 8 testing sessions with processes adopting other termination criteria presented in the literature.

Table 11 reports the median values of the termination points $terP$ and of the corresponding termination levels $terL$ measured for each of the selected termination criteria $t1$, $t2$, $t7$, $t50$, and tm for testing processes involving 8 testing sessions. The number of process repetitions that reached the saturation level of coverage (of 10) are reported in parenthesis beside the coverage levels. The median coverage values corresponding to the saturation levels have been highlighted in bold. The last row of the table reports the frequency of Saturation $fSat$ evaluated as the fraction between the total number of executed testing processes that reached the saturation level of coverage and the total number of executed testing processes.

The termination points and the termination levels obtained with these termination criteria were compared with the ones obtained using the proposed termination criteria AEQ and AIO in the same testing processes. The latter values were reported in Table 5 and 6.

The data reported in Table 11 show that the saturation level of coverage was reached in the majority of the processes repetitions. More precisely, it was reached in 10 applications when applying the $t1$ termination criterion, 11 applications when applying $t2$, 12 applications when applying $t7$, and 11 applications when applying $t50$. However, it was reached in 15 applications when applying AEQ and in 12 applications when applying AIO.

These data would suggest that AEQ is the criterion having the best capability of reaching the saturation, with respect to the other criteria. The capability of AIO instead seems more similar to that of the other termination criteria from the literature.

To have a visual comparison of the performance of processes adopting different termination criteria, Figure 5 shows the median values of the difference of coverage with respect to the saturation level for each of the termination criteria $t1$, $t2$, $t7$, $t50$, AEQ, and AIO. The coverage reached adopting tm as termination criterion always coincides with the one obtained adopting AEQ for each of the considered applications, thus it has not been reported in the figure.

Table 11. Median Values of the Termination Levels *TerL* (in Terms of Code Coverage Percentage) and of the Termination Points *TerP* (in Terms of Number of Executed Events per Session) Obtained with the Termination Criteria $t1$, $t2$, $t7$, $t50$, and tm

		$t1$		$t2$		$t7$		$t50$		tm	
		terL	terP	terL	terP	terL	terP	terL	terP	terL	terP
A1	Remember Password	**97,3% (10)**	300	**97,3% (10)**	600	**97,3% (10)**	2100	**97,3% (10)**	300	**97,3%**	100
A2	Scientific Calculator	**87,1% (10)**	1600	**87,1% (10)**	3200	**87,1% (10)**	11000	**87,1% (10)**	300	**87,1%**	100
A3	Address Book	90,4% (3)	500	**92,3% (5)**	1000	**92,3% (10)**	3600	90,4% (3)	400	**92,3%**	700
A4	Dietetics	84,2% (2)	600	84,2% (3)	1200	**84,2% (8)**	4100	**84,2% (5)**	2200	**84,2%**	900
A5	Minesweeper	91,6% (0)	1800	91,6% (0)	3700	92,6% (0)	12800	91,6% (0)	300	93,7%	1400
A6	SimpleLibrary	**89,0% (10)**	5800	**89,0% (10)**	11700	**89,0% (10)**	40900	**89,0% (7)**	300	**89,0%**	100
A7	Quiz	77,0% (0)	1200	77,0% (0)	2400	77,0% (0)	8300	77,0% (0)	900	77,5%	600
A8	BlackJack	93,2% (0)	4500	93.4% (1)	9000	93,4% (3)	31400	93.4% (1)	6300	**94,0%**	1400
A9	UPM	**72,1% (10)**	7300	**72,1% (10)**	14600	**72,1% (10)**	51000	**72,1% (7)**	700	**72,1%**	300
A10	Jnotepad	**89,5% (10)**	2200	**89,5% (10)**	4400	**89,5% (10)**	15500	**89,5% (10)**	600	**89,5%**	200
A11	Simple Calculator	98,3% (0)	700	98,3% (0)	1400	98,3% (0)	4800	98,3% (0)	700	98,3%	200
A12	Movie Catalog	**88,0% (10)**	1100	**88,0% (10)**	2200	**88,0% (10)**	7700	**88,0% (9)**	600	**88,0%**	600
A13	Bibtex Manager	80,4% (2)	2100	80,4% (2)	4100	80,4% (4)	14400	80,4% (0)	800	**80,5%**	2500
A14	QRCode Generator	**78,5% (10)**	2000	**78,5% (10)**	4100	**78,5% (10)**	14400	**78,5% (10)**	600	**78,5%**	200
A15	Shopping List Manager	**83,0% (10)**	1800	**83,0% (10)**	3600	**83,0% (10)**	12500	**83,0% (10)**	500	**83,0%**	200
A16	HangMan	**95,1% (10)**	800	**95,1% (10)**	1600	**95,1% (10)**	5500	**95,1% (10)**	400	**95,1%**	100
A17	UnitConverter	**51,8% (10)**	700	**51,8% (10)**	1300	**51,8% (10)**	4600	**51,8% (10)**	700	**51,8%**	300
A18	Beagtex	62,5% (0)	4200	63,6% (0)	8400	64,0% (0)	29400	55,4% (0)	49400	**64,4%**	18900
fSat		107/180		111/180		125/180		102/180			

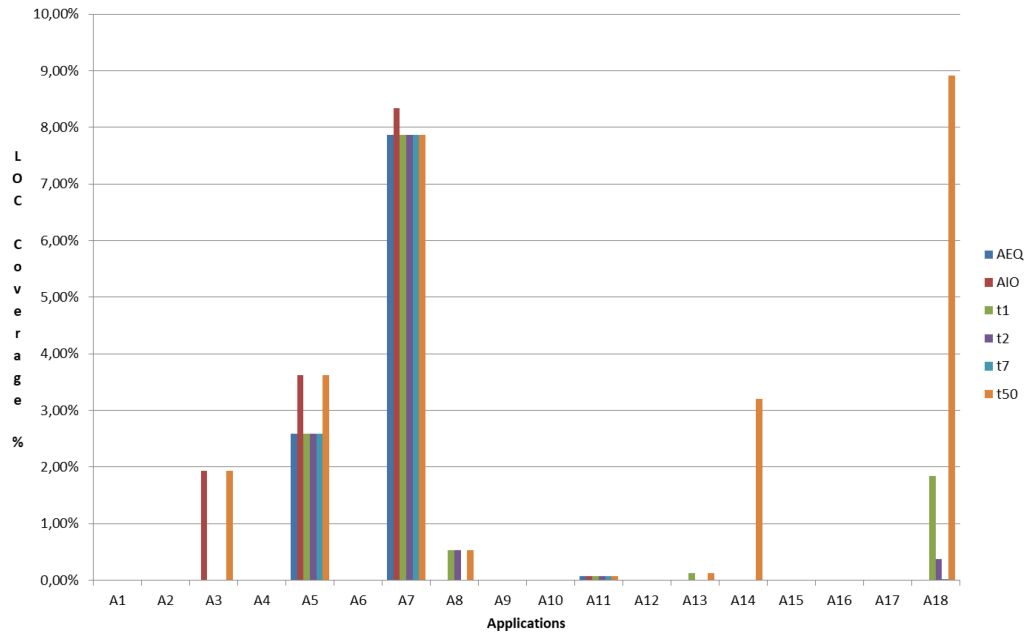


Fig. 5. Differences between saturation levels and coverage levels achieved with different termination criteria.

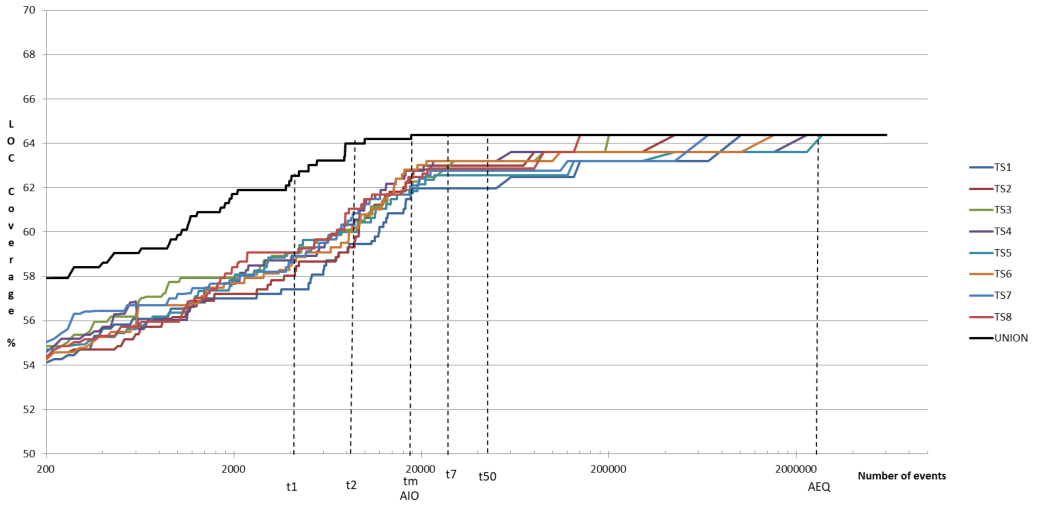


Fig. 6. Differences between saturation level and coverage level achieved with different termination criteria for application A18 in an execution of the proposed testing process.

The figure shows that in the majority of the applications the median of the coverage achieved at the termination of the testing process is generally the same either if we adopt the proposed termination criteria, or the time-based termination ones. More in details, it can be observed that the process adopting the AEQ termination criterion never covers less code than any other process adopting different termination criteria (and consequently, the coverage of processes adopting *tm* is always equivalent to the one of processes using AEQ).

A18 is an example of application where a great difference in coverage between processes adopting AEQ, t1 and t2 can be observed. Figure 6 shows the trend of coverage for A18 obtained by a single execution of the testing process: it is clear that the termination criteria $t1$ and $t2$ provided too early terminations of the process with a significant loss in potential coverage, while $t7$ was more efficient than AEQ and AIO and provided the same coverage.

To compare from a statistical point of view the capabilities of reaching the saturation level of coverage of testing processes adopting the different termination criteria, we considered the total number of experiment repetitions in which each of the executed testing processes reached the saturation level at the different termination points. These values are obtainable from the *fSat* values reported in the last row of Table 11.

Table 12 reports in the first column the proposed null hypotheses and in the latter column the corresponding Fisher test p-values. We can observe that all the null hypotheses regarding AEQ can be rejected, i.e., the testing processes based on termination criterion AEQ are significantly the best ones in terms of capability of reaching the saturation level of coverage. However, none of the null hypotheses regarding AIO can be rejected, i.e., the capability of the processes adopting AIO in reaching the saturation level is not significantly different from the ones of the processes adopting each of the other termination criteria $t1$, $t2$, $t7$, and $t50$.

5.4.8 Data Analysis 6: Evaluating the Termination Points of Processes Adopting AEQ, AIO, t1, t2, t7, t50, and tm. Figure 7 shows the median values of the number of events needed to reach the termination point for each application and for each termination criterion, by using a logarithmic scale.

Table 12. Comparisons between the Capability of Reaching the Saturation for Processes with 8 Testing Sessions: Null Hypotheses and Corresponding p-values Evaluated with the Fisher Test

Null Hypothesis H_0	p-value
Processes adopting AEQ have a worst capability of reaching the saturation with respect to processes adopting t1	$\mathbf{0.3701{*}10^{-6}}$
Processes adopting AEQ have a worst capability of reaching the saturation with respect to processes adopting t2	$\mathbf{0.2983{*}10^{-5}}$
Processes adopting AEQ have a worst capability of reaching the saturation with respect to processes adopting t7	$\mathbf{0.1383{*}10^{-2}}$
Processes adopting AEQ have a worst capability of reaching the saturation with respect to processes adopting t50	$\mathbf{0.2222{*}10^{-7}}$
Processes adopting AIO have the same capability of reaching the saturation of processes adopting t1	0.3279
Processes adopting AIO have the same capability of reaching the saturation of processes adopting t2	0.5846
Processes adopting AIO have the same capability of reaching the saturation of processes adopting t7	0.4320
Processes adopting AIO have the same capability of reaching the saturation of processes adopting t50	0.1305

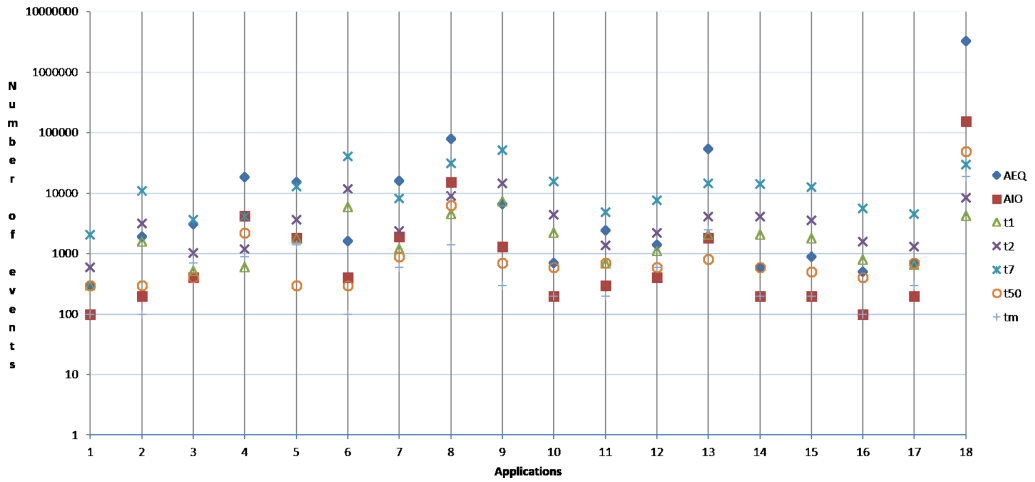


Fig. 7. Comparison of the median values of the termination points obtained adopting different termination criteria.

As shown in the figure, processes using AEQ generally reached the termination point later than the processes adopting the t50 termination criterion or the time-based termination criteria t1 and t2 but earlier than the ones adopting t7. Instead, the processes adopting AIO generally reached the termination point earlier than the ones adopting the time-based termination criteria t1, t2, and t7, whereas there is not a clear difference between the termination points of processes adopting AIO and t50.

For example, for the A9 application, the coverage trend observed in one of the replications of the testing process is the one shown in Figure 8.

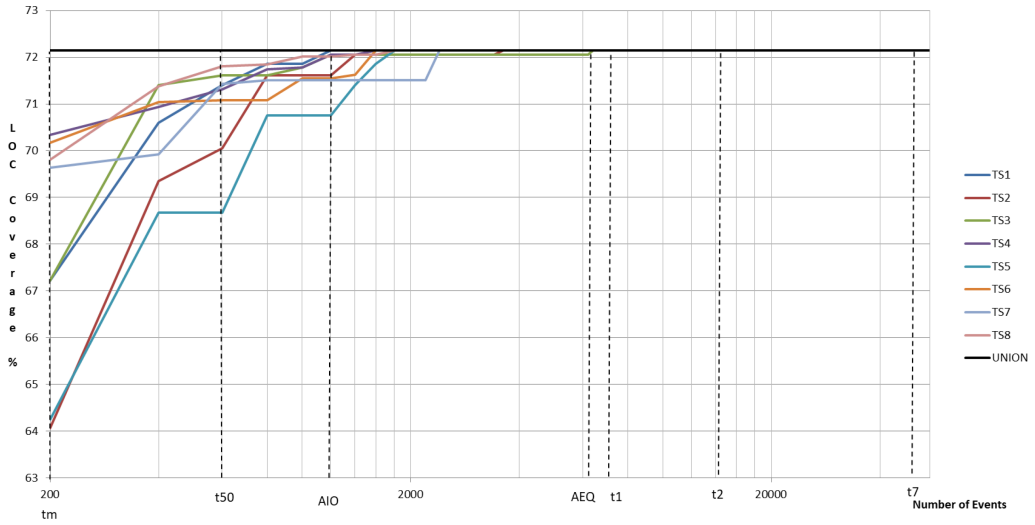


Fig. 8. Code coverage measured for the A9 application.

Table 13. Comparisons between the Termination Points for Processes with Eight Testing Sessions: Null Hypotheses and Corresponding p-values Evaluated with the Mann–Whitney–Wilcoxon Test

Null Hypothesis H_0	p-value
Processes adopting AEQ terminate earlier than processes adopting t1	$\mathbf{0.3491^{*}10^{-6}}$
Processes adopting AEQ terminate earlier than processes adopting t2	**0.0171**
Processes adopting AEQ terminate later than processes adopting t7	$\mathbf{0.5035^{*}10^{-}2}$
Processes adopting AEQ terminate earlier than processes adopting t50	$\mathbf{<0.22{*}10^{-15}}$
Processes adopting AIO terminate later than processes adopting t1	$\mathbf{0.2347^{*}10^{-3}}$
Processes adopting AIO terminate later than processes adopting t2	$\mathbf{0.102{*}10^{-8}}$
Processes adopting AIO terminate later than processes adopting t7	$\mathbf{<0.22 * 10^{-15}}$
Processes adopting AIO terminate earlier than processes adopting t50	$\mathbf{0.8744^{*}10^{-2}}$

In this case, the code coverage rapidly increases with the number of executed events and both the proposed termination criteria stop the process before the three time-based termination criteria $t1$, $t2$, $t7$. In details, AIO is verified after 1,200 events, AEQ after 6,400 events, $t1$ after 7,400 events, $t2$ after 14,600 events and $t7$ after 51,000 events. $t50$, instead, is able to stop the process after the execution of 600 events per session and in this case it represents the faster termination criterion.

To have a statistical evidence of these observations, we compared the termination points of processes adopting AEQ, AIO, t1, t2, t7, and t50 by means of Mann–Whitney–Wilcoxon tests.

Table 13 reports in the first column the proposed null hypotheses and in the latter column the corresponding Mann–Whitney–Wilcoxon test p-values.

All the proposed null hypotheses can be rejected, thus we accepted all the corresponding alternative hypotheses. We can conclude that the termination point of processes adopting AEQ is generally greater than the ones of processes adopting t1, t2, and t50 but less than the ones of processes adopting t7. However, processes adopting AIO terminate earlier than the processes adopting t1, t2, and t7, but later than processes adopting t50.

To have a more accurate idea of the testing time spent by the testing processes adopting the two proposed termination criteria AEQ and AIO, the termination times related to the optimal termination criterion tm were reported in Table 11 and in Figure 7. If we consider all the applications, then we can observe that AEQ needs between 2 times to 40 times the minimum testing time tm, with the exception of the application A18. For example, in the execution shown in Figure 6, the saturation level of coverage was reached at least after 17,600 events (that is tm = 17,600) by the "luckiest" session. However, the AEQ criterion corresponding to the reaching of the same level of coverage for all the 8 sessions was reached after the execution of 2,740,400 events (about 150 times the tm value). Instead, the increase in testing time with respect to tm due to the adoption of AIO is lower for all the applications: in the worst cases, the time increment is between 2 and 6 times.

5.4.9 Answer to RQ3. On the basis of the performed analysis, we could answer to RQ3 and conclude that:

- the capability of testing processes based on timeout of reaching the saturation level at the termination is lower than the one of processes based on AEQ but comparable to the one of processes based on AIO;
- the termination points of processes adopting AEQ are greater than the ones of processes adopting t1, t2, and t50 but less than the ones of processes adopting t7;
- the termination points of processes adopting AIO are smaller than the ones of processes adopting t1, t2, t7, but greater than the ones of processes adopting t50.

5.5 Discussion

The results of this study provided some practical implications for app developers and testers who have the necessity of choosing among different termination criteria of random testing processes.

The study showed that the saturation-based criteria AEQ and AIO represent reliable termination criteria in the considered study context, since they generally allow to reach the saturation effect of random testing, with AEQ showing a stronger capability of reaching the saturation than both AIO and the considered alternative time-based criteria.

The study provided also relevant data about the costs of processes adopting different criteria. The results showed that with AEQ the testing process costs are generally greater than the ones of processes adopting AIO, t1, t2, and t50, but they are less than the ones of processes adopting t7. However, the costs of processes adopting AIO were definitely smaller than the ones of processes adopting AEQ, t1, t2, t7 but greater than the ones of processes adopting t50.

A possible conclusion is that AIO represents a viable option, since it allows to stop a process at a termination point that provides a good tradeoff between reliability of the testing results and testing costs.

In conclusion, we also want to remark something regarding the applicability of the considered saturation-based criteria in the context of testing processes having predefined testing time constraints. In such a context, a combined usage of the AEQ or AIO criterion with a time-based termination criterion, can be still considered advantageous. In all those cases, indeed, in which the saturation-based criteria will provide an earlier termination with respect to the time constraint, the tester will be able to save time and spend the residual testing time in the execution of different testing processes, such as the ones starting from different pre-conditions.

6 SECOND EXPERIMENT ON UNIT TESTING OF JAVA LIBRARIES

In this section, we will present the results of an experiment where we moved to the context of random unit testing of Java applications and libraries and investigated the performance of the

proposed termination criteria in this different context. The aim of this experiment was to extend the validity of the conclusions achieved in the first experiment.

The random testing technique *RTT* considered in this experiment is the one implemented by the random unit testing tool named Randoop.[7] Randoop is a freely available testing tool and it is considered one of the most influential random testing tools in the literature [88]. It has been recently used as a benchmark for the Java Unit Testing Tool Competition at the 2015 Workshop on Search-Based Software Testing [97] and the usefulness of its generated tests has been evaluated by two empirical studies of Ceccato et al. [21, 22]. In addition, it is used in the context of many academic courses and in several industrial processes, as indicated in the Randoop project web page.

The testing technique implemented by Randoop is called Directed Random Testing and is able to generate random test cases by exploiting run-time information from the application under test to avoid the generation of redundant test cases, or test cases with illegal input values. Randoop can be used to generate test cases for Java applications exploiting the JUnit library. It uses by default a fixed set of possible input values that are randomly injected in each method call according to the input type. Randoop can be executed by command line and can be used on applications previously instrumented by Emma.

In this experiment, we focused on Multiple Random Testing processes executing NTS = 8 testing sessions, since the first experiment showed that this choice provides a good tradeoff between capability of reaching the saturation level of coverage and testing costs for both processes adopting AEQ and AIO.

This experiment addressed the following three research questions, which are analogous to the ones of the first experiment:

- RQ4 How does the capability of random testing processes based on AEQ and AIO to reach the saturation level at the termination point vary with the adopted termination criterion?
- RQ5 How do the termination points of the testing processes based on the AEQ and AIO vary with the adopted termination criterion?
- RQ6 How do the termination points and the capability to reach the saturation level of the testing processes adopting AEQ and AIO compare with the ones achieved using other termination criteria presented in the literature?

6.1 Experimental Objects

The objects of the study were 18 sets of classes belonging to 18 distinct Java libraries or applications. Twelve of these sets of classes are included in applications found in the SF 110 corpus[8] proposed by Fraser and Arcuri [39]. The remaining six sets of classes are utility libraries that are used by some of the applications of the same SF110 corpus. The applications have been selected randomly in the corpus by preventively excluding the ones for which Emma was not able to instrument the code and the ones for which Randoop in its standard configuration was not able to provide significant values of coverage (less than 10% of coverage). Table 14 reports some characteristics of the 18 applications involved in this study. For each application the table reports an identifier, the name, and some structural metrics automatically computed by the Emma library during code instrumentation.

The size of the selected objects is similar to the one of the applications involved in the previous experiment, since it varies from 110 to 4,662 LOCs.

[7] https://randoop.github.io/randoop/.
[8] http://www.evosuite.org/experimental-data/sf110/.

Table 14. Some Characteristics of the AUTs Considered in the Second Experiment

Application	Name	Brief description	Number of Classes	Number of Methods	Number of Blocks	LOCs
B1	A4J	API for Amazon.com's XML Web Service	45	522	9773	4662
B2	Aspectj	Aspect-oriented programming extension	68	492	9328	915
B3	Battlecry	Generator of song lyrics	15	130	9550	1831
B4	Calendar	java.util.calendar package	98	562	11600	2814
B5	Classviewer	Application for exploring Java class structures	23	143	5353	996
B6	Dcparseargs	Library to simplify args[] handling	6	21	652	110
B7	Diffi	Binary patch generator	9	54	1425	302
B8	Falselight	Java-based Network Monitoring System	14	32	1189	200
B9	Http-utils	Http utility libraries from Grizzly project at java.net	60	645	21984	4629
B10	Jargs	Command line option parsing utilities	12	52	694	166
B11	Javax	javax.servlet package	31	82	790	254
B12	Jclo	Java Command Line Option parser	8	53	1710	242
B13	Jgaap	Graphical Authorship Attribution Framework	23	143	5353	996
B14	Jipa	Pseudo-Assembly Interpreter	5	36	1488	228
B15	Noen	Framework for dynamic analysis and test	18	71	1353	730
B16	SaxPath	Simple API for XPath	12	257	3115	955
B17	Sugar	Utilities for Java programmers	14	80	2089	407
B18	Tullibee	Components interacting with a brokers' financial-trading Java API	19	204	8857	1912

6.2 Variables

The unique independent variables of this experiment is the termination criterion *TerC* for which the seven values of *AEQ*, *AIO*, *t1*, *t2*, *t7*, *t50*, and *tm* have been considered. The dependent variables are the *terL* level of coverage at the termination reached by the testing process and the testing time *terP* needed to achieve this value. The termination point was measured in terms of testing time per session. The metrics *terP* and *terL* were redefined in the following way:

- $terP$ is the Termination Point of the testing process TP, i.e., the testing time per session at which the termination criterion $TerC$ adopted by TP is verified for the first time.
- $terL$ is the Termination Level indicating the cumulative coverage statement percentage reached by the testing process TP when the termination condition $TerC$ became true for the first time.

6.3 Experimental Materials and Procedure

To execute the experiment, we had to instantiate the testing process described by the Procedure 3 providing values to its five parameters: AUT, AUTPr, RTT, NTS, and TerC.

In this case, the random testing technique *RTT* is the one provided by the Randoop tool. The Randoop version used in the experiments is the 3.0.6 released in October 14, 2016.[9] We defined a test case as the result of the execution of the Randoop tool on the *AUT* for a testing time of 20s. As regards *AUTPr*, no preconditions were set for any of the applications involved in this experiment, while the Number of Testing Sessions *NTS* was fixed to 8.

[9] https://github.com/randoop/randoop/releases.

Table 15. Testing Process Parameters Values

AUT	18 AUTS: {B1, . . . , B18}
AUTPr	No preconditions
RTT	Randoop tool
NTS	8
TerC	{AEQ, AIO, t1, t2, t7, t50, tm}

The testing process parameters values used in the experiment are summarized in Table 15.

As to the experimental procedure, we ran 10 repetitions of each testing process that involved 8 testing sessions. We measured the achieved coverage and evaluated the termination criteria after the execution of each test case (i.e., after 20s of testing time). When the considered termination criterion was met, we evaluated the corresponding termination level *terL* and the termination point *terP*. In this experiment, it was not possible to analytically evaluate if the coverage reached at the termination level was the maximum achievable with the given testing technique, since we had not a complete knowledge about the internals of Randoop and of the applications under test. To approximatively evaluate if the saturation effect was reached at the termination points, we considered an extra testing time of each session that was at least double than the one needed to reach all the considered termination conditions. Therefore, we assumed that the saturation level was reached if no coverage improvements were observed throughout this extra time. This measure provides only an approximation of the saturation level, since we cannot exclude that further increases in coverage could be obtained after a longer testing time.

As to the statistical data analysis, we used the same statistical tests adopted in the former experiment.

The experiments have been carried out on the same PCs involved in previous experiment, all having an Intel I7 processor with a 3GHz clock, 8GB of RAM, equipped with a Solid State Disk (SSD) and using the Windows 7 operative system (64 bits) and Java Runtime Environment 7. The applications under test and the implemented test harness are available at http://reverse.dieti.unina.it/index.php/27-expsaturation.

6.4 Results

Table 16 reports the median values of the termination points *terP* and of the termination levels *terL* measured for each of the 18 AUTs and for the termination criteria AEQ and AIO. The numbers of repetitions that reached the saturation level are reported in parenthesis beside the median value of the coverage level. Median termination levels are reported in bold if they coincide with the maximum obtained coverage level.

The termination levels obtained with AEQ and AIO vary between 15.8% (for B4–Calendar) and 89.9% (for B15–Noen) and are generally lower than the ones obtained in the previous experiment, probably due to the intrinsic limitations of Randoop that is able to generate unit test cases by varying input values only in a predefined set of possible values [89, 90].

The termination times range between few minutes for many of the considered AUTs to more than 10 days for B1 and B9, that are the two most complex libraries considered in this experiment.

6.4.1 Data Analysis 7: Evaluating the Capability of Reaching the Saturation. As to the ability of reaching the saturation level of coverage, we observed that the processes adopting AEQ reached the saturation level of coverage in 175 of 180 cases (there are only five applications for which a single repetition of 10 did not reach this level). However, processes adopting AIO reached the

Table 16. Median Values of the Termination Levels *TerL* (and in Parenthesis the Number of Processes Repetitions of 10 That Achieved the Maximum Coverage Level) and of the Termination Points *TerP* Measured Adopting AEQ and AIO with Eight Testing Sessions

		AEQ		AIO	
	Application	terL	terP (minutes)	terL	terP (minutes)
B1	A4J	**60.9% (9)**	7533	60.7% (2)	55.7
B2	Aspectj	**20.9% (10)**	5.7	**20.9% (10)**	1.3
B3	Battlecry	**49.0% (10)**	42.7	48.5% (2)	2.3
B4	Calendar	**15.8% (10)**	7.3	**15.8% (8)**	0.7
B5	Classviewer	**34.6% (10)**	22.8	**34.6% (8)**	2.3
B6	Dcparseargs	**67.0% (10)**	1.3	**67.0% (10)**	0.3
B7	Diffi	**72.5% (10)**	3.2	**72.5% (10)**	0.7
B8	Falselight	**24.6% (10)**	1.7	**24.6% (10)**	0.3
B9	http-utils	**24.7% (9)**	11783	**24.7% (5)**	577.7
B10	Jargs	**57.5% (10)**	345.3	**57.5% (7)**	20.3
B11	Javax	**79.9% (10)**	1	**79.9% (10)**	0.3
B12	jclo	**46.3% (9)**	145.7	45.5% (1)	0.3
B13	jgaap	**50.3% (10)**	13.8	**50.3% (8)**	2.5
B14	Jipa	**46.8% (10)**	5.7	**46.8% (9)**	0.7
B15	noen	**89.9% (10)**	3	**89.9% (10)**	0.7
B16	SaxPath	**54.9% (10)**	55.5	**54.9% (8)**	7.7
B17	Sugar	**43.3% (9)**	423.7	**43.3% (5)**	63.3
B18	Tullibee	**27.6% (9)**	8377.3	**27.6% (5)**	1077.7
fSat		175/180		128/180	

maximum level in 128 cases of 180. In particular, they reached the saturation level in the majority of the repetitions for 15 of 18 applications. The unique exceptions are B1, B3, and B12 (with differences in coverage with respect to the saturation level between 1 and 10 LOCs).

To evaluate from a statistical point of view if processes adopting AEQ are more capable of reaching the saturation level of coverage than the ones adopting AIO, we evaluated the following null hypothesis by means of the Fisher exact test:

H_0: Processes adopting AEQ have a worst capability of reaching the saturation with respect to processes adopting AIO.

Since we measured a p-value of $0.1769 * 10^{-11}$, we rejected this null hypothesis and accepted that the processes adopting AEQ have a better capability of reaching the saturation with respect to the ones having AIO as the termination criterion.

This result provides the answer to the research question RQ4 and is coherent to the answer obtained by the research question RQ1 when eight testing sessions were considered, reported in Section 5.4.3.

6.4.2 Data Analysis 8: Comparing the Testing Times per Session of AEQ and AIO. The median values reported in Table 16 shows how the processes adopting AEQ appear to be more expensive than the ones based on AIO in terms of testing time per session.

To have a statistical confirmation of this observation, we have evaluated the following null hypothesis by applying the Mann–Whitney–Wilcoxon test:

H_0: Processes adopting AEQ terminate earlier than processes adopting AIO.

Table 17. Median Values of the Termination Levels *TerL* and of the Corresponding Termination Points *TerP* Obtained with the Termination Criteria *t1*, *t2*, *t7*, *t50*, and *tm*

		*t*1		*t*2		*t*7		*t*50		*tm*	
		terL	terP	terL	terP	terL	terP	terL	terP	terL	terP
B1	A4J	60.3% (0)	45	60.5% (2)	90	60.7% (4)	315	60,7% (0)	8,7	**60.9%**	351,3
B2	Aspectj	**20.9% (10)**	68	**20.9% (10)**	136	**20.9% (10)**	476	**20,9% (10)**	1,0	**20.9%**	0,3
B3	Battlecry	**49.0% (7)**	15	**49.0% (9)**	30	**49.0% (10)**	105	**49,0% (5)**	3,3	**49.0%**	1,3
B4	Calendar	**15.8% (10)**	98	**15.8% (10)**	196	**15.8% (10)**	686	**15,8% (9)**	1,3	**15.8%**	0,3
B5	Classviewer	**34.6% (10)**	23	**34.6% (10)**	46	**34.6% (10)**	161	**34.6% (9)**	7,0	**34.6%**	22,3
B6	dcparseargs	**67.0% (10)**	6	**67.0% (10)**	12	**67.0% (10)**	42	**67,0% (10)**	1,0	**67.0%**	0,7
B7	Diffi	**72.5% (10)**	9	**72.5% (10)**	18	**72.5% (10)**	63	**72,5% (10)**	3,3	**72.5%**	1,3
B8	Falselight	**24.3% (10)**	14	**24.3% (10)**	28	**24.3% (10)**	98	**24,3% (10)**	1,0	**24.6%**	0,3
B9	http-utils	24.3% (0)	60	24.3% (1)	120	24.5% (4)	420	24,3% (0)	25,7	**24.7%**	147,7
B10	Jargs	**57.5% (5)**	12	**57.5% (7)**	24	**57.5% (9)**	84	56,9% (0)	1,0	**57.5%**	4,7
B11	Javax	**79.9% (10)**	31	**79.9% (10)**	62	**79.9% (10)**	217	**79,9% (10)**	1,0	**79.9%**	0,3
B12	Jclo	39.1% (0)	8	45.2% (0)	16	**46.3% (5)**	56	35,0% (0)	2,3	**46.3%**	42.3
B13	jgaap	**50.3% (10)**	23	**50.3% (10)**	46	**50.3% (10)**	161	**50,3% (6)**	1,7	**50.3%**	0,7
B14	Jipa	**46.8% (10)**	5	**46.8% (10)**	10	**46.8% (10)**	35	**46,8% (10)**	1,0	**46.8%**	0,3
B15	noen	**89.9% (10)**	18	**89.9% (10)**	36	**89.9% (10)**	126	**89,9% (10)**	1,0	**89.9%**	0,3
B16	SaxPath	**54.9% (9)**	12	**54.9% (10)**	24	**54.9% (10)**	84	54.7% (4)	3,7	**54.9%**	6,7
B17	Sugar	43.0% (1)	14	43.1% (3)	28	**43.3% (6)**	98	43,0% (0)	1,7	**43.3%**	0,7
B18	Tullibee	27.5% (0)	19	27.5% (0)	38	27.5% (1)	133	27,4% (0)	4,6	**27.6%**	527.3
fSat		122/180		132/180		145/180		103/180			

We measured a p-value that is under $0.22 * 10^{-15}$, so we rejected the null hypothesis and accepted the alternative hypothesis stating that processes adopting AEQ terminate later than the ones adopting AIO.

This response to the research question RQ5 is coherent with the one obtained for RQ2 in the first experiment when eight testing sessions were considered and reported in Section 5.4.6.

6.4.3 Data Analysis 9: Evaluating the Termination Levels of Processes Adopting AEQ, AIO, t1, t2, t7, t50, and tm. To answer to RQ6, we compared the performance of the processes adopting the two proposed termination criteria AEQ and AIO with the ones of the same processes adopting the five termination criteria $t1$, $t2$, $t7$, $t50$, and tm.

Table 17 reports the median values of the termination levels *terL* and the corresponding termination points *terP* measured for each of the five selected criteria $t1$, $t2$, $t7$, $t50$, and tm. The numbers of processes repetitions (of 10) that reached the saturation level of coverage are reported in parenthesis near the termination levels. The median values of the termination levels that are equal to the saturation levels are highlighted in bold. The last row of the table reports the frequency of Saturation *fSat* evaluated as the fraction between the total number of executed testing processes that reached the saturation level of coverage and the total number of executed testing processes.

To highlight the differences in coverage between the different termination criteria, Figure 9 shows the lacks of coverage with respect to the saturation level for each of the five considered criteria evaluated on the basis of the measured median values of coverage.

It can be observed that the processes adopting the three time-based termination criteria ($t1$, $t2$, and $t7$) were not able to always reach the saturation level of coverage. In details, the saturation

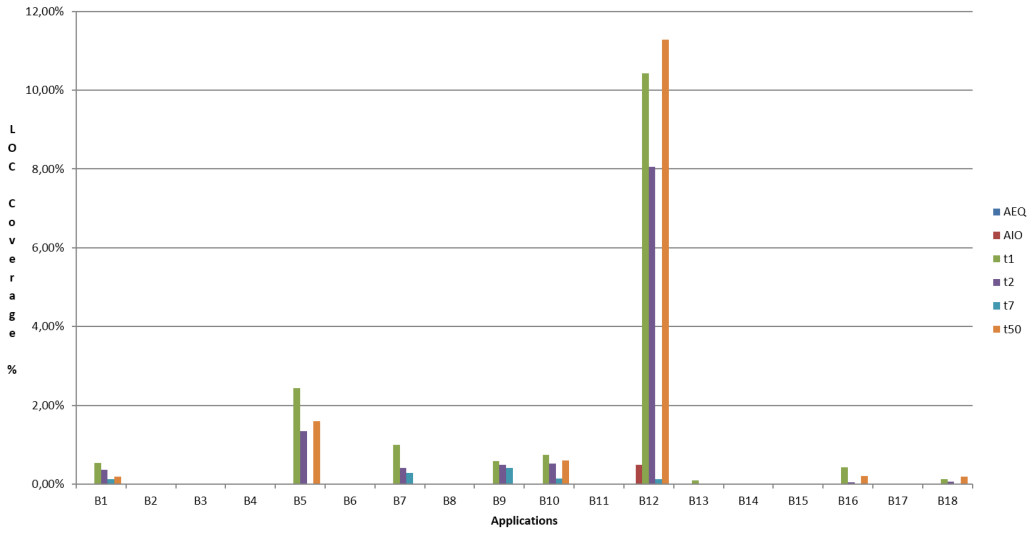


Fig. 9. Differences between saturation level and the median values of the coverage level achieved with each termination criterion.

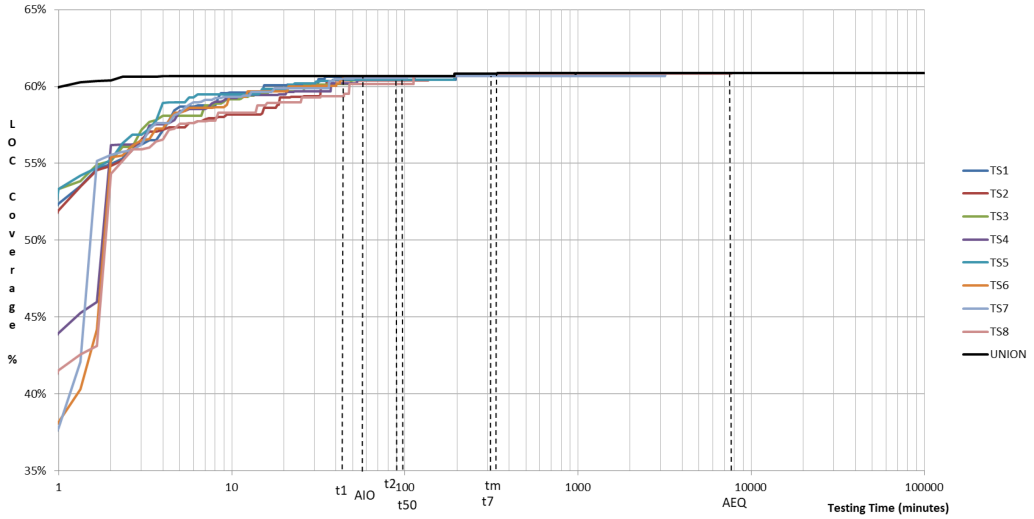


Fig. 10. Coverage of the B1 application with different termination criteria.

was reached in the majority of cases for 13 of 18 applications by processes adopting $t1$ and $t2$, for 15 applications by processing adopting $t7$ and for 11 applications by the ones adopting $t50$. In particular, the largest lack in coverage can be observed for B5 and B12, for which $t1$, $t2$, and $t50$ stopped the testing process too early, achieving a lack of coverage with respect to the saturation level that is greater than 1% for B5 and greater than 8% for B12.

Figure 10 shows an example of the observed coverage trends. This figure reports the trend of the coverage of the eight considered testing sessions (from TS1 to TS8) and of their cumulative coverage (Union) for one of the repetitions of the testing process on the application B1. The figure shows how the coverage increases with the testing time per session (reported in logarithmic scale)

Table 18. Comparisons between the Capability of Reaching the Saturation for Processes with Eight Testing Sessions: Null Hypotheses and Corresponding p-values Evaluated with the Fisher Test

Null Hypothesis H_0	p-value
Processes adopting AEQ have a worst capability of reaching the saturation with respect to processes adopting t1	$\mathbf{0.7795{*}10^{-14}}$
Processes adopting AEQ have a worst capability of reaching the saturation with respect to processes adopting t2	$\mathbf{0.1836{*}10^{-10}}$
Processes adopting AEQ have a worst capability of reaching the saturation with respect to processes adopting t7	$\mathbf{0.1754{*}10^{-6}}$
Processes adopting AEQ have a worst capability of reaching the saturation with respect to processes adopting t50	$\mathbf{0.5536{*}10^{-21}}$
Processes adopting AIO have the same capability of reaching the saturation of processes adopting t1	0.5673
Processes adopting AIO have the same capability of reaching the saturation of processes adopting t2	0.7242
Processes adopting AIO have a better capability of reaching the saturation with respect to processes adopting t7	**0.0242**
Processes adopting AIO have a worst capability of reaching the saturation with respect to processes adopting t50	**0.0041**

and depicts the different termination points obtained with the different termination criteria. It can be observed that all the time-based termination criteria and the $t50$ criterion provided just a slightly lower coverage with respect to AEQ but with a saving in terms of testing time.

To have statistical comparisons between the different capabilities of reaching the saturation of the testing processes adopting AEQ and AIO against the ones adopting t1, t2, t7, and t50, we formulated the set of null hypotheses reported in the first column of Table 18 and evaluated them by means of the Fisher exact test. The second column of Table 18 reports the measured p-values.

All the null hypotheses involving AEQ can be rejected; therefore, we can conclude that processes adopting the termination criterion AEQ have a better capability of reaching the saturation level at the termination point with respect to all the other considered termination criteria. This conclusion confirmed the one obtained by examining the result of the first experiment and reported in Section 5.4.9.

However, only the latter two null hypotheses involving AIO can be rejected. They show that the processes based on the termination criteria AIO are more capable than the ones adopting t50 in reaching the saturation level at the termination point (the p-value is 0.0041) but less capable than the ones adopting t7 (the p-value is 0.0242). No statistical evidences, instead, can be obtained by the comparisons between the capabilities of reaching the saturation of processes adopting AIO, t1, and t2 (the hypotheses that they have the same capabilities cannot be rejected). These results are quite similar to the ones obtained in the first experiment (see Section 5.4.8), in which no statistical evidences were obtained neither in the comparisons of AIO with t1 or t2, nor in the ones with t7 and t50.

6.4.4 Data Analysis 10: Evaluating the Termination Points of Processes Adopting AEQ, AIO, t1, t2, t7, t50, and tm. As regards the comparisons between the testing times of the executed processes, Figure 11 shows the median values of the testing times per session needed to reach the termination point for each application and for each termination criterion, by using a logarithmic scale.

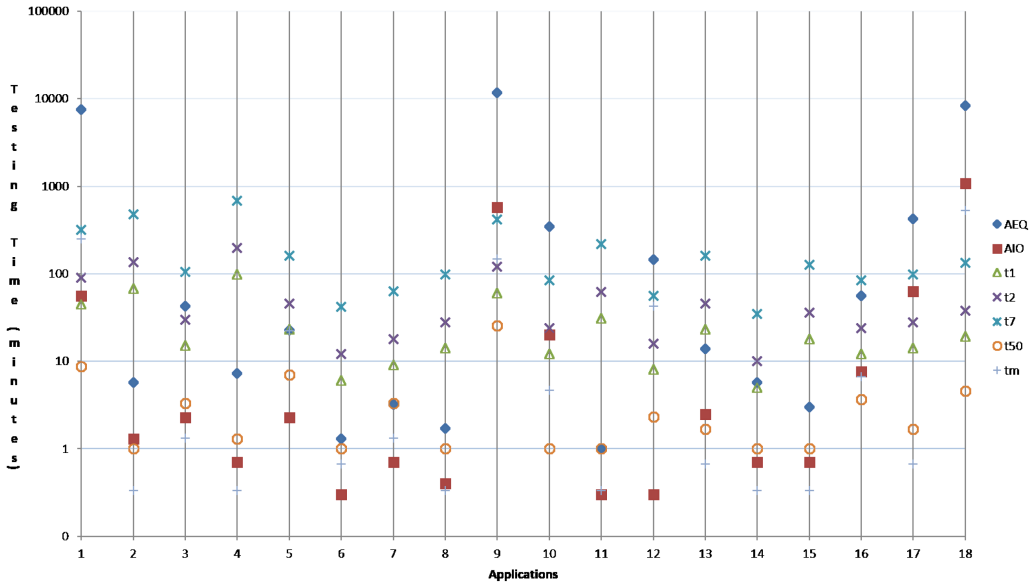


Fig. 11. Comparison of the termination points obtained adopting different termination criteria.

Table 19. Comparisons between the Termination Points for Processes with Eight Testing Sessions: Null Hypotheses and Corresponding p-values Evaluated with the Mann–Whitney–Wilcoxon Test

Null Hypothesis H_0	p-value
Processes adopting AEQ terminate earlier than processes adopting t1	$\mathbf{0.4611{*}10^{-6}}$
Processes adopting AEQ terminate earlier than processes adopting t2	**0.0016**
Processes adopting AEQ have the same termination points of processes adopting t7	0.2773
Processes adopting AEQ terminate earlier than processes adopting t50	$\mathbf{<0.22{*}10^{-15}}$
Processes adopting AIO terminate later than processes adopting t1	**0.0011**
Processes adopting AIO terminate later than processes adopting t2	$\mathbf{0.7933{*}10^{-8}}$
Processes adopting AIO terminate later than processes adopting t7	$\mathbf{0.3192{*}10^{-14}}$
Processes adopting AIO terminate earlier than processes adopting t50	$\mathbf{0.3256{*}10^{-5}}$

In this experiment, the processes adopting AEQ and t7 were the ones having the greatest values of the termination points. The processes adopting the termination criteria AIO and $t50$, instead, were the ones having the smallest values of termination points.

By comparing the average values of the termination time of the proposed termination criteria AEQ and AIO with the average value of the minimum time needed to reach the saturation level (tm), it is possible to observe how the testing time spent to reach the AEQ condition was between 2 times and 150 times the time needed to reach tm, while the time spent to reach the AIO condition was up to 13 times the one spent to reach tm.

To have a statistical evidence of these observations, we compared the termination points of processes adopting AEQ, AIO, t1, t2, t7, and t50 by means of Mann–Whitney–Wilcoxon tests as in the first experiment.

Table 13 reports in the first column the proposed null hypotheses and in the latter column the corresponding Mann–Whitney–Wilcoxon test p-values.

Three of the four null hypotheses regarding AEQ can be rejected. The corresponding alternative hypotheses state that processes adopting AEQ generally terminate later than processes adopting t1, t2, or t50. However, the termination points measured for processes adopting AEQ and t7 are not very different (the corresponding null hypothesis has a p-value of 0.2773 and cannot be rejected). Except for this last observation, the conclusions of this analysis coincided with the ones obtained in the first experiment (see Section 5.4.8).

As regards AIO, all the null hypotheses can be rejected, so we can conclude that processes involving AIO generally terminated earlier than processes involving t1, t2, or t7 but later than processes involving t50. These results are coherent with the ones obtained in the first experiment.

6.4.5 Answer to RQ6. On the basis of the performed analysis, we could answer to RQ6 and conclude that:

- the capability of testing processes $t1$, $t2$, $t7$, and $t50$ of reaching the saturation level at the termination is lower than the one of processes based on AEQ. Processes based on AIO have a better capability than the ones based on t50 in reaching the saturation level but a worst capability than the ones based on $t7$;
- the termination points of processes adopting AEQ are greater than the ones of processes adopting t1, t2, and t50;
- the termination points of processes adopting AIO are smaller than the ones of processes adopting t1, t2, t7, and t50.

6.5 Discussion

The results of this second experiment carried out in the context of unit testing of Java applications and libraries by the Randoop tool were substantially coherent with the ones obtained in the first experiment, conducted in the context of GUI testing of Java desktop applications by the Java Random Ripper tool.

This result indicates that the proposed termination criteria are still applicable in different testing contexts and the performance of the saturation-based criteria does not vary with the considered context and the implemented random testing technique.

This experimental result makes us confident about the general applicability of these termination criteria at least in the two considered contexts. Of course, to further extend the validity of these conclusions, new experiments targeting different testing contexts and random testing tools will have to be designed and carried out.

7 THREATS TO VALIDITY

This subsection discusses some threats that could have affected the results of the experiments and their generalization, according to the guidelines proposed by Wohlin et al. [109].

Threats to Conclusion Validity. There are threats to conclusion validity due to randomness and to measure accuracy. Both the testing processes presented in Sections 5 and 6 were repeated 10 times for each of the 36 AUTs, to mitigate the threats to the conclusion validity due to the possible influence of randomness on the measured results. Thus, the statistical analysis considered to provide answers to the proposed research questions were carried out on sets including 180 observations (10 for each of the 18 objects involved in each experiment).

As regards the measure accuracy of the termination points, it is threatened by the experimental procedure that we adopted for measuring them. However, since we evaluated the reaching of the termination points after exactly 100 events in the experiments involving Java GUI testing, or each 20s in the ones involving Java Unit testing, the inaccuracy of our measure can be considered

negligible with respect to the time usually devoted to testing activities (e.g., in Reference [40] and Reference [11] 2 minutes are considered as the minimum testing time for a class).

Finally, the saturation level in the second experiment has been approximated to the cumulative coverage reached after a testing time that is at least the double of the time needed to reach all the termination criteria. This represents an approximation but is coherent with the fundamental property of the saturation effect, according to which no increases in coverage are generally observable after the reaching of the saturation level.

Threats to Internal Validity. There are some conditions that can affect the dependent variables of the experiment without the researcher's knowledge. To reduce the risk of influences due to Java Virtual Machines characteristics and to their aging, all the testing sessions were executed on the same PCs with the same Java Virtual Machine version. A new instance of the Java Virtual Machine was created after the execution of each sequence of test cases (consisting of 200 events in the first experiment and 20s of test cases in the second experiment). To avoid interactions with the file system, the dependencies on file system objects and standard dialog windows were removed by the source code of the applications under test in the first experiment and substituted by stubs (e.g., the ones allowing to explore the file system for opening/saving of files). Finally, the preconditions were reset after the execution of each test case to avoid interactions between the execution of consecutive test cases. The applications objects of the first experiment have no interactions with remote resources and are single threaded, without any time-dependent behavior. As regards the sets of classes selected in the second experiment, they are all related to the SF 110 corpus proposed by Fraser and Arcuri [39]. These classes were tested without any modification to their source codes.

Threats to Construct Validity. They concern the appropriateness of the measures for capturing the dependent variables. The testing adequacy was measured in terms of the LOCs covered at the termination level. LOC coverage has been selected, since it is an indicator of test adequacy largely used in the literature and that can be automatically and accurately measured. The termination points were evaluated in terms of the number of events executed by the random testing process or in terms of testing time. The first metric was considered in the first experiment, since it is independent on the specific characteristics of the testing harness and of the machines involved in the experiment. The second metric was the one evaluated in the second experiment, so the observations related to the difference in testing time between different testing processes can be influenced by the characteristics of the testing environment. To limit this dependency, all the testing processes were executed on machines having the same hardware characteristics.

Threats to External Validity. These threats limit the ability to generalize the results of the experiment. A threat is given by the selection of the experimental objects involved in the study. Third-party publicly available applications were selected instead of applications produced by the authors or by students in both the experiments. They belong to different domains and some of them have been already used in other empirical studies on testing, such as A9 in the first experiment (cf. Reference [78]) and all the applications selected for the second experiment (cf. Reference [39]). To achieve a sufficient degree of generalization, 36 different applications and libraries were considered. They are all quite small in size, so a residual threat to external validity is related to the scalability of the approach when applied to testing processes executed on medium and large applications.

To mitigate the threat related to the generalization of the conclusions, the two experiments were carried out on different instances of the testing process 3, involving different testing activities (GUI-based testing and unit testing), testing techniques and tools (Java Random Ripper and Randoop). The results of these two experiments confirmed and extended the ones previously obtained by

the experiment presented in Reference [3] and involving a random testing process carried out on Android applications.

8 CONCLUSIONS AND FUTURE WORK

Random testing is a very popular testing technique in many contexts. Valuable points of strength of random testing techniques are represented by their simplicity of use and the ability to reach a good level of test adequacy using fully automatic test cases. An open issue with random testing processes is the definition of a stopping criterion that represents an optimal tradeoff between test adequacy and cost.

A possible solution to the termination problem can be defined using the *saturation effect* of random testing. This solution requires the independent execution of several random testing sessions and the comparisons between the sets of testing targets covered by each of them to automatically assess when the saturation effect is verified.

In this article, we have presented the theoretical foundation of the saturation effect in random testing processes, the relationship between the saturation effect and the predictability property of random testing, and the consequent theoretical soundness of termination criteria based on the saturation effect. We have presented the Multiple Random Testing Process that is based on the execution of several independent random testing sessions and on the adoption of saturation-based termination criteria called AEQ and AIO. The termination criteria can be applied without the need to formulate any hypotheses related to the expected trend of coverage or to the expected termination point. They can be applied, too, without any knowledge about the application under test.

We have investigated the properties of two saturation-based termination criteria of random testing, AIO and AEQ and compared their performance against the ones of other criteria proposed in the literature by two experiments involving 36 different Java applications. The experimental results have shown that both the termination criteria are generally able to reach and correctly identify the saturation level of code coverage, leaving uncovered only in few cases small portions of code characterized by a very low probability of being executed by the selected random technique. In particular, testing processes terminating at the reaching of the AEQ condition have been generally able to provide slightly higher levels of code coverage, while the adoption of the AIO termination criterion allowed to have more efficient testing processes.

We have also experimentally studied the influence of the number of executed testing sessions on these testing processes. The experimental results have shown that the selection of eight testing sessions represents a good solution, both in terms of code coverage and in terms of testing time. Testing processes adopting the AEQ termination criterion are generally able to reach higher code coverage than processes adopting either a time termination criterion based on the number of classes of the tested applications, or a termination criterion based on the evaluation of the number of test cases executed without any further increase in coverage. However, processes adopting the AIO criterion have shown better efficiency than the other ones.

On the basis of the obtained experimental results, we conclude that the proposed saturation-based termination criteria represent two feasible solutions to the problem of random testing termination. They can be used in conjunction, or with other time-based termination criteria, to exploit the better efficiency achievable by AIO, the better test adequacy obtainable by AEQ, and respecting time constraints imposed by time-based termination criteria.

Using the proposed termination criteria to stop the random testing processes gives the tester the possibility not only to save computational resources, but also to convoy the saved resources toward further random testing process executions, starting from different testing pre-conditions and using different settings.

As future work, we plan to implement and experiment composite testing processes involving the dynamic selection of the number of testing sessions and the combination between different termination criteria. In addition, we intend to study the usefulness of these processes in random testing of large size applications. The problem of estimating the remaining time needed to reach the termination condition will be investigated, to allow the tester to dynamically choose an alternative termination criterion in cases in which the reaching of the considered termination condition requires an excessive testing time.

APPENDIX

A OTHER RELEVANT PROPERTIES OF THE RANDOM TESTING PROCESS

In this appendix some interesting properties of the Random Testing Process 2 are defined and demonstrated. These properties has shown their usefulness in some aspects of the discussion of the results of the experiments.

A.1 Property 1

It is possible to estimate the number of iterations of the Testing Process 2 needed to cover each testing target t_i if the probability to p_i to cover this target with a test case is known.

The number of test cases k^* needed to achieve a probability to cover t_i that is at least equal to a fixed value of probability p^* (cfr. Menzies and Cukic [81]) is

$$k^* = \frac{log(1-p^*)}{log(1-p_i)}. \tag{19}$$

This formula can be obtained by fixing $k = k^*$ in Formula 5:

$$p^* = 1-(1-p_i)^{k^*}, \tag{20}$$

$$1-p^* = (1-p_i)^{k^*}, \tag{21}$$

$$k^* = \frac{log(1-p^*)}{log(1-p_i)}. \tag{22}$$

For example, the number k^* of test cases needed to achieve a probability $p^* = 99.99\%$ to cover a testing target t_i having probability $p_i = 10^{-4}$ is

$$k^* = \frac{log(1-0.9999)}{log(1-10^{-4})} \cong 92,099 \text{ test cases}. \tag{23}$$

A.2 Property 2

The second property that we report directly correlates k^* and p_i. For $p_i << 1$, we can observe that:

$$\lim_{p_i->0} p_i \cdot k^* = \lim_{p_i->0} \cdot \frac{p_i \cdot log(1-p^*)}{log(1-p_i)} \tag{24}$$

$$= log(1-p^*) \cdot \lim_{p_i->0} \frac{p_i}{log(1-p_i)}. \tag{25}$$

Since the value of the above notable limit is one, it is possible to conclude that:

$$\lim_{p_i->0} p_i \cdot k^* = log(1-p^*). \tag{26}$$

In other words, for little values of p_i (testing targets that are rarely covered by random test cases), the product between p_i and k^* tends to be a constant, i.e., the number of test cases to be executed to reach a given probability p^* to cover the testing target t_i is inversely proportional to the probability p_i to reach the testing target with a single test case.

For example, if we have achieved a given probability $p^* = 99.99\%$ to cover a testing target t_i having probability $p_i = 10^{-4}$, and we want to reach the same probability for another target t_j having a probability that is half the one of t_i ($p_j = p_i/2 = 0.5 * 10^{-4}$), then we have to double the number of executed test cases.

REFERENCES

[1] 2015. ISO/IEC/IEEE international standard–Software and systems engineering–Software testing–Part 4: Test techniques. *ISO/IEC/IEEE 29119-4:2015* (Dec. 2015), 1–149. DOI: https://doi.org/10.1109/IEEESTD.2015.7346375

[2] Moataz A. Ahmed and Irman Hermadi. 2008. GA-based multiple paths test data generator. *Comput. Operat. Res.* 35, 10 (Oct. 2008), 3107–3124. DOI: https://doi.org/10.1016/j.cor.2007.01.012

[3] Domenico Amalfitano, Nicola Amatucci, Anna Rita Fasolino, Porfirio Tramontana, Emily Kowalczyk, and Atif M. Memon. 2015. Exploiting the saturation effect in automatic random testing of android applications. In *Proceedings of the 2nd ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft'15)*. IEEE Press, Los Alamitos, CA, 33–43. http://dl.acm.org/citation.cfm?id=2825041.2825046

[4] Domenico Amalfitano, Anna Rita Fasolino, Salvatore De Carmine, Atif Memon, and Porfirio Tramontana. 2012. Using GUI ripping for automated testing of android applications. In *Proceedings of the 27th IEEE international Conference on Automated Software Engineering (ASE'12)*. IEEE Computer Society, Los Alamitos, CA.

[5] D. Amalfitano, V. Riccio, N. Amatucci, V. D. Simone, and A. R. Fasolino. 2019. Combining automated GUI exploration of Android apps with capture and replay through machine learning. *Inf. Softw. Technol.* 105 (2019), 95–116. DOI: https://doi.org/10.1016/j.infsof.2018.08.007

[6] J. H. Andrews, A. Groce, M. Weston, and R. G. Xu. 2008. Random test run length and effectiveness. In *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE'08)*. 19–28. DOI: https://doi.org/10.1109/ASE.2008.12

[7] A. Arcuri. 2012. A theoretical and empirical analysis of the role of test sequence length in software testing for structural coverage. *IEEE Trans. Softw. Eng.* 38, 3 (May 2012), 497–519. DOI: https://doi.org/10.1109/TSE.2011.44

[8] Andrea Arcuri and Lionel Briand. 2011. Adaptive random testing: An illusion of effectiveness? In *Proceedings of the 2011 International Symposium on Software Testing and Analysis (ISSTA'11)*. ACM, New York, NY, 265–275. DOI: https://doi.org/10.1145/2001420.2001452

[9] A. Arcuri and L. Briand. 2012. Formal analysis of the probability of interaction fault detection using random testing. *IEEE Trans. Softw. Eng.* 38, 5 (Sep. 2012), 1088–1099. DOI: https://doi.org/10.1109/TSE.2011.85

[10] Andrea Arcuri and Lionel Briand. 2014. A Hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. *Softw. Test. Verif. Reliab.* 24, 3 (May 2014), 219–250. DOI: https://doi.org/10.1002/stvr.1486

[11] Andrea Arcuri and Gordon Fraser. 2013. Parameter tuning or default values? An empirical investigation in search-based software engineering. *Empir. Softw. Eng.* 18, 3 (2013), 594–623. DOI: https://doi.org/10.1007/s10664-013-9249-9

[12] Andrea Arcuri, Muhammad Zohaib Iqbal, and Lionel Briand. 2012. Random testing: Theoretical results and practical implications. *IEEE Trans. Softw. Eng.* 38, 2 (Mar. 2012), 258–277. DOI: https://doi.org/10.1109/TSE.2011.121

[13] Thomas R. Arnold. 1998. *Visual Test 6 Bible* (1st ed.). IDG Books Worldwide, Inc., Foster City, CA.

[14] Sebastian Bauersfeld and Tanja E. J. Vos. 2012. GUITest: A Java library for fully automated GUI robustness testing. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE'12)*. ACM, New York, NY, 330–333. DOI: https://doi.org/10.1145/2351676.2351739

[15] Marcel Böhme and Soumya Paul. 2014. On the efficiency of automated testing. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'14)*. ACM, New York, NY, 632–642. DOI: https://doi.org/10.1145/2635868.2635923

[16] L. Briand and D. Pfahl. 1999. Using simulation for assessing the real impact of test coverage on defect coverage. In *Proceedings of the 10th International Symposium on Software Reliability Engineering 1999*. 148–157. DOI: https://doi.org/10.1109/ISSRE.1999.809319

[17] P. M. S. Bueno and M. Jino. 2000. Identification of potentially infeasible program paths by monitoring the search for test data. In *Proceedings of the 15th IEEE International Conference on Automated Software Engineering (ASE'00)*. 209–218. DOI: https://doi.org/10.1109/ASE.2000.873665

[18] Paulo Marcos Siqueira Bueno and Mario Jino. 2002. Automatic test data generation for program paths using genetic algorithms. *Int. J. Softw. Eng. Knowl. Eng.* 12, 06 (2002), 691–709. DOI: https://doi.org/10.1142/S0218194002001074

[19] M. Böhme and S. Paul. 2016. A probabilistic analysis of the efficiency of automated software testing. *IEEE Trans. Softw. Eng.* 42, 4 (Apr. 2016), 345–360. DOI: https://doi.org/10.1109/TSE.2015.2487274

[20] Xia Cai and Michael R. Lyu. 2005. The effect of code coverage on fault detection under different testing profiles. In *Proceedings of the 1st International Workshop on Advances in Model-based Testing (A-MOST'05)*. ACM, New York, NY, 1–7. DOI: https://doi.org/10.1145/1082983.1083288

[21] Mariano Ceccato, Alessandro Marchetto, Leonardo Mariani, Cu D. Nguyen, and Paolo Tonella. 2012. An empirical study about the effectiveness of debugging when random test cases are used. In *Proceedings of the 34th International Conference on Software Engineering (ICSE'12)*. IEEE Press, Los Alamitos, CA, 452–462. http://dl.acm.org/citation.cfm?id=2337223.2337277

[22] Mariano Ceccato, Alessandro Marchetto, Leonardo Mariani, Cu D. Nguyen, and Paolo Tonella. 2015. Do automatically generated test cases make debugging easier? An experimental assessment of debugging effectiveness and efficiency. *ACM Trans. Softw. Eng. Methodol.* 25, 1, Article 5 (Dec. 2015), 38 pages. DOI: https://doi.org/10.1145/2768829

[23] Wen-Kui Chang and Shuen-Lin Jeng. 2007. Practical stopping criteria for validating safety-critical software by estimating impartial reliability. *Appl. Math. Model.* 31, 7 (2007), 1411–1424. DOI: https://doi.org/10.1016/j.apm.2006.04.017

[24] Mei-Hwa Chen, M. R. Lyu, and W. E. Wong. 1996. An empirical study of the correlation between code coverage and reliability estimation. In *Proceedings of the 3rd International Symposium on Software Metrics: From Measurement to Empirical Results (METRICS'96)*. IEEE Computer Society, Los Alamitos, CA, 133–. http://dl.acm.org/citation.cfm?id=525586.823872

[25] Mei-Hwa Chen, M. R. Lyu, and W. E. Wong. 1997. Incorporating code coverage in the reliability estimation for fault-tolerant software. In *Proceedings of the 16th Symposium on Reliable Distributed Systems 1997*. 45–52. DOI: https://doi.org/10.1109/RELDIS.1997.632796

[26] Sanping Chen and S. Mills. 1996. A binary Markov process model for random testing. *IEEE Trans. Softw. Eng.* 22, 3 (Mar. 1996), 218–223. DOI: https://doi.org/10.1109/32.489081

[27] T. Y. Chen, F. C. Kuo, R. G. Merkel, and T. H. Tse. 2010. Adaptive random testing: The ART of test case diversity. *J. Syst. Softw.* 83, 1 (2010), 60–66.

[28] Tsong Yueh Chen and Robert Merkel. 2008. An upper bound on software testing effectiveness. *ACM Trans. Softw. Eng. Methodol.* 17, 3, Article 16 (Jun. 2008), 27 pages. DOI: https://doi.org/10.1145/1363102.1363107

[29] T. Y. Chen, T. H. Tse, and Y. T. Yu. 2001. Proportional sampling strategy: A compendium and some insights. *J. Syst. Softw.* 58, 1 (Aug. 2001), 65–81. DOI: https://doi.org/10.1016/S0164-1212(01)00028-0

[30] Ilinca Ciupa, Andreas Leitner, Manuel Oriol, and Bertrand Meyer. 2007. Experimental assessment of random testing for object-oriented software. In *Proceedings of the 2007 International Symposium on Software Testing and Analysis (ISSTA'07)*. ACM, New York, NY, 84–94. DOI: https://doi.org/10.1145/1273463.1273476

[31] I. Ciupa, A. Pretschner, A. Leitner, M. Oriol, and B. Meyer. 2008. On the predictability of random tests for object-oriented software. In *Proceedings of the 2008 1st International Conference on Software Testing, Verification, and Validation*. 72–81. DOI: https://doi.org/10.1109/ICST.2008.20

[32] Ilinca Ciupa, Alexander Pretschner, Manuel Oriol, Andreas Leitner, and Bertrand Meyer. 2011. On the number and nature of faults found by random testing. *Softw. Test. Verif. Reliab.* 21, 1 (2011), 3–28.

[33] P. A. Currit, M. Dyer, and H. D. Mills. 1986. Certifying the reliability of software. *IEEE Trans. Softw. Eng.* SE-12, 1 (Jan. 1986), 3–11. DOI: https://doi.org/10.1109/TSE.1986.6312914

[34] Michael Dyer. 1987. A formal approach to software error removal. *J. Syst. Softw.* 7, 2 (1987), 109–114. DOI: https://doi.org/10.1016/0164-1212(87)90015-X

[35] R. A. Fisher. 1922. On the interpretation of Chi squared from contingency tables, and the calculation of P. *J. Roy. Stat. Soc.* 85, 1 (1922), 87–94. http://www.jstor.org/stable/2340521

[36] Justin E. Forrester and Barton P. Miller. 2000. An empirical study of the robustness of windows NT applications using random testing. In *Proceedings of the 4th Conference on USENIX Windows Systems Symposium—Volume 4 (WSS'00)*. USENIX Association, Berkeley, CA, 6–6. http://dl.acm.org/citation.cfm?id=1267102.1267108

[37] Phyllis G. Frankl and Oleg Iakounenko. 1998. Further empirical studies of test effectiveness. In *Proceedings of the 6th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT'98/FSE-6)*. ACM, New York, NY, 153–162. DOI: https://doi.org/10.1145/288195.288298

[38] G. Fraser and A. Arcuri. 2011. Evolutionary generation of whole test suites. In *Proceedings of the 2011 11th International Conference on Quality Software*. 31–40. DOI: https://doi.org/10.1109/QSIC.2011.19

[39] G. Fraser and A. Arcuri. 2012. Sound empirical evidence in software testing. In *Proceedings of the 2012 34th International Conference on Software Engineering (ICSE'12)*. 178–188. DOI: https://doi.org/10.1109/ICSE.2012.6227195

[40] Gordon Fraser and Andrea Arcuri. 2013. Whole test suite generation. *IEEE Trans. Softw. Eng.* 39, 2 (Feb. 2013), 276–291. DOI: https://doi.org/10.1109/TSE.2012.14

[41] G. Fraser and A. Gargantini. 2009. Experiments on the test case length in specification based test case generation. In *Proceedings of the ICSE Workshop on Automation of Software Test (AST'09)*. 18–26. DOI: https://doi.org/10.1109/IWAST.2009.5069037

[42] Carlo A. Furia, Bertrand Meyer, Manuel Oriol, Andrey Tikhomirov, and Yi Wei. 2013. The search for the laws of automatic random testing. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing (SAC'13)*. ACM, New York, NY, 1211–1216. DOI: https://doi.org/10.1145/2480362.2480590

[43] P. Garg. 1994. Investigating coverage-reliability relationship and sensitivity of reliability to errors in the operational profile. In *Proceedings of the 1st International Conference on Software Testing, Reliability and Quality Assurance 1994.* 21–35. DOI: https://doi.org/10.1109/STRQA.1994.526380

[44] Moheb R. Girgis. 2005. Automatic test data generation for data flow testing using a genetic algorithm. *Journal of Universal Computer Science (JUCS)* 11, 6 (Jun. 2005), 898–915. DOI: https://doi.org/10.3217/jucs-011-06-0898

[45] Patrice Godefroid, Adam Kiezun, and Michael Y. Levin. 2008. Grammar-based whitebox fuzzing. *SIGPLAN Not.* 43, 6 (Jun. 2008), 206–215. DOI: https://doi.org/10.1145/1379022.1375607

[46] Patrice Godefroid, Nils Klarlund, and Koushik Sen. 2005. DART: Directed automated random testing. *SIGPLAN Not.* 40, 6 (Jun. 2005), 213–223. DOI: https://doi.org/10.1145/1064978.1065036

[47] A. L. Goel. 1985. Software reliability models: Assumptions, limitations, and applicability. *IEEE Trans. Softw. Eng.* SE-11, 12 (Dec. 1985), 1411–1423. DOI: https://doi.org/10.1109/TSE.1985.232177

[48] S. S. Gokhale, T. Philip, P. N. Marinos, and K. S. Trivedi. 1996. Unification of finite failure non-homogeneous Poisson process models through test coverage. In *Proceedings of the 7th International Symposium on Software Reliability Engineering 1996.* 299–307. DOI: https://doi.org/10.1109/ISSRE.1996.558886

[49] S. S. Gokhale and K. S. Trivedi. 1998. Log-logistic software reliability growth model. In *Proceedings of the 3rd IEEE International High-Assurance Systems Engineering Symposium 1998.* 34–41. DOI: https://doi.org/10.1109/HASE.1998.731593

[50] Alex Groce, Chaoqiang Zhang, Eric Eide, Yang Chen, and John Regehr. 2012. Swarm testing. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis (ISSTA'12).* ACM, New York, NY, 78–88. DOI: https://doi.org/10.1145/2338965.2336763

[51] Florian Gross, Gordon Fraser, and Andreas Zeller. 2012. EXSYST: Search-based GUI testing. In *Proceedings of the 34th International Conference on Software Engineering (ICSE'12).* IEEE Press, Los Alamitos, CA, 1423–1426. http://dl.acm.org/citation.cfm?id=2337223.2337435

[52] Florian Gross, Gordon Fraser, and Andreas Zeller. 2012. Search-based system testing: High coverage, no false alarms. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis (ISSTA'12).* ACM, New York, NY, 67–77. DOI: https://doi.org/10.1145/2338965.2336762

[53] Michael Grottke. 2002. A vector Markov model for structural coverage growth and the number of failure occurrences. In *Proceedings of the 13th International Symposium on Software Reliability Engineering (ISSRE'02).* IEEE Computer Society, Los Alamitos, CA, 304. http://dl.acm.org/citation.cfm?id=851033.856318

[54] A. Hajjar, T. Chen, I. Munn, A. Andrews, and M. Bjorkman. 2001. High quality behavioral verification using statistical stopping criteria. In *Proceedings of the Design, Automation and Test in Europe, 2001. Conference and Exhibition 2001.* 411–418. DOI: https://doi.org/10.1109/DATE.2001.915057

[55] A. Hajjar, T. Chen, I. Munn, A. Andrews, and M. Bjorkman. 2001. Stopping criteria comparison: Towards high quality behavioral verification. In *Proceedings of the 2001 International Symposium on Quality Electronic Design.* 31–37. DOI: https://doi.org/10.1109/ISQED.2001.915202

[56] Dick Hamlet. 2006. When only random testing will do. In *Proceedings of the 1st International Workshop on Random Testing (RT'06).* ACM, New York, NY, 1–9. DOI: https://doi.org/10.1145/1145735.1145737

[57] Richard Hamlet. 2002. *Random Testing.* John Wiley & Sons, New York, NY. DOI: https://doi.org/10.1002/0471028959.sof268

[58] Shuai Hao, Bin Liu, Suman Nath, William G. J. Halfond, and Ramesh Govindan. 2014. PUMA: Programmable UI-automation for large-scale dynamic analysis of mobile apps. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys'14).* ACM, New York, NY, 204–217. DOI: https://doi.org/10.1145/2594368.2594390

[59] B. Hofer, B. Peischl, and F. Wotawa. 2009. GUI savvy end-to-end testing with smart monkeys. In *Proceedings of the ICSE Workshop on Automation of Software Test (AST'09).* 130–137. DOI: https://doi.org/10.1109/IWAST.2009.5069051

[60] W. E. Howden. 1997. Confidence-based reliability and statistical coverage estimation. In *Proceedings of the 8th International Symposium on Software Reliability Engineering, 1997.* 283–291. DOI: https://doi.org/10.1109/ISSRE.1997.630877

[61] Monica Hutchins, Herb Foster, Tarak Goradia, and Thomas Ostrand. 1994. Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria. In *Proceedings of the 16th International Conference on Software Engineering (ICSE'94).* IEEE Computer Society Press, Los Alamitos, CA, 191–200. http://dl.acm.org/citation.cfm?id=257734.257766

[62] P. Jalote and Y. R. Muralidhara. 1994. A coverage based model for software reliability estimation. In *Proceedings of the 1st International Conference on Software Testing, Reliability and Quality Assurance 1994.* 6–10. DOI: https://doi.org/10.1109/STRQA.1994.526377

[63] Saileshwar Krishnamurthy and Aditya P. Mathur. 1996. On predicting reliability of modules using code coverage. In *Proceedings of the 1996 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON'96).* IBM Press, 22-. http://dl.acm.org/citation.cfm?id=782052.782074

[64] M. Lil, Y. Wei, D. Desovski, H. Nejad, S. Ghose, B. Cukic, and C. Smidts. 2004. Validation of a methodology for assessing software reliability. In *Proceedings of the 15th International Symposium on Software Reliability Engineering (ISSRE'04)*. 66–76. DOI: https://doi.org/10.1109/ISSRE.2004.47

[65] Jin-Cherng Lin and Pu-Lin Yeh. 2001. Automatic test data generation for path testing using GAs. *Inf. Sci.* 131, 1–4 (Jan. 2001), 47–64. DOI: https://doi.org/10.1016/S0020-0255(00)00093-1

[66] B. Littlewood and D. Wright. 1997. Some conservative stopping rules for the operational testing of safety critical software. *IEEE Trans. Softw. Eng.* 23, 11 (Nov. 1997), 673–683. DOI: https://doi.org/10.1109/32.637384

[67] H. Liu, X. Xie, J. Yang, Y. Lu, and T. Y. Chen. 2010. Adaptive random testing by exclusion through test profile. In *Proceedings of the 2010 10th International Conference on Quality Software*. 92–101. DOI: https://doi.org/10.1109/QSIC.2010.61

[68] Z. Liu, X. Gao, and X. Long. 2010. Adaptive random testing of mobile application. In *Proceedings of the 2nd International Conference on Computer Engineering and Technology (ICCET'10)*, Vol. 2. IEEE, V2–297.

[69] Michael R Lyu et al. 1996. *Handbook of Software Reliability Engineering*. Vol. 222. IEEE Computer Society, Los Alamitos, CA.

[70] M. R. Lyu, Zubin Huang, S. K. S. Sze, and Xia Cai. 2003. An empirical study on testing and fault tolerance for software reliability engineering. In *Proceedings of the 14th International Symposium on Software Reliability Engineering (ISSRE'03)*. 119–130. DOI: https://doi.org/10.1109/ISSRE.2003.1251036

[71] L. Ma, C. Artho, C. Zhang, H. Sato, J. Gmeiner, and R. Ramler. 2015. GRT: Program-analysis-guided random testing (T). In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE'15)*. 212–223. DOI: https://doi.org/10.1109/ASE.2015.49

[72] Aravind Machiry, Rohan Tahiliani, and Mayur Naik. 2013. Dynodroid: An input generation system for Android apps. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE'13)*. ACM, New York, NY, 224–234. DOI: https://doi.org/10.1145/2491411.2491450

[73] Riyadh Mahmood, Nariman Mirzaei, and Sam Malek. 2014. EvoDroid: Segmented evolutionary testing of Android apps. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'14)*. ACM, New York, NY, 599–609. DOI: https://doi.org/10.1145/2635868.2635896

[74] Y. K. Malaiya, M. N. Li, J. M. Bieman, and R. Karcich. 2002. Software reliability growth with test coverage. *IEEE Trans. Reliabil.* 51, 4 (Dec. 2002), 420–426. DOI: https://doi.org/10.1109/TR.2002.804489

[75] Y. K. Malaiya, Naixin Li, J. Bieman, R. Karcich, and B. Skibbe. 1994. The relationship between test coverage and reliability. In *Proceedings of the 5th International Symposium on Software Reliability Engineering 1994*. 186–195. DOI: https://doi.org/10.1109/ISSRE.1994.341373

[76] N. Malevris and E. Petrova. 2000. On the determination of an appropriate time for ending the software testing process. In *Proceedings of the 1st Asia-Pacific Conference on Quality Software 2000*. 81–88. DOI: https://doi.org/10.1109/APAQ.2000.883781

[77] Nashat Mansour and Miran Salame. 2004. Data generation for path testing. *Softw. Qual. J.* 12, 2 (Jun. 2004), 121–136. DOI: https://doi.org/10.1023/B:SQJO.0000024059.72478.4e

[78] Leonardo Mariani, Mauro Pezzè, Oliviero Riganelli, and Mauro Santoro. 2014. Automatic testing of GUI-based applications. *Softw. Test. Verif. Reliab.* 24, 5 (Aug. 2014), 341–366. DOI: https://doi.org/10.1002/stvr.1538

[79] Leonardo Mariani, Mauro PezzÃl, and Daniele Zuddas. 2015. Recent advances in automatic black-box testing. In *Advances in Computers*, Vol. 99. Elsevier, 157–193. DOI: https://doi.org/10.1016/bs.adcom.2015.04.002

[80] Lorenzo Martignoni, Roberto Paleari, Giampaolo Fresi Roglia, and Danilo Bruschi. 2010. Testing system virtual machines. In *Proceedings of the 19th International Symposium on Software Testing and Analysis (ISSTA'10)*. ACM, New York, NY, 171–182. DOI: https://doi.org/10.1145/1831708.1831730

[81] Tim Menzies and Bojan Cukic. 2000. When to test less. *IEEE Softw.* 17, 5 (2000), 107–112. DOI: https://doi.org/doi.ieeecomputersociety.org/10.1109/52.877876

[82] James Miller, Marek Reformat, and Howard Zhang. 2006. Automatic test data generation using genetic algorithm and program dependence graphs. *Inf. Softw. Technol.* 48, 7 (2006), 586–605. DOI: https://doi.org/10.1016/j.infsof.2005.06.006

[83] Rajeev Motwani and Prabhakar Raghavan. 1995. *Randomized Algorithms*. Cambridge University Press, New York, NY.

[84] Akbar Siami Namin and James H. Andrews. 2009. The influence of size and coverage on test suite effectiveness. In *Proceedings of the 18th International Symposium on Software Testing and Analysis (ISSTA'09)*. ACM, New York, NY, 57–68. DOI: https://doi.org/10.1145/1572272.1572280

[85] Simeon C. Ntafos. 2001. On comparisons of random, partition, and proportional partition testing. *IEEE Trans. Softw. Eng.* 27, 10 (Oct. 2001), 949–960. DOI: https://doi.org/10.1109/32.962563

[86] P. Oehlert. 2005. Violating assumptions with fuzzing. *IEEE Secur. Priv.* 3, 2 (Mar. 2005), 58–62. DOI: https://doi.org/10.1109/MSP.2005.55

[87] M. Oriol. 2012. Random testing: Evaluation of a law describing the number of faults found. In *Proceedings of the 2012 IEEE 5th International Conference on Software Testing, Verification and Validation*. 201–210. DOI: https://doi.org/10.1109/ICST.2012.100

[88] Alessandro Orso and Gregg Rothermel. 2014. Software testing: A research travelogue (2000–2014). In *Proceedings of the on Future of Software Engineering (FOSE'14)*. ACM, New York, NY, 117–132. DOI: https://doi.org/10.1145/2593882.2593885

[89] Carlos Pacheco and Michael D. Ernst. 2005. Eclat: Automatic generation and classification of test inputs. In *Proceedings of the 19th European Conference on Object-Oriented Programming (ECOOP'05)*. Springer-Verlag, Berlin, 504–527. DOI: https://doi.org/10.1007/11531142_22

[90] C. Pacheco, S. K. Lahiri, M. D. Ernst, and T. Ball. 2007. Feedback-directed random test generation. In *Proceedings of the 29th International Conference on Software Engineering (ICSE'07)*. 75–84. DOI: https://doi.org/10.1109/ICSE.2007.37

[91] David L. Parnas, A. John van Schouwen, and Shu Po Kwan. 1990. Evaluation of safety-critical software. *Commun. ACM* 33, 6 (Jun. 1990), 636–648. DOI: https://doi.org/10.1145/78973.78974

[92] Min Pei, Erik D. Goodman, Zongyi Gao, and Kaixiang Zhong. 1994. Automated software test data generation using a Genetic Algorithm. Technical Report. Michigan State University.

[93] Paul Piwowarski, Mitsuru Ohba, and Joe Caruso. 1993. Coverage measurement experience during function test. In *Proceedings of the 15th International Conference on Software Engineering (ICSE'93)*. IEEE Computer Society Press, Los Alamitos, CA, 287–301. http://dl.acm.org/citation.cfm?id=257572.257635

[94] James Ramsey and Victor R. Basili. 1985. Analyzing the test process using structural coverage. In *Proceedings of the 8th International Conference on Software Engineering (ICSE'85)*. IEEE Computer Society Press, Los Alamitos, CA, 306–312.

[95] Abdul Rauf, Sajid Anwar, M. Arfan Jaffar, and Arshad Ali Shahid. 2010. Automated GUI test coverage analysis using GA. In *Proceedings of the 7th International Conference on Information Technology: New Generations (ITNG'10)*. 1057–1062. DOI: https://doi.org/10.1109/ITNG.2010.95

[96] Lenin Ravindranath, Suman Nath, Jitendra Padhye, and Hari Balakrishnan. 2014. Automatic and scalable fault detection for mobile applications. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys'14)*. ACM, New York, NY, 190–203. DOI: https://doi.org/10.1145/2594368.2594377

[97] Urko Rueda, Tanja E. J. Vos, and I. S. W. B. Prasetya. 2015. Unit testing tool competition: Round three. In *Proceedings of the 8th International Workshop on Search-Based Software Testing (SBST'15)*. IEEE Press, Piscataway, NJ, 19–24. http://dl.acm.org/citation.cfm?id=2821339.2821346

[98] M. Sahinoglu. 2003. An empirical Bayesian stopping rule in testing and verification of behavioral models. *IEEE Trans. Instrum. Meas.* 52, 5 (Oct. 2003), 1428–1443. DOI: https://doi.org/10.1109/TIM.2003.818548

[99] M. Sahinoglu, A. von Mayrhauser, A. Hajjar, T. Chen, and C. Anderson. 1999. On the efficiency of a compound Poisson stopping rule for mixed strategy testing. In *Proceedings of the 1999 IEEE Aerospace Conference*, Vol. 5. 93–98 vol. 5. DOI: https://doi.org/10.1109/AERO.1999.790193

[100] Rohan Sharma, Milos Gligoric, Andrea Arcuri, Gordon Fraser, and Darko Marinov. 2011. Testing container classes: Random or systematic? In *Proceedings of the 14th International Conference on Fundamental Approaches to Software Engineering: Part of the Joint European Conferences on Theory and Practice of Software (FASE'11/ETAPS'11)*. Springer-Verlag, Berlin, 262–277. http://dl.acm.org/citation.cfm?id=1987434.1987460

[101] Elena Sherman, Matthew B. Dwyer, and Sebastian Elbaum. 2009. Saturation-based testing of concurrent programs. In *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'09)*. ACM, New York, NY, 53–62. DOI: https://doi.org/10.1145/1595696.1595706

[102] P. Thévenod-Fosse and H. Waeselynck. 1991. An investigation of statistical software testing. *Softw. Test. Verif. Reliabil.* 1, 2 (1991), 5–25. DOI: https://doi.org/10.1002/stvr.4370010203

[103] Petar Tsankov, Mohammad Torabi Dashti, and David Basin. 2012. SecFuzz: Fuzz-testing security protocols. In *Proceedings of the 7th International Workshop on Automation of Software Test (AST'12)*. IEEE Press, Piscataway, NJ, 1–7. http://dl.acm.org/citation.cfm?id=2663608.2663610

[104] Alan Veevers and Adam C. Marshall. 1994. A relationship between software coverage metrics and reliability. *Softw. Test. Verif. Reliabil.* 4, 1 (1994), 3–8. DOI: https://doi.org/10.1002/stvr.4370040103

[105] Y. Wei, S. Gebhardt, B. Meyer, and M. Oriol. 2010. Satisfying test preconditions through guided object selection. In *Proceedings of the 2010 3rd International Conference on Software Testing, Verification and Validation*. 303–312. DOI: https://doi.org/10.1109/ICST.2010.34

[106] Yi Wei, Bertrand Meyer, and Manuel Oriol. 2012. Is branch coverage a good measure of testing effectiveness? In *Empirical Software Engineering and Verification*, Bertrand Meyer and Martin Nordio (Eds.). Lecture Notes in Computer Science, Vol. 7007. Springer, Berlin, 194–212. DOI: https://doi.org/10.1007/978-3-642-25231-0_5

[107] Elaine J. Weyuker and Bingchiang Jeng. 1991. Analyzing partition testing strategies. *IEEE Trans. Softw. Eng.* 17, 7 (Jul. 1991), 703–711. DOI: https://doi.org/10.1109/32.83906

[108] J. A. Whittaker and M. G. Thomason. 1994. A Markov chain model for statistical software testing. *IEEE Trans. Softw. Eng.* 20, 10 (Oct. 1994), 812–824. DOI: https://doi.org/10.1109/32.328991

[109] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. 2012. *Experimentation in Software Engineering*. Springer, Berlin.

Received March 2018; revised February 2019; accepted May 2019