Proceedings Paper:
Shin, D. orcid.org/0000-0002-0840-6449, Yoo, S. and Bae, D.-H. (2016) Diversity-aware mutation adequacy criterion for improving fault detection capability. In: 2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW). 2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 11-15 Apr 2016, Chicago, IL, USA. Institute of Electrical and Electronics Engineers (IEEE) , pp. 122-131. ISBN 9781509036752

https://doi.org/10.1109/icstw.2016.37

Diversity-Aware Mutation Adequacy Criterion for Improving Fault Detection Capability

Donghwan Shin
School of Computing
KAIST
Daejeon, Republic of Korea
Email: donghwan@se.kaist.ac.kr

Shin Yoo
School of Computing
KAIST
Daejeon, Republic of Korea
Email: shin.yoo@kaist.ac.kr

Doo-Hwan Bae
School of Computing
KAIST
Daejeon, Republic of Korea
Email: bae@se.kaist.ac.kr

***Abstract*—Many existing testing techniques adopt diversity as an important criterion for the selection and prioritization of tests. However, mutation adequacy has been content with simply maximizing the number of mutants that have been killed. We propose a novel mutation adequacy criterion that considers the diversity in the relationship between tests and mutants, as well as whether mutants are killed. Intuitively, the proposed criterion is based on the notion that mutants can be distinguished by the sets of tests that kill them. A test suite is deemed adequate by our criterion if the test suite distinguishes all mutants in terms of their kill patterns. Our hypothesis is that, simply by using a stronger adequacy criterion, it is possible to improve fault detection capabilities of mutation-adequate test suites. The empirical evaluation selects tests for real world applications using the proposed mutation adequacy criterion to test our hypothesis. The results show that, for real world faults, test suites adequate to our criterion can increase the fault detection success rate by up to 76.8 percentage points compared to test suites adequate to the traditional criterion.**

I. Introduction

One fundamental limitation of software testing is the fact that, to validate the behavior of the Program Under Test (PUT), we can only ever sample a very small number of test inputs out of the vast input space. Almost all existing testing techniques are, at some level, attempts to answer the following question: *how does one sample a finite number of test inputs to cover as wide a range of program behaviours as possible?*

The concept of diversity has received much attention while answering the above question. For example, Adaptive Random Testing (ART) [1] seeks to increase diversity of randomly sampled test inputs by choosing an input that is as different from those already sampled as possible. Clustering-based test selection and prioritization [2], [3] assumes that a diverse set of test inputs would explore and validate a wider range of program behaviors. Diversity in test output has been studied as a test adequacy criterion for black box testing of web applications [4]. Information theoretic measures of diversity has also been studied as a test selection criterion [5], [6].

In contrast, diversity has received little attention in relation to mutation testing. Mutation adequacy remains essentially as a simple count of the number of killed mutants. Many of existing works focus either on reducing the cost of mutation testing (i.e., *do fewer, do smarter*, and *do faster* as summarized by Offutt and Untch [7]) or analyzing equivalent mutants [8], [9] (i.e., mutants semantically equivalent to an original program). Relatively very few attention has been paid to improve the fault detection capability of the mutation adequacy criterion itself.

The existing mutation adequacy as a count of killed mutants does not cater for diversity. Consequently, despite its potential correlation to the fault detection capability, many diverse mutants are generated but wasted. Suppose a pathological case in which a single test can kill all generated mutants. In terms of the kill information, it means that the single test does not capture the diversity of the mutants enough, while the traditional mutation adequacy simply determines the single test as adequate. Such a case calls for a richer adequacy criterion in mutation testing.

This paper proposes a novel adequacy criterion called distinguishing mutation adequacy criterion, which includes the notion of diversity. The proposed metric is based on our previous work on theoretical framework for mutation testing [10]. At the core of the new criterion lies the idea that mutants can be *distinguished* from each other by the set of tests that kill them. Our mutation adequacy criterion aims not only to kill, but also to distinguish as many mutants as possible. The aforementioned pathological case of a single test killing all mutants will perform poorly under our new criterion.

By aiming to distinguish the maximum number of mutants, the proposed adequacy criterion can select more diverse set of test cases. Suppose there exist two mutants. Test t_1 kills both, while t_2 and t_3 kill different one each. Under the existing criterion, the set $\{t_1\}$ is deemed adequate, whereas the proposed criterion will choose the set $\{t_2, t_3\}$ instead. We hypothesize that this, more diverse set of tests will show higher fault detection capability.

The hypothesis on fault detection capability is validated by an empirical study. We use the `Defects4J` [11] data set to study real world faults in non-trivial systems. The control group consists of sets of test suites selected based on the traditional mutation adequacy (i.e., ones that kill all mutants), while the treatment group consists of sets of test suites that can collectively kill and distinguish all mutants. Both groups are evaluated for their fault detection capabilities by executing them against faulty and fixed versions of programs collected from the real world. The results show that our novel mutation

adequacy criterion shows either equal or higher fault detection capabilities for all studied subject faults.

The technical contributions of this paper are as follows:

- This paper introduces a novel mutation adequacy criterion called *distinguishing mutation criterion*. A test suite is mutation adequate with respect to this criterion if all considered mutants have unique sets of tests that kill them, i.e. can be *distinguished* by their kill patterns.
- The proposed adequacy criterion is empirically evaluated using real faults in the `Defects4J` repository and random testing. The results show that the new adequacy criterion shows at least equal or higher fault detection capability than the traditional mutation adequacy criterion. The increase in the fault detection success rate is up to 76.8 percentage points.

The rest of the paper is organised as follows. Section II presents a formal definition of the existing mutation adequacy criterion. Section III introduces the distinguishing mutants adequacy criterion using the same formal notations. Section IV describes the design of our empirical evaluation, the results of which are presented and analysed in Section V. Section VI discusses related work, and finally Section VII concludes.

II. Background

A. Formal Model of Mutation Testing

To formally represent mutation adequacy criteria considered in this paper, we summarize the essential elements of the formal framework for the mutation-based testing methods. Detailed descriptions for the formal framework are presented in [10].

Let P be a set of programs which includes the program under test. In mutation testing, there are three essential programs in P: an original program $p_o \in P$, a mutant $m \in M \subseteq P$ generated from p_o, and a correct program $p_s \in P$ which represents the true requirements[1] about p_o. For a test $t \in T$ for P, if the behaviors of p_o and p_s are different, it is said that t *detects* a fault in p_o. Similarly, if the behaviors of p_o and m are different for t, it is said that t *kills* m. Note that the notion of behavioral difference is an abstract concept. It is formalized by a testing factor, called a test differentiator, which is defined as follows:

Definition 1: A *test differentiator* $d : T \times P \times P \rightarrow \{0, 1\}$ is a function,[2] such that

$$d(t, p_x, p_y) = \begin{cases} 1\ (true), & \text{if } p_x \text{ is } \textit{different} \text{ with } p_y \text{ for } t \\ 0\ (false), & \text{otherwise} \end{cases}$$

for all tests $t \in T$ and programs $p_x, p_y \in P$.

By definition, a test differentiator concisely represents whether the behaviors of $p_x \in P$ and $p_y \in P$ are different for t. We make no attempt to incorporate any specific definition of program differences. The specific definition of differences can only be decided in context. For example, while 0.3333 is different with 1/3 in the strict sense, 0.3333 will be regarded as the same as 1/3 in some cases. To keep things general, we consider a set of test differentiators D that includes all possible test differentiators for P.

A test differentiator, or simply differentiator, can formally describe the notion of differences in mutation testing. For example, when t detects a fault in p_o, it is clearly formalized as follows[3]:

$$d(t, p_o, p_s) = 1$$

On the other hand, when t kills a mutant m, it is also clearly formalized as follows:

$$d(t, p_o, m) = 1$$

Note that p_o, p_s, and m are general entities, and largely separated from any specifics such as programming languages or mutation methods.

B. Mutation Adequacy Criterion

Since mutation testing was first proposed in the 1970s, it has been widely studied in the aspects of both theory and practice, and a mutation adequacy criterion has played the key role in the studies of mutation testing. A mutation adequacy criterion is a predicate that determines the adequacy of a test suite using mutants. It is said that a test suite is *mutation-adequate* when the test suite kills all of the generated mutants. Using a differentiator, it is clearly and concisely formalized as follows:

$$\forall m \in M, \exists t \in TS, d(t, p_o, m) = 1. \quad (1)$$

In other words, a test suite TS is mutation-adequacy if all mutants $m \in M$ are killed by at least one test $t \in TS$.

Equation (1) is general enough to consider various mutation testing approaches. For example, there is a spectrum of mutation approaches from a strong mutation [12] to a weak mutation [13], depending on which d is used. In a strong mutation analysis, a test t kills a mutant m when the output of m differs from the output of the original program p_o for t. In a weak mutation analysis, t kills m when the internal states of m and p_o are different for t. In the rest of this paper, we refer (1) as the traditional mutation adequacy criterion in compared to the new mutation adequacy proposed in Section III-B.

III. Extending Mutation Adequacy Considering Diversity of Mutants

A. Limitation of Traditional Mutation Adequacy

To see the limitation of the traditional mutation adequacy criterion, we provide a working example with four mutants and three tests in Figure 1. Each of the values represents whether a test kills a mutant. For example, $d(t_1, p_o, m_1)$ is 1 which means that t_1 kills m_1.

[1] While p_s is not a real program, this is not a serious assumption, because we only require the behavior of p_s for a given set of tests. In practice, a human may play the role of p_s, acting as a human oracle.

[2] This function-style definition is replaceable by a predicate-style definition, such as $d \subseteq T \times P \times P$.

[3] In experiments, when the correct version of a program for a fault is known in advance, the correct version can be used as p_o. In this case, the corresponding faulty version should be used as p_s so that the difference between p_o and p_s implies the fault.

Test	$d(t_i, p_o, m_1)$	$d(t_i, p_o, m_2)$	$d(t_i, p_o, m_3)$	$d(t_i, p_o, m_4)$
t_1	1	1	1	1
t_2	0	0	1	1
t_3	0	1	0	1

Fig. 1. An working example for demonstrating the limitation of the traditional mutation adequacy criterion. The table represents whether a test kills a mutant. For example, $d(t_1, p_o, m_1)$ is 1 which means that t_1 kills m_1.

In the working example, a test suite $TS_1 = \{t_1\}$ is adequate to both the traditional mutation adequacy criterion and the traditional mutation adequacy criterion, because all the four mutants are killed by t_1. However, TS_1 does not capture the diversity of the four mutants, and the mutants are simply redundant to TS_1. This implies that mutants are generated but wasted in terms of the traditional mutation adequacy criterion, because it does not consider the diversity of mutants.

B. Diversity-Aware Mutation Adequacy Criterion

To consider the diversity of mutants in terms of the adequacy criterion of test suites, we first formally define the distinguishment of mutants with respect to a test as follows:

Definition 2: Two mutants m_x and m_y generated from p_o are *distinguished* by a test t if and only if the following condition holds:

$$d(t, p_o, m_x) \neq d(t, p_o, m_y)$$

for a differentiator d.

In other words, two mutants are distinguished by a test when the test differentiates the two mutants' kill patterns. In the working example, the four mutants are undistinguished from each other by t_1 because $d(t_1, p_o, m_i) = 1$ for all $i \in \{1, \cdots, 4\}$. By t_2, m_1 is distinguished from m_3 and m_4 but not from m_2. By t_3, m_1 is distinguished from m_2 and m_4 but not from m_3.

In terms of a set of tests, a mutant is *killed by a set of tests* when there is at least one test that kills a mutant. Similarly, we extend the distinguishment of mutants with respect to a set of tests as follows: two mutants are *distinguished by a set of tests* when there is at least one test that distinguishes the two mutants. In the working example, if we consider a test suite $TS_2 = \{t_1, t_2\}$, m_1 is distinguished from m_3 and m_4 but not distinguished from m_2. By another test suite $TS_3 = \{t_1, t_2, t_3\}$, all of the four mutants are distinguished from each other.

We now define a new mutation adequacy criterion, called a *distinguishing mutation adequacy criterion*, as follows:

Definition 3: For a set of mutants M generated from an original program p_o, a test suite TS is *distinguishing mutation-adequate* when the following condition holds:

$$\forall m_x, m_y \in M', \exists t \in TS, d(t, p_o, m_x) \neq d(t, p_o, m_y)$$

where $m_x \neq m_y$, $M' = M \cup \{m_o\}$, and $m_o = p_o$.

In other words, a test suite TS is distinguishing mutation-adequate if all possible pair of different mutants m_x and m_y in M' are distinguished by at least one test $t \in TS$. In the working example, M' is $\{m_o, m_1, \cdots, m_4\}$ and there are $\binom{5}{2} = 10$ pairs of mutants including (m_o, m_1), (m_o, m_1), $\cdots$, (m_1, m_2), (m_1, m_3), $\cdots$, (m_3, m_4). Among the test suites TS_1, TS_2, and TS_3, only TS_3 is adequate to the distinguishing mutation adequacy criterion.

It is important to appreciate the role of m_o in the distinguishing mutation adequacy. Consider $m_y = m_o$, the distinguishing mutation adequacy criterion is simplified as follows:

$$\forall m_x \in M, \exists t \in TS, d(t, p_o, m_x) \neq d(t, p_o, m_o). \quad (2)$$

Since it is trivial that $d(t, p_o, m_o) = 0$ for all $t \in T$, (2) is exactly same as (1) (i.e., the traditional mutation adequacy criterion). This means that the distinguishing mutation adequacy criterion *subsumes* the traditional mutation adequacy criterion. In other words, if a test suite is adequate to the distinguishing mutation adequacy criterion, the test suite is guaranteed to be adequate to the traditional mutation adequacy criterion.

For the sake of simplicity, let d-criterion hereafter refer to the distinguishing mutation adequacy criterion (i.e., diversity-aware) and, similarly, k-criterion to the traditional mutation adequacy criterion (i.e., kill-only).

C. Time Complexity for Calculating Adequacy

In mutation testing, most time consuming computation is the execution of mutants for tests to determine whether it is killed or not. However, this cost is featured equally for both d-criterion and k-criterion because both criteria needs to determine whether a mutant is killed by a test or not. What differs is that d-criterion additionally needs to compare the kill information of each mutant as a pair. In Section V-D, we will see how much additional time takes for the comparison.

As a test suite selection criterion, for each test, the d-criterion should decide whether it should add a test to the test suite or not. It may initially appear that the number of comparison for n mutants is $\binom{n}{2}$ because it is the number of all possible pairs among the set of n elements. However, we need to compare not all n mutants but only k ($\leq n$) undistinguished mutants among them because the distinguishment of mutants is monotonic—once two mutants are distinguished, they are never subsequently undistinguished.

Further, the number of comparison for k undistinguished mutants is only $k - 1$ because a test can partition the set of undistinguished mutants into only two groups—one for the mutants that are killed by the test, and the other for the mutants that are not killed by the test. In the working example, for the 5 initially undistinguished mutants $(m_o, \cdots, m_4)$, t_3 partitions it into two groups (m_0, m_1, m_3) and (m_2, m_4). Here, it is required to consider not all $\binom{5}{2} = 10$ pairs but only the 4 pairs (i.e., (m_0, m_1), (m_0, m_2), (m_0, m_3), and (m_0, m_4)) to calculate the distinguishment of mutants by t_3. As a result, the number of comparison for each test is linearly proportional to the number of undistinguished mutants, making the d-criterion

more practical. The detailed algorithm for test suite selection will be explained in Section IV-B.

D. Generalized Equivalent Mutant Problem

In the k-criterion, it is possible that a mutant is semantically equivalent to an original program so that there is no test to kill it. The mutant is called an *equivalent mutant*. Formally, an equivalent mutant m_e is described as $\forall t \in T, d(t, p_o, m_e) = 0$. Unfortunately, deciding whether a mutant is equivalent or not is undecidable. However, many researchers have attempt to tackle this problem with practical approximation [14]–[16].

In the d-criterion, it is possible that two mutants are semantically equivalent to each other so that there is no test to distinguish them. Let us call the mutants *universally undistinguishable mutants*. Formally, the universally undistinguishable mutants m_x and m_y are described as $\forall t \in T, d(t, p_o, m_x) = d(t, p_o, m_y)$. This is essentially the extension of the equivalent mutant, which means the deciding whether a pair of mutants are universally undistinguishable or not is another undecidable problem. Many ideas developed to tackle the traditional equivalent mutant problem are fully applicable to this extended problem as well. However, attempting to solve this problem is not in the scope of this paper.

IV. Empirical Evaluation Design

In the experimental evaluation, we investigate the following four main research questions:

- RQ1: Are there rooms for improvement of the fault detection capability by distinguishing more mutants?
- RQ2: Is the distinguishing mutation adequacy criterion more likely to detect faults than the traditional mutation adequacy criterion?
- RQ3: How many tests are needed to be a mutation-adequate test suite?
- RQ4: How much time takes for selecting a mutation-adequate test suite?

RQ1 deals with the necessary question before we analyze the improvement of fault detection capability. For example, it may be the case that a traditional mutation-adequate test suite is capable of distinguishing all mutants for some faults in practice. In this case, it is not possible for a distinguishing mutation-adequate test suite to improve the fault detection capability by distinguishing more mutants. We classify faults as several types and analyze the room for improvement for each type of faults.

RQ2 is the main question of this paper: improving the fault detection capabilities of mutation-adequate test suites. If using a new test adequacy criterion improves the fault detection capabilities for real faults, it is worth to consider to adapt the new adequacy criterion not only in research but also in practice. We compare the fault detection capabilities of the traditional and distinguishing mutation adequacy for real faults.

RQ3 is a practical question. The number of tests is directly related to the testing efforts, especially when the cost of test oracles is considered. A test suite with too many tests may be practically useless even if its fault detection effectiveness is promising. We compare the size of adequate test suites (i.e., the number of tests) for the traditional and distinguishing mutation adequacy.

RQ4 considers the time complexity for calculating the mutant distinguishment in terms of test suite selection. We compare the test suite selection times for the traditional and distinguishing mutation adequacy.

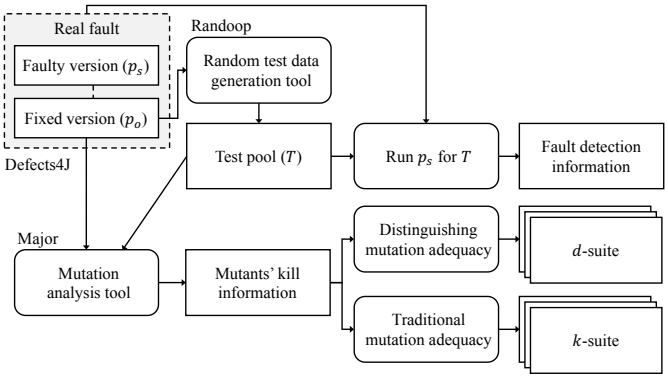

Fig. 2. Experimental setup: overview

To answer the above questions, we design our experiments as described in Figure 2.

We use the developer-fixed and manually-verified faults of the real applications in the database of `Defects4J` [11]. For each fault, we generate a large number of random tests as a test pool with the aid of `Randoop` [17]. The test pool is used to execute many mutants generated by the mutation analysis tool `Major` [18], and it returns the kill information for all mutants to the test pool. We then generate distinguishing mutation-adequate test suites and traditional mutation-adequate test suites using the kill information. Meanwhile, since `Defects4J` provides both faulty and fixed version of programs for each fault, it is measurable whether each of the generated test suites detects the fault or not.

A. Subject Faults

We conduct experiments on real applications provided by `Defects4J`. Its database includes 357 developer-fixed and manually-verified real faults and corresponding fixes from five applications (JFreeChart, Closure compiler, Commons Math, Joda-Time, and Commons Lang). For each fault, the faulty version and the fixed version of the fault is given. The difference between the faulty and fixed version of a fault does not include unrelated changes such as refactorings. Since each fault is given as an independent fault-fix pair of program versions, we treat each fault as that of a separate subject program.

We study the subset of the 357 faults which satisfy the following conditions: (1) a fault must be detected by a test pool, and (2) at least one mutant generated from the fault must be killed by the test pool. As a result, 45 real faults are remained. In Table I, the column labeled as fault represents the identifiers of the 45 faults given by the `Defects4J`

database. For example, Chart-5 means the fifth real fault of the JFreeChart program.

B. Test Pool Generation

As we attempt to generate a large number of unbiased tests as a test pool for the test suite selection, we use a random test generation tool, `Randoop` [17]. For each fault, we generate at most 10,000 tests with the 200 seconds time limit. However, not all test pools for the studied faults reached 10,000 tests because `Randoop` may generate tests that make compile errors, runtime errors, and sporadically fails as noted in [11]. We automatically removed those problematic tests using the script given in `Defects4J`. In Table I, the columns labeled test pool size and trigs represent the total number of tests in the test pool and the number of fault-triggering tests among the test pool, respectively, for each fault. For example, Chart-5 has the test pool whose size is 10,000, and only 165 tests among the pool are capable of detecting the Chart-5 fault.

C. Mutant Generation and Execution

We use `Major` [18] mutation analysis tool for generating and executing all mutants to the test pool for each fault. Be default, `Major` provides a set of commonly used mutation operators [19] including binary operator replacement, unary operator replacement, constant value replacement, branch condition manipulation, and statement deletion. We applied all the mutation operators and generate all possible mutants for each fault. In Table I, the columns labeled as mutation analysis represent the mutation-related information including the total number of generated mutants (allM), killed mutants (by the test pool) (kM), distinguished mutants (by the test pool) (dM), and the mutants-tests execution time (time), for each fault. The number of killed and distinguished mutants (by the test pool) are the maximum number of killable and distinguishable mutants in test suite selection, respectively.

D. Test Suite Generation

We generate an adequate test suite by selecting tests from a test pool for each fault. We define the fault detection capability of an adequacy criterion as the probability that a test suite selected to be adequate to the criterion will detect a fault [20]. Considering that test suites adequate to the same adequacy criterion may have different fault detection capabilities in practice, we generate 500 adequate test suites for each criterion and analyze them to obtain statistically sound analysis results. Let 500 distinguishing mutation-adequate test suites as a d-suite (i.e., distinguish-suite) and 500 traditional mutation-adequate test suites as a k-suite (i.e., kill-suite). For example, let x be the number of fault-detecting test suites among k-suite for a fault. Then, the fault detection success rate of the k-suite $x/500$ implies the fault detection capability of the k-criterion for the fault.

Algorithm 1 shows how to generate a d-suite from a test pool. The algorithm takes a test pool T, a set of mutants M, an original program p_o, and the maximum number of distinguishable mutants (with respect to the test pool) max_δ as inputs, and returns a distinguishing mutation-adequate test suite TS_{dist}. In Algorithm 1, δ_{map} represents the dictionary of mutant distinguishment. The values of δ_{map} are the set of undistinguished mutants. For each set of undistinguished mutants, one mutant in the set is selected (and removed from the set) as the key for the dictionary. For example, Line 4 initializes δ_{map} with the key p_o and the value M, because all mutants $m \in M$ and $p_o = m_o$ are undistinguished at first. Lines 8-16 calculate and handle the distinguishment of previously undistinguished mutants. Lines 17-20 update δ_{map} and check the exit condition of the while-loop.

Algorithm 1 d-suite generation

```
1:  function GENERATEDISTTS(T, M, p_o, max_δ)
2:      TS_dist ← SET()
3:      δ_map ← DICT()              ▷ k: a mutant, v: a set of mutants
4:      PUT(δ_map[p_0], M)
5:      while True do
6:          t ←POP(T)
7:          δ'_map ← DICT()
8:          for all m_k ∈ KEYS(δ_map) do
9:              M_tmp ← SET()
10:             for all m_u ∈ δ_map[m_k] do
11:                 if d(t, p_o, m_k) ≠ d(t, p_o, m_u) then
12:                     ADD(M_tmp, m_u)
13:             if ISNOTEMPTY(M_tmp) then
14:                 REMOVEALL(δ_map[m_k], M_tmp)
15:                 m_k' ← POP(M_dist)
16:                 PUT(δ'_map[m_k'], M_tmp)
17:         if ISNOTEMPTY(δ'_map) then
18:             ADD(TS_dist, t)
19:             PUTALL(δ_map, δ'_map)
20:             if SIZE(KEYS(δ_map)) = max_δ then
21:                 break
22:     return TS_dist
```

V. Results and Analysis

A. RQ1: Are there rooms for improvement of the fault detection effectiveness by distinguishing more mutants?

To consider the room for improvement of the fault detection success rates of d-suites by distinguishing more mutants in compared to k-suites, we have classified the studied faults based on the fault detection success rate and the distinguished mutants rate of the k-suite of each fault. The distinguished mutants rate of a k-suite for a fault is the ratio of the average number of distinguished mutants by the k-suite to the maximum number of mutants distinguished by the test pool. The results are presented in Figure 3.

There are four types of faults in Figure 3. For type1 faults, k-suites neither certainly detect the faults nor distinguish all mutants from the faults. More type1 faults means more potential rooms for improvement of fault detection success rates of d-suites. Type2 faults are not certainly detected by k-suites while all mutants are distinguished by k-suites. Type3 includes faults that are certainly detected by k-suites while not all mutants are distinguished by k-suites. Type4 faults are certainly detected and their mutants are fully distinguished by k-suites. The sum of type3 and type4 faults implies the

TABLE I
SUBJECT FAULTS, TESTS, AND MUTANTS

Fault	Test pool size	Test pool trigs	Mutation analysis allM	Mutation analysis kM	Mutation analysis dM	Mutation analysis time (sec)	Fault	Test pool size	Test pool trigs	Mutation Analysis allM	Mutation Analysis kM	Mutation Analysis dM	Mutation Analysis time (sec)
Chart-5	10000	165	271	168	110	2157	Math-9	3788	1	84	51	31	106
Chart-11	236	34	221	26	13	82	Math-14	5492	2	24	10	6	206
Chart-12	6759	2	51	44	24	1314	Math-22	2865	62	285	267	84	293
Chart-14	5693	9	762	198	88	1320	Math-27	8567	7	400	296	227	2733
Chart-15	5639	24	198	147	113	12502	Math-29	3574	11	220	146	89	396
Chart-16	9341	6254	161	153	131	4880	Math-35	10000	21	33	15	8	206
Chart-17	3398	376	316	146	71	280	Math-60	10000	2	125	116	76	2474
Chart-18	1897	147	218	121	73	652	Math-61	3896	296	64	57	36	569
Chart-22	3370	307	112	63	28	204	Math-66	3740	3	22	6	4	57
Chart-24	930	238	35	28	27	31	Math-68	3728	9	10	2	2	53
Closure-56	6411	14	174	128	59	813	Math-77	1229	1	918	542	217	1902
Closure-107	5421	1	119	3	3	296	Math-90	515	13	60	48	32	35
Lang-12	857	8	139	76	45	526	Math-92	541	13	927	623	278	3128
Lang-37	6956	55	1924	1319	817	6307	Math-93	273	1	794	517	207	525
Lang-41	749	368	214	99	19	675	Math-95	872	23	74	67	42	40
Lang-45	9114	28	300	215	127	220	Math-98	3026	811	1366	950	401	2625
Lang-56	3707	381	495	337	246	12223	Math-102	2844	46	282	206	61	217
Lang-59	1199	28	1412	652	321	408	Math-103	5878	226	108	91	75	707
Math-1	2045	3	710	415	236	962	Math-104	4216	319	337	302	167	1783
Math-3	2087	6	1535	1292	911	2151	Time-8	890	5	379	170	56	1282
Math-4	3537	1	48	6	6	65	Time-9	5686	21	369	221	150	5023
Math-5	1476	2	670	506	291	2689	Time-13	1695	5	910	398	84	1263
Math-6	6672	12	37	18	16	270	average	4018.0	230.2	398.1	250.2	135.7	1703.3

	Fault detection success rate of k-suite < 1	Fault detection success rate of k-suite = 1
Distinguished mutants rate of k-suite< 1	**Type1**: not certainly detected by the k-suite while there are undistinguished mutants.	**Type3**: certainly detected by the k-suite while there are undistinguished mutants.
Distinguished mutants rate of k-suite= 1	**Type2**: not certainly detected by the k-suite while there is no undistinguished mutants.	**Type4**: certainly detected by the k-suite while there is no undistinguished mutants.

Fig. 3. Types of faults. If the fault detection success rate of the k-suite for a fault is equal to 1, it means that the k-criterion certainly detects the fault. If the distinguished mutants rate of the k-suite for a fault is equal to 1, it means that the k-criterion distinguishes all mutants generated from the faulty program.

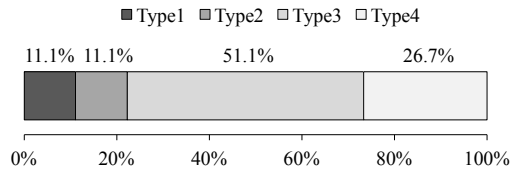

Fig. 4. The percentages of types of faults. Each percentage of a type represents the ratio of the number of faults for the type to the number of all faults. For example, the first box shows that 11.1% (5/45) of faults are type1.

proportion of faults that are certainly detected by the k-criterion. The sum of type2 and type4 faults implies the mutant distinguishment power of k-suites.

We provide the proportions of types in Figure 4. As a result, 11.1% of faults are type1 which means that there are rooms for improvement in 11% of faults. In other words, 11.1% of faults are not effectively detected by the k-criterion possibly because the k-criterion failed to distinguish mutants. We will carefully investigate the amount of improvements for the type1 faults in Section V-B.

Type2 faults are 11.1% of all faults. This means that distinguishing all mutants cannot guarantee to detect all faults. For example, the fault detection success rate of both the k-criterion and the d-criterion for Closure-107 (i.e., one of the type2 fault) is zero. While it is not in the scope of this work, a further investigation on this type of faults would be an interesting future work.

The sum of type3 and type4 faults are 77.8% of all faults, meaning that the k-criterion can detects 77.8% of the studied faults without explicitly considering the diversity of mutants. This accounts for why the k-criterion seems effective at detecting faults in many previous studies. Still, there is considerable room for improvement with the type1 faults, and it would be worthwhile to investigate the d-criterion to explicitly consider the diversity of mutants.

Note that the results given in Figure 4 are bounded by several experimental parameters including test data generation methods, studied faults, and mutation operators. However, the results clearly show that, simply by using a stronger adequacy criterion while using the same set of mutants and tests, it is possible to improve fault detection capabilities of mutation-adequate test suites.

TABLE II
EXPERIMENTATION RESULTS SUMMARY

Fault	Fault detection capability				Dist. mutants rate		Fault	Fault detection capability				Dist. mutants rate	
	d-suite	k-suite	p-value	OR	d-suite	k-suite		d-suite	k-suite	p-value	OR	d-suite	k-suite
Chart-5	0.976	0.970	0.728	1.248	1.000	1.000	Math-9	0.552	0.486	0.018	1.302	1.000	0.757
Chart-11	1.000	1.000	-	1.000	1.000	0.864	Math-14	0.172	0.166	0.400	1.043	1.000	1.000
Chart-12	0.808	0.040	0.000	98.25	1.000	0.533	Math-22	1.000	1.000	-	1.000	1.000	0.523
Chart-14	1.000	1.000	-	1.000	1.000	0.924	Math-27	0.432	0.328	0.000	1.557	1.000	0.834
Chart-15	1.000	1.000	-	1.000	1.000	0.712	Math-29	1.000	1.000	-	1.000	1.000	0.899
Chart-16	1.000	1.000	-	1.000	1.000	0.759	Math-35	0.774	0.754	0.772	1.117	1.000	1.000
Chart-17	1.000	1.000	-	1.000	1.000	1.000	Math-60	1.000	1.000	-	1.000	1.000	0.530
Chart-18	1.000	1.000	-	1.000	1.000	0.893	Math-61	1.000	1.000	-	1.000	1.000	0.664
Chart-22	1.000	1.000	-	1.000	1.000	0.980	Math-66	1.000	1.000	-	1.000	1.000	0.974
Chart-24	1.000	1.000	-	1.000	1.000	0.724	Math-68	0.184	0.162	0.179	1.166	1.000	1.000
Closure-56	1.000	1.000	-	1.000	1.000	0.850	Math-77	1.000	1.000	-	1.000	1.000	0.862
Closure-107	0	0	-	1.000	1.000	1.000	Math-90	0.594	0.374	0.000	2.444	1.000	0.755
Lang-12	1.000	1.000	-	1.000	1.000	0.776	Math-92	1.000	1.000	-	1.000	1.000	0.806
Lang-37	1.000	1.000	-	1.000	1.000	1.000	Math-93	1.000	1.000	-	1.000	1.000	1.000
Lang-41	1.000	1.000	-	1.000	1.000	1.000	Math-95	1.000	1.000	-	1.000	1.000	1.000
Lang-45	1.000	1.000	-	1.000	1.000	1.000	Math-98	1.000	1.000	-	1.000	1.000	1.000
Lang-56	1.000	1.000	-	1.000	1.000	1.000	Math-102	1.000	1.000	-	1.000	1.000	1.000
Lang-59	1.000	1.000	-	1.000	1.000	1.000	Math-103	1.000	1.000	-	1.000	1.000	1.000
Math-1	1.000	1.000	-	1.000	1.000	1.000	Math-104	1.000	1.000	-	1.000	1.000	0.437
Math-3	1.000	1.000	-	1.000	1.000	0.756	Time-8	1.000	1.000	-	1.000	1.000	0.909
Math-4	1.000	1.000	-	1.000	1.000	0.977	Time-9	1.000	1.000	-	1.000	1.000	0.799
Math-5	1.000	1.000	-	1.000	1.000	0.748	Time-13	1.000	1.000	-	1.000	1.000	0.864
Math-6	0.182	0.006	0.000	31.76	1.000	0.863	average	0.882	0.851	-	3.91	1.000	0.885

B. RQ2: Is the distinguishing mutation adequacy criterion more likely to detect faults than the traditional mutation adequacy criterion?

To investigate the difference of the fault detection capabilities between the d-criterion and the k-criterion, for each fault, we calculate the *percentage points (pp)* of the fault detection success rate of the d-suite in compared to the k-suite. For example, if the fault detection success rate of the d-suite for a fault is 90% and the fault detection success rate of the k-suite for the fault is 40%, then the increased fault detection success rate for the fault is 50 pp. The results for all faults, ranked in descending order, are presented in Figure 5.

In Figure 5, it is clear that the increased fault detection success rates for all faults never be negative. This means that the d-criterion is *always better than or equal to* the k-criterion in terms of fault detection capability. This is intuitive considering the subsumption relationship between the distinguishing and k-criterion as explained in Section III-B.

The top five faults (Chart-12, Math-90, Math-6, Math-27, and Math-9) in Figure 5 show that the d-criterion is statistically better than the k-criterion in terms of the fault detection capability, based on non-parametric proportion test with $\alpha = 0.05$. While Math-68, Math-35, Math-14, and Chart-5 also show increased fault detection success rates, their results are not statistically significant. Additionally, we provide effect sizes for the improvements of the fault detection capabilities in Table II. We calculate the *odd ratio* (OR) as the effect size because the fault detection result of a test suite for each fault is dichotomous [21]. If OR$= 1$, it means that the difference has no practical significance. The higher the OR value is, the stronger the association between the success of fault detection and the d-criterion in compared to the k-criterion becomes.

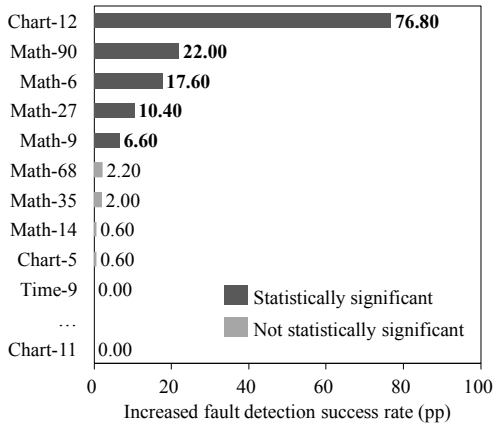

Fig. 5. Effect size of the d-criterion on fault detection. Each bar represents the *increased* fault detection success rate for each fault. The results are given in descending order. A fault with dark-grayed bar signifies that the fault detection success rate of the d-criterion is statistically greater than the fault detection success rate of the k-criterion (non-parametric proportion test, one-sided, $N = 500$, $\alpha = 0.05$) for the fault.

Interestingly, the five faults with the significant results exactly correspond to the five type1 faults. In other words, the d-criterion statistically significantly improves the fault detection capability for all type1 faults. It means that the d-criterion successfully improves the fault detection capabilities of adequate test suites whenever mutants are undistinguished

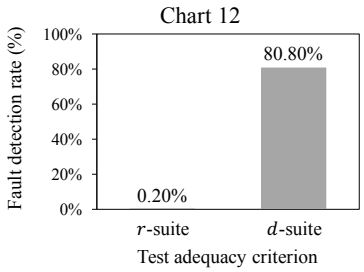

Fig. 6. Fault detection success rate of the r-suite and the d-suite for Chart-12. The r-suite is composed of randomly selected test suites whose size is equal to the average size of the d-suite.

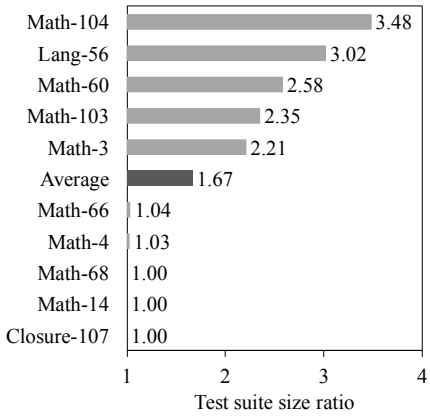

Fig. 7. Test suite size ratio of the d-suite to the k-suite for each fault. The results are given in descending order, while the average value is given instead of the results in the middle. In average, the d-criterion needs 1.67 times more tests than the k-criterion.

by the k-criterion.

In Figure 5, the result for Chart-12 (i.e., 76.80 pp) is especially surprising. We first investigate the influence of test set size. In Figure 6, we compare the fault detection rates of test suites d-suite and r-suite (i.e., 500 randomly selected test suites whose size is equal to the average size of d-suite). Note that the fault detection success rate of the r-suite is very smaller than the fault detection success rate of the d-suite while their test suite sizes are equal. This means that the fault detection effectiveness of the d-criterion is not because of the test suite size.

We also manually investigated the fault-detecting tests and kill information of all mutants with respect to the test pool for Chart-12. We found a general case where the d-criterion outperforms the k-criterion. Consider two distinguishable mutants m and m', we can think of two different test sets T_{kill} and T_{dist} as follows:

$$T_{kill} = \{t \in T \mid d(t, p_o, m) = 1 \vee d(t, p_o, m') = 1\}$$
$$T_{dist} = \{t \in T \mid d(t, p_o, m) \neq d(t, p_o, m')\}$$

In other words, T_{kill} is the set of tests that kills m or m', and T_{dist} is the set of tests that distinguishes m and m'. Interestingly, T_{dist} is the subset of T_{kill} by their formal descriptions. In this sense, if there exists a fault-detecting test t_{trig} in T_{dist}, it is more frequently selected from T_{dist} than T_{kill}. Note that the k-criterion considers only T_{kill} while the d-criterion considers not only T_{kill} but also T_{dist}. Thus, the d-criterion outperforms the k-criterion if $t_{trig} \in T_{dist}$ for two arbitrary mutants, and the amount of improvement increases as decreasing the size ratio of T_{dist} to T_{kill}.

C. RQ3: How many tests are needed to be a mutation-adequate test suite?

Considering all k-suites and d-suites for all faults, we provide the test suite sizes of k-suites and d-suites and their ratios in Table III. For each fault, each size value represents the average test suite size of the k/d-suite. For example, the test suite size of the d-suite for Chart-5 is 81.37. This means that 81.37 tests are selected to satisfy the d-criterion for Chart-5 in average. The test suite size ratio column represents the size ratio of the d-suite to the k-suite for each fault. Figure 7 shows the representative ratios in descending order.

For all faults, the average of the size ratios of d-suites to k-suites is 1.67. This means that the d-criterion needs 1.67 times more tests than required by the k-criterion in average. While the d-criterion needs more tests than the k-criterion, the improvement of fault detection effectiveness is not simply because of the larger test suite size. For example, there are only two fault-detecting tests in the test pool for Chart-12, which means that randomly adding more tests to the test suite would hardly improve the fault detection capability.

D. RQ4: How much time takes for selecting a mutation-adequate test suite?

Similar to the test suite sizes, we provide the test suite selection times of all k-suites and d-suites and their ratios in Table III. For each fault, the selection time of the k/d-suite is the average selection time of all test suites in the k/d-suite. For example, the selection time of the d-suite for Chart-5 is 33.56 milliseconds. This means that selecting a distinguishing mutation-adequate test suite takes 33.56 milliseconds in average. The selection time ratio column represents the selection time ratio of the d-suite to the k-suite for each fault. Figure 8 shows the representative ratios in descending order.

For all faults, the average of the selection time ratios of d-suites to k-suites is 5.74. This means that the d-criterion needs 5.74 times more selection time than the k-criterion in average. While the d-criterion takes more time for selecting an adequate test suite than the k-criterion, it is acceptable in a sense that each of distinguishing mutation-adequate test suites is selected within 0.078 seconds in average.

E. Threats to Validity

There may be several threats to validity for our empirical evaluations. One threat is due to the representativeness of the studied faults. While this threat can only be properly addressed by further study, we tried to use a non-trivial number of real faults collected in `defect4j` repository. Our results are also dependent on the test pool generated by `Randoop`. While another test pool generated by a coverage-aware test generation

TABLE III
EXPERIMENTATION RESULTS SUMMARY FOR TEST SUITE SIZE AND TEST SUITE SELECTION TIME

Fault	Test suite size			Selection time (m sec)			Fault	Test suite size			Selection time (m sec)		
	d-suite	k-suite	ratio	d-suite	k-suite	ratio		d-suite	k-suite	ratio	d-suite	k-suite	ratio
Chart-5	81.37	55.68	1.46	33.56	5.17	6.49	Math-9	21.15	12.59	1.68	2.53	0.75	3.37
Chart-11	9.76	6.83	1.43	1.28	0.39	3.30	Math-14	5.00	5.00	1.00	0.66	0.42	1.59
Chart-12	20.22	9.31	2.17	5.22	1.30	4.03	Math-22	52.09	23.69	2.20	6.00	0.46	12.98
Chart-14	74.03	61.04	1.21	18.75	11.03	1.70	Math-27	128.94	72.90	1.77	49.90	4.37	11.42
Chart-15	89.10	44.96	1.98	15.23	1.93	7.88	Math-29	64.11	46.94	1.37	7.95	1.56	5.11
Chart-16	118.79	77.66	1.53	26.85	2.35	11.45	Math-35	6.18	4.75	1.30	1.62	0.94	1.71
Chart-17	56.77	41.69	1.36	7.09	2.29	3.10	Math-60	50.31	19.47	2.58	16.18	1.27	12.73
Chart-18	54.07	37.04	1.46	3.92	0.83	4.71	Math-61	25.76	13.17	1.96	2.62	0.25	10.34
Chart-22	24.13	18.58	1.30	2.51	0.84	2.98	Math-66	3.00	2.90	1.04	0.42	0.32	1.30
Chart-24	18.93	10.18	1.86	0.24	0.05	4.74	Math-68	1.00	1.00	1.00	0.12	0.10	1.19
Closure-56	38.65	26.86	1.44	9.27	1.68	5.53	Math-77	140.25	96.38	1.46	9.66	2.49	3.87
Closure-107	2.00	2.00	1.00	0.96	1.39	0.69	Math-90	19.58	9.02	2.17	0.29	0.04	7.76
Lang-12	28.00	14.88	1.88	0.94	0.25	3.78	Math-92	168.07	120.02	1.40	5.00	1.43	3.49
Lang-37	450.18	291.36	1.55	185.93	19.98	9.31	Math-93	119.74	94.55	1.27	2.11	0.82	2.56
Lang-41	13.46	6.73	2.00	0.40	0.19	2.09	Math-95	25.56	13.18	1.94	0.72	0.11	6.75
Lang-45	80.84	41.40	1.95	30.03	3.74	8.03	Math-98	246.04	179.75	1.37	44.17	6.87	6.43
Lang-56	175.89	58.22	3.02	21.66	2.61	8.30	Math-102	38.14	29.30	1.30	4.69	1.03	4.56
Lang-59	176.69	132.40	1.33	13.11	4.76	2.75	Math-103	51.07	21.71	2.35	9.79	0.86	11.40
Math-1	137.96	96.53	1.43	13.08	2.83	4.62	Math-104	91.23	26.20	3.48	18.55	1.00	18.50
Math-3	294.12	132.97	2.21	59.58	3.95	15.07	Time-8	37.50	28.28	1.33	1.65	0.81	2.04
Math-4	5.00	4.86	1.03	0.57	0.57	1.01	Time-9	101.90	63.40	1.61	22.58	3.86	5.85
Math-5	127.37	63.51	2.01	11.39	1.53	7.44	Time-13	32.22	22.12	1.46	5.33	3.17	1.68
Math-6	13.39	9.92	1.35	1.92	0.76	2.51	average	78.21	47.80	1.67	15.20	2.30	5.74

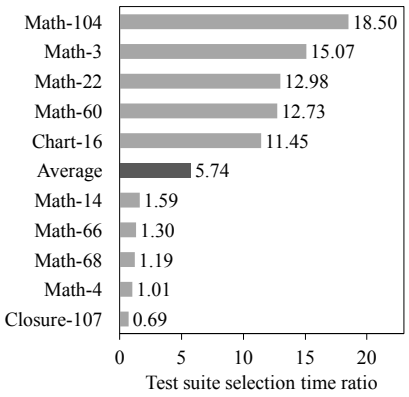

Fig. 8. Test suite selection time ratio of the d-suite to the k-suite for each fault. The results are given in descending order, while the average value is given instead of the results in the middle. In average, the d-criterion needs 5.74 times more time than the k-criterion.

tool, such as `EvoSuite` [22], may detect a different set of faults, we initialised our study with `Randoop` because we wanted to generate a large number of unbiased tests. In the future work, we will investigate different test pools using both `Randoop` and `EvoSuite`.

The fault detection capability of a test adequacy criterion may vary depending on the testing scenarios. For example, a test adequacy criterion may be used for not test suite selection but test data generation (e.g., Counter-exampled based test generation [23], [24] and search-based test generation [25], [26]). Further information about the testing scenarios and their impact on the fault detection capability is well-described in Zhu et al. [27]. We have a plan for developing a method for test generation for directly satisfy the d-criterion.

VI. RELATED WORK

Ammann et al. [28] recently discussed the notion of "undistinguished" mutants. They stated that, if two mutants are killed by precisely the same set of tests, the mutants are undistinguished, even though the mutants may involve different syntactic changes to p_o. We follow this concept to formally define the mutant distinguishment. However, they tried to remove such undistinguished mutants to establish minimal set of mutants, while we attempt to utilize the undistinguished mutants to improve the fault detection capabilities of mutation-adequate test suites.

Baudry et al. [29] studied the idea of distinguishment of execution traces to improve the fault localization effectiveness. They defined the concept of a Dynamic Basic Block (DBB) which is the set of statements that is covered by the same set of tests. A Large DBB implies low accuracy of fault localization since all statements in the DBB are equally suspicious as the faulty statement. They reported that optimizing a test suite to distinguish statements in a DBB leads the improvement of the fault localization accuracy. While both work use the concept of distinguishment in similar ways, we consider the mutant distinguishment to improve fault detection capability.

VII. CONCLUSION

This paper introduces a novel mutation-based test adequacy criterion called distinguishing mutation adequacy criterion based on the formal definition of the mutant distinguishment. The new adequacy aims to make adequate test suites capture the diversity of mutants.

We provide an empirical evaluation for the comparison of the distinguishing mutation adequacy criterion with the traditional mutation adequacy criterion in terms of their fault detection capabilities, test suite sizes, and test suite selection times. We use 45 real faults to study real worlds applications. The results show that the distinguishing mutation adequacy improves the fault detection success rate up to 76.8 percentage points compared to the traditional mutation adequacy, while the distinguishing mutation adequacy requires more tests and selection times to adequate test suites. In average, the distinguishing mutation adequacy requires 1.67 times and 5.74 times more tests and selection times compared to the traditional mutation adequacy, respectively.

While the cost of mutation is a long-stranding problem, we should not miss the fault detection capability as well. Since a mutation adequacy criterion is independent to the other mutation-related artifacts, studying stronger mutation adequacy is one promising way to improve the fault detection capability of mutation. We will provide more comprehensive investigations on the distinguishing mutation adequacy in the future work.

ACKNOWLEDGMENT

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No. R0126-15-1101, (SW Star Lab) Software R&D for Model-based Analysis and Verification of Higher-order Large Complex System)

REFERENCES

[1] T. Y. Chen, H. Leung, and I. Mak, "Adaptive random testing," in *Advances in Computer Science-ASIAN 2004. Higher-Level Decision Making*. Springer, 2004, pp. 320–329.

[2] D. Leon, A. Podgurski, and W. Dickinson, "Visualizing similarity between program executions," in *Proceedings of the IEEE 16th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2005, pp. 310–321.

[3] S. Yoo, M. Harman, P. Tonella, and A. Susi, "Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge," in *Proceedings of the 18th International Symposium on Software Testing and Analysis (ISSTA)*. ACM, 2009, pp. 201–212.

[4] N. Alshahwan and M. Harman, "Coverage and fault detection of the output-uniqueness test selection criteria," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*. ACM, 2014, pp. 181–192.

[5] R. Feldt, R. Torkar, T. Gorschek, and W. Afzal, "Searching for cognitively diverse tests: Towards universal test diversity metrics," in *Proceedings of the IEEE International Conference on Software Testing Verification and Validation Workshop (ICSTW)*. IEEE, 2008, pp. 178–186.

[6] R. Feldt, S. Poulding, D. Clark, and S. Yoo, "Test set diameter: Quantifying the diversity of sets of test cases," in *Proceedings of the IEEE 9th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, *to appear*.

[7] A. J. Offutt and R. H. Untch, "Mutation 2000: Uniting the orthogonal," in *Mutation testing for the new century*. Springer, 2001, pp. 34–44.

[8] A. J. Offutt and J. Pan, "Automatically detecting equivalent mutants and infeasible paths," *Software testing, verification and reliability*, vol. 7, no. 3, pp. 165–192, 1997.

[9] B. J. Grun, D. Schuler, and A. Zeller, "The impact of equivalent mutants," in *Proceedings of the International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2009, pp. 192–199.

[10] D. Shin and D.-H. Bae, "A theoretical framework for understanding mutation-based testing methods," in *Proceedings of the IEEE 9th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, *to appear*.

[11] R. Just, D. Jalali, and M. D. Ernst, "Defects4J: A database of existing faults to enable controlled testing studies for Java programs," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis (ISSTA)*. ACM, 2014, pp. 437–440.

[12] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on test data selection: Help for the practicing programmer," *Computer*, vol. 11, no. 4, pp. 34–41, 1978.

[13] M. Woodward and K. Halewood, "From weak to strong, dead or alive? an analysis of some mutation testing issues," in *Proceedings of the Second Workshop on Software Testing, Verification, and Analysis*. IEEE, 1988, pp. 152–158.

[14] L. Madeyski, W. Orzeszyna, R. Torkar, and M. Józala, "Overcoming the equivalent mutant problem: A systematic literature review and a comparative experiment of second order mutation," *IEEE Transactions on Software Engineering (TSE)*, vol. 40, no. 1, pp. 23–42, 2014.

[15] M. Papadakis, Y. Jia, M. Harman, and Y. Le Traon, "Trivial compiler equivalence: A large scale empirical study of a simple, fast and effective equivalent mutant detection technique," in *Proceedings of the IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE)*, vol. 1. IEEE, 2015, pp. 936–946.

[16] M. Kintis and N. Malevris, "Medic: A static analysis framework for equivalent mutant identification," *Information and Software Technology*, vol. 68, pp. 1–17, 2015.

[17] C. Pacheco and M. D. Ernst, "Randoop: feedback-directed random testing for Java," in *Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion*. ACM, 2007, pp. 815–816.

[18] R. Just, "The Major mutation framework: Efficient and scalable mutation analysis for Java," in *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*. ACM, 2014, pp. 433–436.

[19] A. Siami Namin, J. H. Andrews, and D. J. Murdoch, "Sufficient mutation operators for measuring test effectiveness," in *Proceedings of the 30th International Conference on Software Engineering (ICSE)*, 2008, pp. 351–360.

[20] P. G. Frankl and S. N. Weiss, "An experimental comparison of the effectiveness of branch testing and data flow testing," *IEEE Transactions on Software Engineering (TSE)*, vol. 19, no. 8, pp. 774–787, 1993.

[21] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proceedings of the IEEE/ACM 33rd International Conference on Software Engineering (ICSE)*. IEEE, 2011, pp. 1–10.

[22] G. Fraser and A. Arcuri, "Evosuite: automatic test suite generation for object-oriented software," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM, 2011, pp. 416–419.

[23] A. Gargantini and C. Heitmeyer, "Using model checking to generate tests from requirements specifications," in *Proceeding of the 7th European Software Engineering Conference Held Jointly with the 7th ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*. Springer, 1999, pp. 146–162.

[24] C. Cadar, D. Dunbar, and D. R. Engler, "Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs," in *OSDI*, vol. 8, 2008, pp. 209–224.

[25] P. McMinn, "Search-based software test data generation: a survey," *Software testing, Verification and reliability*, vol. 14, no. 2, pp. 105–156, 2004.

[26] M. Harman, Y. Jia, and W. B. Langdon, "Strong higher order mutation-based test data generation," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering (ESEC/FSE)*. ACM, 2011, pp. 212–222.

[27] H. Zhu, "A formal analysis of the subsume relation between software test adequacy criteria," *IEEE Transactions on Software Engineering (TSE)*, vol. 22, no. 4, pp. 248–255, 1996.

[28] P. Ammann, M. E. Delamaro, and J. Offutt, "Establishing theoretical minimal sets of mutants," in *Software Testing, Verification and Validation, IEEE 7th International Conference on*, 2014, pp. 21–30.

[29] B. Baudry, F. Fleurey, and Y. Le Traon, "Improving test suites for efficient fault localization," in *Proceedings of the 28th international conference on Software engineering*. ACM, 2006, pp. 82–91.