Isolating First Order Equivalent Mutants via Second Order Mutation

Marinos Kintis
Department of Informatics
Athens University of Economics and Business,
Athens, Greece
kintism@aueb.gr

Mike Papadakis
Interdisciplinary Center for Security, Reliability and Trust (SnT),
University of Luxembourg
Luxembourg, Luxembourg
michail.papadakis@uni.lu

Nicos Malevris
Department of Informatics
Athens University of Economics and Business,
Athens, Greece
ngm@aueb.gr

Abstract—**In this paper, a technique named I-EQM, able to dynamically isolate first order equivalent mutants, is proposed. I-EQM works by employing a novel dynamic execution scheme that integrates both first and second order mutation. The proposed approach combines the "impact" on the program execution of the first order mutants with the "impact" on the output of second order ones, to isolate likely to be first order equivalent mutants. Experimental results on a benchmark set of manually classified mutants, selected from real word programs, reveals that I-EQM achieves to classify equivalent mutants with a 71% and 82% classification precision and recall respectively. These results improve the previously proposed approaches by selecting (retrieving) a considerably higher number of killable mutants with only a limited loss on the classification precision.**

Keywords; Equivalent mutants; Higher Order mutation; Mutants' Impact;

I. Introduction

Among the various software development phases, testing usually constitutes more than half of what the overall process costs. Hence, researchers are motivated to automate all the software testing activities in order to reduce its expenses. Unfortunately, this is not possible for all software testing tasks. Since testing involves many "undecidable" problems [1] its full automation is impossible. Therefore, developing effective heuristics and achieving higher levels of automation for such problems is highly desirable.

Generally, the testing activity is performed by employing a set of test cases based on which the software's behavior is explored. In view of this, the testing process seeks to find a representative of the programs' behavior set of test cases. In practice, this task is accomplished by the test adequacy criteria. Testing criteria position a set of requirements to be exercised by the selected tests. These requirements provide practical solutions on the adequacy of testing i.e. when the testing process stops on the one hand, and how to achieve testing adequacy i.e. how to construct new tests on the other. Mutation testing, also called mutation analysis is a fault-based adequacy criterion. It operates by introducing faults, called mutants, into the program's code and by comparing the differences they produce on the program's output. Mutants are constructed by employing simple syntactic rules, one at a time or multiple ones together. In the former case the mutants are termed fist order mutants (*foms*) and in the latter one according to their order i.e. second (*soms*), third order etc, or generally higher order mutants (*homs*). The production of differences in the program's output by the introduced mutants, establishes the criterion requirements. The term killed is used to refer to a mutant producing different output than the original (not faulty) program when executed with a test. The term live refers to the opposite situation. Conversely, the tester's aim -criterion requirements- is to produce test cases capable of killing all introduced mutants. Unfortunately, not all mutants can be killed. Thus, there exist a lot of them, termed equivalent mutants, which are functionally equivalent with the original program and must be eliminated from the requirement set in order to assess the testing adequacy.

Mutation testing is considered to be a rather powerful method, able to subsume or probsubsume most of the structural testing criteria [2], [3]. Additionally, empirical studies suggest that mutation reveals more faults e.g. [4] than most of the structural testing criteria. In the studies made by Andrews et al. [5] and [6], it has been shown that mutants exhibit a similar behavior with real faults. Thus, if mutants can be killed, then programming faults might also be exposed. Despite its power, mutation lacks of use in practice. This maybe due to its excessive computation demands i.e. mutation testing requires a huge number of mutants to be executed with test cases. In attempts to overcome mutation computational demands, researchers have proposed the various mutation approximation techniques [1], [7], [8] such as: selective mutation [1], [7], [9] and mutant sampling [1], [7]. The main issue dealt by such approximation approaches is the production of small mutant sets with similar power as the initial ones. The use of mutation approximation approaches is essential in order to make mutation scalable. The Javalance framework [10], utilized in the present experimental evaluation, makes use of most of the proposed mutation cost reduction advances.

Equivalent mutant instances introduce further difficulties in the mutation testing process. These instances are analogous to the infeasible elements of structural testing [11]. However, the side effects caused by them are more serious. This is due to the number of introduced equivalent mutants which is considerably high compared to the structural infeasible elements. Equivalent mutant detection is essential in order to measure the adequacy of the performed testing activity. Adequacy, with respect to mutation testing, is measured by the ratio of: the number of killed mutants to the entire number of the introduced mutants reduced by the equivalent ones. This adequacy measure is called mutation score.

The utilization of a testing criterion requires automated processes in order to be practical. In view of this, mutation should employ automated tools for generating and executing test cases with mutants and suggesting the adequacy of the performed process. Of the abovementioned problems, the automated production of test cases and the adequacy evaluation i.e. the elimination of equivalent mutants, are difficult to handle "undecidable" problems [1], [11], [12].

Recent advances in the area [13], [14], [15], [16] have achieved to automate the test cases production quite effectively. However, equivalent mutants' identification still lacks of such approaches. In an attempt to overcome the difficulties caused by equivalent mutants Schuler and Zeller [17] proposed a technique able to isolate possible equivalent mutants. This approach is based on the observation that mutants affecting the program execution are more likely to be killable than those not affecting the program execution. Realizing this observation, mutants can be classified to those that affect the program's execution and to those that do not. In the study of Schuler and Zeller, the first category was found to be comprised of 75% of killable mutants. Thus, they concluded that practitioners should target this mutant set in order to restrict the side effects caused by equivalent mutants. Although, such an approach provides a high likely to be killable mutant set, it was empirically found that it consisted of 56% of the whole killable mutant set, thus, loosing a high number of killable mutants. The approach described by the present paper expands the Schuler and Zeller approach aiming to overcome the abovementioned limitation.

The proposed approach uses second order mutants in order to isolate possible first order equivalent mutants. The underlying idea is to determine the impact of a first order mutant on other, already killed mutants. Thus, it is argued that a killable mutant is likely to impact the output of another killable mutant when both mutants are put together i.e. forming a second order one. Although, such an approach is not better than the one of Schuler and Zeller in terms of classification recall and precision, it achieves to correctly classify different mutants. Hence, its combination with the one proposed by the Schuler and Zeller achieves to correctly classify 82% of the killable mutants with a precision of 71%.

Generally, the present work's contributions can be summarized to the following three points:

- A novel dynamic method to effectively classify first order mutations using second order mutants.
- An automated technique able to effectively classify mutations as being killable and equivalent. The proposed approach expands the method introduced by Schuler and Zeller [17] in order to classify a considerably higher number of mutations as killable.
- A case study indicating the effectiveness of the proposed approach. In particular the proposed approach achieves a 71% classification precision and 82% classification recall on killable mutants.

The rest of this paper is organized as follows: Section II presents the core concepts utilized by the present work. Section III and IV details the *I-EQM* method and discusses respectively the evaluation of the proposed approach. In Section V some related to the present work is presented. Finally, in Section VI conclusions and possible future directions are given.

II. Mutant Classification

This paper suggests the use of a mutant classification approach in order to isolate equivalent mutants. The proposed classification scheme utilizes second order mutations and the mutants' impact in order to highlight the majority of the killable mutants. This section addresses these techniques and concepts.

A. Mutation Analysis using classifiers

Applying first order mutation testing entails the generation of a mutant set, referred to as the candidate mutant set. Then, this candidate mutant set is executed with the employed tests in order to determine its adequacy. Those mutants that are killed, are removed from the candidate set of mutants. The remaining (live) mutants must then be analyzed in order to produce new tests able to kill them. This process (test case production and test case evaluation) iteratively continues until all non-equivalent mutants have been killed. In view of this, the ratio of the equivalent mutants to the candidate mutant set increases, as more tests are added to the considered test suite. This happens since the number of equivalent mutants remains constant while the number of killable ones decreases (since they are killed). Thus, if the live mutants could be automatically classified as killable and equivalent, one could claim substantial benefits by analyzing only the killable ones [17], [18], [19]. In such a situation two benefits arise. First, an accurate adequacy evaluation is employed. Second, the test generation process can be effectively guided by only those killable mutants saving considerable resources during the process of trying to kill those non killable mutants.

Based on the above arguments, automated mutant classification approaches have been proposed in the literature. A typical mutant classification process, steps a-c of Fig. 1, involves the categorization of the set of live mutants into two disjoint sets; the possibly killable mutants and the possibly equivalent ones. Before applying the classification scheme, the set of live ones must be found, which means that the first order mutants of a program under test must be generated (Fig. 1 (a)) and executed with the available test suite (Fig. 1 (b)). At this point a set of killed mutants and a set of live mutations will be created. The latter set will be provided as input to the classification system and will be categorized accordingly (Fig. 1 (c)). In order for a mutant classification approach to be practical it must satisfy two conditions:

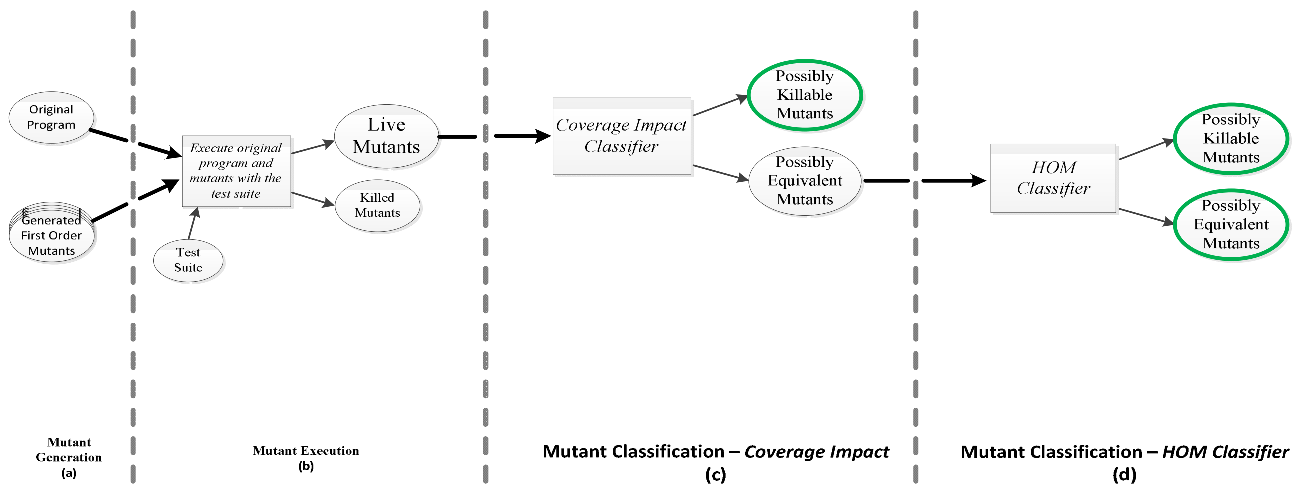
Original Program
Generated First Order Mutants
Execute original program and mutants with the test suite
Test Suite
Live Mutants
Killed Mutants
Coverage Impact Classifier
Possibly Killable Mutants
Possibly Equivalent Mutants
HOM Classifier
Possibly Killable Mutants
Possibly Equivalent Mutants
Mutant Generation (a)
Mutant Execution (b)
Mutant Classification – Coverage Impact (c)
Mutant Classification – HOM Classifier (d)

Figure 1. Mutant Classification Process. The live mutants will be classified to the sets of Possibly Killable and Possibly Equivalent Mutants. The *I-EQM* process works in two phases. First, the live mutants will be classified via the *Coverage Impact Classifier* and subsequently the produced Possibly Equivalent Mutant set will be classified by the *HOM Classifier*. The highlighted first order mutant sets are the outcome of the *I-EQM* classification scheme.

1. High Precision: *The precision metric*[1] *quantifies the ability of the classifier to categorize correctly the killable mutations. A high score indicates that the classifier can sufficiently distinguish between killable and equivalent mutants.*
2. High Recall: *The recall metric*[2] *measures the capability of the classifier in retrieving killable mutants. A high ratio shows that the classification scheme is able to recover the majority of the live killable mutants.*

Generally, high precision is difficult to be achieved nevertheless, extremely desirable for killing and analyzing killable mutants. However, if high precision is not accompanied by a relatively high recall, the process might lose some rather valuable mutations. Since low recall indicates that many killable mutants are ignored, such processes will experience losses of their strength. On the contrary, high recall solely is easily achieved by classifying most or all undetected mutants as killable. Therefore, in such a case higher testing quality is achieved with considerable cost. Thus, a combination of high precision and high recall is more suitable for a mutant classification scheme. In this manner, most of the mutants are classified as potentially killable and are indeed non-equivalent ones and at the same time testing based on the retrieved non-equivalent mutants is deemed sufficient. In a different situation the classifier would be deficient as it would categorize many equivalent mutants as possibly killable ones (a case of a classifier with a low precision) or it would classify many non-equivalent mutants as possibly equivalent ones (a classifier with a low recall). Consequently, it is the reconciliation of a classifier's precision and recall scores that stipulates its success.

[1] *Precision = (True Killable) / (True Killable + False Killable)*, where "True Killable" denotes the number of the correctly classified non-equivalent mutations and "False Killable" the number of the incorrectly classified equivalent ones.

[2] *Recall = (True Killable) / (True Killable + False Equiv)*, where "True Killable" denotes the number of the correctly classified non-equivalent mutations and "False Equiv" the number of the incorrectly classified ones.

B. The mutants' impact

Mutant classification has been attempted in the literature [17], [18]. The classification procedure is usually based on a set of mutants' properties and must unambiguously categorize mutants in the available categories. Properties such as the impact of mutants in dynamic program invariants, on the program execution traces and on return values have been proposed [17], [18], [19]. Specifically, impact measures the differences in behavior between the original and mutant programs. Such differences appear during the program executions and between the execution of the mutated program location to the programs' output [19].

Along these lines, impact on coverage [17] measures the differences of the coverage in both the original and mutated program versions. In the approach taken in [17], also adopted here and referred to as the *Coverage Impact Classifier*, the executed statements were counted along with the number of times they were executed constituting a coverage measure. A comparison of the coverage measures of both the original and mutated program executions results in the impact measure (coverage difference) of the examined mutant [17]. Impact on return values, measures the differences in the values returned by the encountered public methods [17]. Impact on dynamic program invariants measures the number of invariants that were violated by the introduction of mutants [18].

Generally, it has been empirically found that mutants with impact are more likely to be killable than those with no impact regardless the impact measure [17]. However, different impact measures or their combinations generally result in different mutant classifiers with variations on their effectiveness. Constructing a more effective classifier forms the objective of the present paper, which proposes the use of higher order mutants as impact measures. Based on this novel measure, different mutants with the abovementioned approaches can be classified.

C. *Higher order mutation*

Considering multiple mutants at the same time has long been identified as an issue of the mutation analysis research. DeMillo et al. [20] proposed the Coupling Effect as "Test data that distinguishes all programs differing from a correct one by only simple errors is so sensitive that it also implicitly distinguishes more complex errors". This definition was later extended by Offutt [21] as the Mutation Coupling Effect Hypothesis, where first order mutants were defined as simple faults whereas higher order ones as complex. In view of this, empirical evidence was provided and showed that tests able to kill first order mutants are also capable of killing over a 99% of second and third order ones [21]. As a consequence, the considered mutants were limited only to first order ones.

Recently, Jia and Harman [22] suggested using higher order mutation as a possible answer to the difficulties faced by mutation testing. According to their study, there are a few but extremely valuable *homs*, named as "subsuming" *homs*. These mutants are harder to kill than most of the *foms* and thus, one should aim at them only, ignoring most of the *foms*. In other studies [8], [23] it was shown that sampling second order mutants based on various strategies produces tests with a 10% loss on the fault revealing ability while reducing the number of the produced equivalent mutants by a 80-90%. In this study, second order mutants were employed in order to provide information about the first order ones they are composed of. Thus, possible first order equivalent mutants can be identified by observing the behavior of the second order ones.

D. *The Javalanche tool*

The present study constitutes an extension of the Schuler and Zeller work [17] on the equivalent mutant classification. In order to provide fairly comparable results the same mutant sets and tools as the Schuler and Zeller study were used. Thus, the present study utilizes the Javalanche tool [10].

Javalanche [10] is a publicly available mutation testing tool for Java programs. It enables the efficient application of the mutation process by implementing a large number of optimizations in order to be scalable and practical to real world programs. Additionally, Javalanche has also been employed in many recent studies such as [15], [17], [24].

Table I records details about the utilized mutant operators supported by the tool and the present study. These mutant operators are based on the suggestions made by Offutt et al. [9] on mutant sufficiency.

TABLE I. MUTANT OPERATORS UTILIZED BY JAVALANCHE

Mutant Operators	**Description**
replace numerical constant	Replaces a numerical constant instance by a value + 1, a value -1 or by 0.
negate jump condition	Inserts the negation operator to logical conditional instances.
replace arithmetic operator	Replaces an arithmetic operator instance by another one.
omit method calls	Omits a method call and sets in its possition a default value (the default value repalces the returned one).

III. ISOLATING EQUIVALENT MUTANTS: *I-EQM*

The primary purpose of this paper is the introduction of a new mutant classification scheme, hereafter referred to as *HOM Classifier*, which would further attenuate the effects of the equivalent mutant problem. The salient feature of the suggested approach is the employment of higher order mutation in the classification process.

A. *First order Mutant Classification via Second order Mutation*

HOM Classifier categorizes mutants based on the impact they have on each other. In view of this, it produces pairs of mutants composed of the examined (first order) mutant with others. The classifier works based on the intuition that since equivalent mutants have a small effect on the state of the program, they should not have an apparent impact on the state of another mutant. Hence, a possible equivalent mutant will have a minor impact on the execution and no observable impact on the output of another mutant program. This leads to the *HOM Classifier* hypothesis.

1) *Classifier Hypothesis*

Let *fom* be a first order mutant, *umut* an unclassified mutant and *umut.fom* the second order mutant created by the combination of the two corresponding mutants. The *HOM Classifier* hypothesis states that the results of the execution of the first order mutant *fom* and the execution of the second order mutant *umut.fom* should not differ. More formally,

$$outputOf(fom, test) \neq outputOf(umut.fom, test) \Rightarrow$$

$$umut \in Killable$$

If the above condition holds for a mutant in the mutant set then the *umut* mutant is classified as possibly killable. In a different case, *umut* is classified as possibly equivalent. This practice is presented in Fig. 2. Although this condition may not always hold, the present study suggests that it can provide substantial guidance for identifying killable and equivalent mutants.

2) *Mutant Classification Process*

The mutant classification process of the *HOM Classifier* requires three main inputs. These are requisites for the evaluation of the *HOM Classifier* hypothesis condition.

- The first input is the set of the live-unclassified mutants that need to be classified. The mutants of this set will be categorized as possibly killable or possibly equivalent based on the truth or the falsehood of the classification's predicate.
- The second required input is the set of mutants, referred to as the classification mutant (CM) set that will be used for constructing the sought mutant pairs. The mutant pairs are constructed by combining these mutants with the unclassified ones.
- The final input to the classification process is the test suite, with which the second order mutants and their corresponding first order ones will be executed.

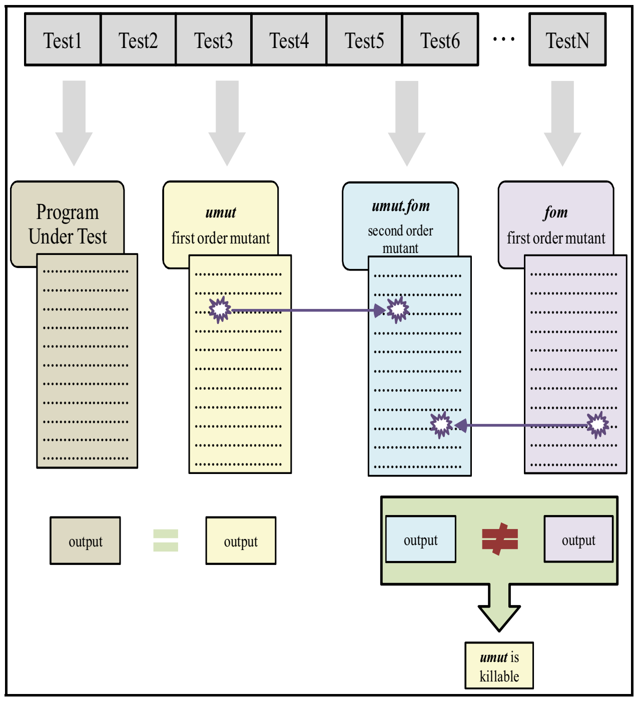

Figure 2. The unclassified first order mutant ***umut*** is classified as possibly killable based on its impact on other ***fom*** mutants

Finally, the algorithm pertaining to the abovementioned classification process is presented in Fig. 3. The algorithm takes as inputs a) the set of the live-unclassified first order mutations, b) the set of the first order mutants of the classification mutant set that will be used for the generation of the second order ones and c) the available test suite. Subsequently, for every live mutant the truth value of the *HOM Classifier* hypothesis condition is evaluated. If the condition holds for every generated second order mutation then the corresponding live mutant is classified as possibly equivalent, otherwise it is classified as possibly killable.

3) *Classification Mutant (CM) set*

Generally, the proposed classification technique determines the impact of mutants using other mutations referred to as the classification mutant (CM) set. Consider Fig. 2, the CM set is the set of the *fom* mutants to be employed in order to perform an effective classification process. The question that is raised here is which mutants are suitable for supporting the mutant classification. Specifically, it has been found that not all mutations are of the same value in assessing the mutants' impact. Perhaps, by utilizing all the available mutants one could get the best results. However, such an approach constitutes a very expensive one, since it requires executing all mutants with all tests for each mutant to be classified.

Considering the above issue a necessary restriction on the utilized CM mutant set was made; this set was composed of the mutations that appear in the same class with the aimed unclassified mutant. Thus, for each mutant to be classified, a different CM set was considered. This choice was based on the intuition that mutations appearing in the same class are more likely to interact and be exercised by the same test cases. Here it is noted that since there is no test able to kill *umut*, if there is no interaction between the mutants composing the second order one (*umut* and *fom*), then the outputs of *fom* and *umut.fom* will be the same for all the employed tests. Additionally, choosing a CM set composed of the mutations that appear in the same class seems natural as it is in accordance with the aimed level of testing which is the unit one. Nevertheless, the optimal choice of the CM set constitutes an issue not addressed by the present paper and forms a matter open for further research.

B. *HOM Classifier Variations*

The work presented here considers two variations of the *HOM Classifier*, namely *HOM Classifier(all foms)* and *HOM Classifier(killed foms)*. These two approaches differ in the sets of mutants that will be used for the creation of the second order ones. Recall, that the employed CM set is the one composed of the mutations that appear in the same class with the aimed unclassified mutant. The first approach, considers the whole CM set whereas the second one utilizes the subset of CM containing only those mutants that have been killed by the employed tests. The reason of using only the already killed mutants is that since these mutants are more sensitive to the utilized tests (easy-to-kill) than the live ones, they will be more sensitive to the impact of the examined ones too. Besides the abovementioned reason, this argument was empirically found to be sound. Furthermore, constructing the sought second order mutants based on the killed mutant set results in reducing the overall computational cost of the technique due to the reduced number of mutant combinations and their respective executions.

```
1.  for each first order mutant umut in the live-unclassified set
2.    for each first order mutant fom in the CM set
3.      for each test case test in the test suite set
4.        if the result of the execution of fom with test does not exist
5.          execute fom with test and record the fom output
6.        end if
7.        generate the second order mutant umut.fom
8.        execute umut.fom with test and record the umut.fom output
9.        compare the ouputs of fom and umut.fom
10.       if the ouputs differ
11.         add umut to the set of "Possibly Killable Mutants"
12.         continue to the next mutant of the live-unclassified set
13.       end if
14.     end for each
15.   end for each
16.   add umut to the set of "Possibly Equivalent Mutants"
17. end for each
18. return the set of "Possibly Killable Mutants"
19. return the set of "Possibly Equivalent Mutants"
```

Figure 3. HOM Classification process

C. *I-EQM Mutant Classification*

In addition to the previously described techniques, the possibility of employing a combined strategy is also investigated. To this end, the *I-EQM* classification scheme is proposed. The *I-EQM Classifier* constitutes a combination of the *Coverage Impact Classifier* proposed by Schuler and Zeller [17] with the two variations of the *HOM Classifier*. The utilization of the *Coverage Impact Classifier* is based on its evaluation, which gave the best results among those examined by the Schuler and Zeller study [17]. Specifically it resulted in a classification precision of 75% and a recall value of 56%. Since the corresponding precision measure of the *Coverage Impact Classifier* can be judged as an accurate one, i.e. it does not misclassify many equivalent mutants; effort should be put upon improving the recall measure. Achieving higher recall results in considering a larger number of killable mutants strengthening the testing process.

In order to effectively combine different classifiers they must adhere to the following two requirements. First, the precision of both classification schemes must be reasonably high. Second, they should classify different non-equivalent mutants as possibly killable. In other words, both produced sets must be accurate and their intersection must be of minimum size. That way, the precision of the combined classifier will not be greatly affected, whereas its recall will be significantly improved.

The *I-EQM* classification process is summarized in Fig. 1. After obtaining the set of Live Mutations (Fig. 1 (a), (b)), the *Coverage Impact Classifier* is employed in order to perform the first step of the *I-EQM*'s classification procedure (Fig. 1 (c)). This step will produce the set of the possibly killable and the possibly equivalent mutants of the *Coverage Impact Classifier*. Next, the *HOM Classifier* is applied to the possibly equivalent mutant set that was previously generated (Fig. 1 (d)). This phase will create the set of the potentially killable mutants and the set of potentially equivalent ones of the *HOM Classifier*. The *I-EQM Classifier*'s resulting set of possibly killable mutants is the union of the possibly killable mutant set of the *Coverage Impact Classifier* and the corresponding killable set of the *HOM Classifier*. The possibly equivalent mutant set of *I-EQM Classifier* is the one generated by the *HOM Classifier*. The aforementioned sets appear highlighted in Fig. 1.

IV. Empirical Study

The present study investigates a new aspect of higher order mutants, their ability to classify first order ones. This attribute constitutes the basis of the *HOM* and the *I-EQM Classifiers,* described in Section III. The present section describes the empirical evaluation of these approaches and directly compares them to the state of the art, *Coverage Impact Classifier,* proposed by Schuler and Zeller [17]. Further, an additional attempt to quantify the benefits of each of the examined classifiers is presented.

A. *Research Objectives*

The primary purpose of the presented empirical evaluation is summarized by the following research points:

- *Can second order mutation provide adequate guidance in equivalent mutant isolation, i.e. what is the respective recall and precision of the HOM and I-EQM Classifiers?*
- *How is the categorization ability of HOM and I-EQM Classifiers compared to the Coverage Impact Classifier?*

B. *Subject programs and mutants*

The evaluation of the introduced mutant classification schemes is based on a set of seven open-source projects. The motivation behind their selection is twofold. Primary, some of these projects have been utilized in various mutation testing studies such as [15], [17], [18] and thus, they can be considered as benchmarks. Moreover, the study of Schuler and Zeller [17] considers the same program set, hence, a comparison between their approach and the ones proposed in the present paper can be directly performed. Secondly, these programs constitute real-world examples and therefore they would provide valuable insights about the practical applicability and the efficacy of the examined mutant classification techniques.

In order to investigate the effectiveness of the proposed approaches, the provided mutant set by Schuler and Zeller [17] was used, hereafter referred to as the "control mutant set". This set of mutants comprises of 140 manually classified mutations each one belonging to a different class of the subject programs, with a total number of 20 mutations per test subject.

Details about the subject program names, their respective description, program versions, lines of code and accompanied test cases are recorded in Table II in the first five columns respectively. The column "Considered Generated Mutants" of Table II refers to the number of considered mutants, among those generated by the Javalanche framework. It must be mentioned that the considered mutants are those employed by the present experiment (see Section III for details) and belong to the same classes with the examined mutants (manually classified mutations). The first sub-column presents the total number of the considered mutations, whereas the second one the number of mutations that are covered by the available test cases. Finally, the column "Manually Classified Mutations" presents the number of the equivalent and non-equivalent mutants among the manually classified ones.

C. *Experimental Setup*

In order to deal with the aforementioned research points, the recall and precision values of the examined techniques must be determined. To this end, the *HOM, I-EQM* and *Coverage Impact* Classifiers are applied to the control mutant set with the aim of classifying them as potentially killable or potentially equivalent ones.

The experiment uses the Javalanche mutation testing framework in order to produce and execute the sought mutants. Javalanche was employed to generate the first order mutants belonging to the classes of the control mutant set.

TABLE II. SUBJECT PROGRAM DETAILS

Subject Programs	Short Description	Version	Lines of Code	Test Cases	Considered Generated Mutants		Manually Classified Mutations	
					Number of Mutations	*Covered Mutations*	*Non-Equivalent*	*Equivalent*
ASPECTJ[a]	AOP extension to Java	cvs: 2007-09-15	25,913	336	6613	2878	15	5
BARBECUE	Bar code creator	svn: 2007-11-26	4,837	153	3283	1603	14	6
COMMONS	Helper utilities	svn: 2009-08-24	19,583	1608	8693	8305	6	14
JAXEN	XPath engine	svn: 2008-12-03	12,438	689	6619	3944	10	10
JODA-TIME	Date and time library	svn: 2009-08-17	25,909	3497	5322	4197	14	6
JTOPAS	Parser tools	1.0 (SIR)	2,031	128	1676	1402	10	10
XSTREAM	XML object serialization	svn: 2009-09-02	16,791	1122	2156	1730	8	12
TOTAL	-	-	-	-	**34362**	**24059**	**77**	**63**

a. Only the *org.aspectj.ajdt.core* package was considered.

These mutants were then executed with the available test cases[3]. Mutant execution settings were set as: a) 100 seconds for timeout limit[4] and b) execution of all tests with all mutants. Since Javalanche does not support second order mutation, second order mutants were generated as follows: a) each of the examined mutants in the control mutant set was introduced manually. This resulted in 140 different class versions. b) Javalanche was employed to produce mutations for each of these 140 classes. Mutants belonging to the same position[5] with the examined one and produced by the same mutant operator with the examined mutant were discarded from the considered mutation set. Should such an action not be applied, the examined mutant would have been replaced by a discarded one making it impossible to assess its impact. Based on this process, all the required mutant pairs, i.e. second order mutants, were generated. Each pair is composed of the examined mutant and another one belonging to the same class. Finally, the *HOM* classification process, as presented in Fig. 3, was performed.

In summary for each mutant of the control mutant set the following procedure was followed:

- The first order mutants of the appropriate class were generated and executed via the available test cases.
- The appropriate second order mutants were generated and executed via the available test cases.
- The execution results of the generated first order mutants and the respective second order ones were compared in order to classify the examined mutation.

The above process was performed for the *HOM* and *I-EQM Classifiers* for both of their respective variants i.e. *all foms and killed foms.* Recall, that these approaches differ on the set of second order mutants that they rely on. For the case of *I-EQM Classifier*, presented in Fig. 1, the *Coverage Impact Classifier* classifies the control mutants as killable and equivalent ones. Those classified as equivalent are then categorized based on the *HOM Classifier*.

[3] The same tests as with the Schuler and Zeller study were employed

[4] To avoid discrepancies such as endless loops caused by the introduced mutants, a timeout limit was set. If mutant execution time exceeds this limit the mutant is treated as killed.

[5] Position refers to the exact part of the original source code that differs from the mutant programs

A comparison between mutant classifiers was attempted based on the accuracy and the F-measure scores, metrics usually used in comparing classifiers in information retrieval experiments [25]. These measures were utilized to validate in a more typical way the differences between the classifiers. Specifically, F-measures with β equal to 1, 2 and 0.5 were employed. Here it should be noted that high recall values indicate that more killable mutants are to be considered, hence leading to a more thorough testing process. High precision indicates that less equivalent mutants are to be examined, leading to an efficient process. By using β equal to 1, a comparison scenario of equal importance between the recall and precision is adopted. While, the use of β equal to 2 or 0.5, indicates a scenario where recall is of higher or lower importance than the precision value, respectively.

In the present experiment special care was taken to handle some specific cases due to some inconsistencies of the utilized tool. Specifically, it was observed that in the cases of execution timeouts the tool gave different results when some mutants are executed in isolation than together with others. To circumvent this problem, when mutants are categorized as killable due to timeout conditions (if either the first order mutant or the second order one results in a timeout) the corresponding mutants of the considered mutant pair were isolated and re-executed individually with a necessary timeout limit. Other special considerations include issues concerning the comparison of the programs' output. Note, that such a comparison is performed between every first order mutant and its respective second order one. Many programs had outputs dependent on each specific execution. Execution outputs containing time information, e.g. Joda-Time and AspectJ test subjects, or folder locations, e.g. the JTopas program, are some examples of such cases. To effectively handle these problematic situations, the execution dependent portions of the considered outputs were replaced with predefined values via the employment of appropriate regular expressions.

D. Empirical results

The respective results of the *HOM Classifier* are recorded in Table III. The table presents the respective classification precision and recall values of the method's variations per subject program. The columns named

"Killable" and "Equivalent" correspond to the sets of the possibly killable and equivalent mutants respectively. Their sub-columns present details about the number of the correctly and incorrectly classified mutations. The first figure of each cell refers to the *HOM Classifier (all foms)* variation, whereas the second one to the *HOM Classifier (killed foms)* variation. The last column presents the precision and the recall metrics per subject. On average, the *HOM Classifier (all foms)* achieves a precision value of 69% and a recall value of 57% and the *HOM Classifier (killed foms)* variation realizes a precision of 70% and a recall of 55% respectively.

The *I-EQM Classifier*'s experimental evaluation is depicted in Table IV (the same structure with Table III is used), which presents details about the precision and the recall metrics of the *I-EQM*'s variations for each test subject. The average precision of the *I-EQM*'s variation that employs the *HOM Classifier (all foms)* method is 70% and its recall is 83%. The corresponding values of utilizing the *HOM Classifier (killed foms)* variation are 71% and 82% respectively.

These empirical results provided evidence that the *HOM Classifier* hypothesis is an appropriate mutant classification property. Therefore, the employment of second order mutation can be beneficial in isolating equivalent mutants. Additionally, the results of the *HOM Classifier*'s variations indicate that the utilization of only the killed first order mutants as the classification mutant set (CM set) achieves approximately the same classification effectiveness as the employment of all the generated first order mutants. With this fact evident, since the *HOM Classifier (killed foms)* is more efficient than *HOM Classifier (all foms)* its use is advisable. Finally, compared to the *Coverage Impact Classifier*, which achieves a precision of 75% and a recall of 56%, the *HOM Classifier* attains similar recall values, with a lower precision, indicating that the *Coverage Impact Classifier* is a better one.

Considering the *I-EQM Classifier* results it is evident that it forms an effective combinatory strategy. It achieves to retrieve more that 81% of the unclassified non-equivalent mutants with a reasonably high precision (71%). This high retrieval capability is attributed to the ability of the *HOM Classifier* to classify different non-equivalent mutants than the *Coverage Impact Classifier.* As a consequence, their combination enhances the corresponding recall value by nearly 30%, meaning that approximately 30% more killable mutants are to be considered. Concisely, the *I-EQM* method achieves a superior recall value, while attaining a small loss of its precision. In particular, the *I-EQM* technique realizes a gain of 26% over the *Coverage Impact*'s recall metric for a loss of 4% on its precision.

The comparison results based on the accuracy and the F-measures between the examined approaches are given in Fig. 4. In this figure, the "All Mutants" refers to a naive classifier that categorizes all mutants as killable. In such a case, it achieves a 100% recall and 55% precision. The *I-EQM* and *HOM* methods refer to their respective *killed foms* variation method. It is noted that the *all foms* and *killed foms* variations were found to have similar results. From the findings of Fig. 4 referring to the accuracy measure, it can be observed that the *I-EQM* classification technique is the most accurate of all, followed by the *Coverage Impact*, the *HOM* and the *ALL Mutants* classifiers. Regarding the F-measure, three possible scenarios are examined. Note that high recall values indicate a more thorough testing process, while high precision a more efficient one. The first scenario refers to the case where a balanced importance between the recall and precision metrics is desirable. For this scenario, represented by the F1 measure, it can be seen that the *Coverage Impact* is better than the *HOM Classifier* but worse than the *I-EQM* and the *ALL Mutants*. Conversely, *I-EQM* is better than the *ALL Mutants* indicating that it is more suitable for this particular case. The second scenario, examined by the F2 measure, emphasizes on the recall value. It can be seen that the *I-EQM* classification approach achieves by far better results than the *Coverage Impact* and *HOM* techniques but worse than the *ALL Mutants* classifier. This is expected, since the *ALL Mutants* categorizes all examined mutations as killable.

TABLE III. MUTANT CLASSIFICATION USING *HOM CLASSIFIER* (DENOTING: ALL FOMS – KILLED FOMS)

Subject Programs	Killable		Equivalent		Possibly Killable	
	True Killable	*False Equiv*	*True Equiv*	*False Killable*	*Precision*	*Recall*
ASPECTJ	8 – 8	7 – 7	2 – 2	3 – 3	73% –73%	53% – 53%
BARBECUE	9 – 7	5 – 7	5 – 5	1 – 1	90% – 88%	64% – 50%
COMMONS	4 – 4	2 – 2	10 – 10	4 – 4	50% – 50%	67% – 67%
JAXEN	6 – 6	4 – 4	7 – 7	3 – 3	67% – 63%	60% – 60%
JODA-TIME	9 – 9	5 – 5	4 – 5	2 – 1	82% – 90%	64% – 64%
JTOPAS	3 – 3	7 – 7	10 – 10	0 – 0	100% – 100%	30% –30%
XSTREAM	5 – 5	3 – 3	5 – 6	7 – 6	42% – 45%	63% – 63%
TOTAL	**44 – 42**	**33 – 35**	**43 – 45**	**20 – 18**	**69% – 70%**	**57% – 55%**

TABLE IV. MUTANT CLASSIFICATION USING *I-EQM CLASSIFIER* (DENOTING: ALL FOMS – KILLED FOMS)

Subject Programs	Killable		Equivalent		Possibly Killable	
	True Killable	*True Equiv*	*True Equiv*	*False Killable*	*Precision*	*Recall*
ASPECTJ	14 – 14	1 – 1	0 – 0	5 – 5	74% – 74%	93% – 93%
BARBECUE	11 – 10	3 – 4	5 – 5	1 – 1	92% – 91%	79% – 71%
COMMONS	4 – 4	2 – 2	7 – 7	7 – 7	36% – 36%	67% – 67%
JAXEN	8 – 8	2 – 2	6 – 6	4 – 4	67% – 67%	80% – 80%
JODA-TIME	13 – 13	1 – 1	4 – 5	2 – 1	87% – 93%	93% – 93%
JTOPAS	8 – 8	2 – 2	10 –10	0 – 0	100% – 100%	80% – 80%
XSTREAM	6 – 6	2 – 2	3 –4	9 – 8	40% – 43%	75% – 75%
TOTAL	**64 – 63**	**13 – 14**	**35 – 37**	**28 – 26**	**70% – 71%**	**83% – 82%**

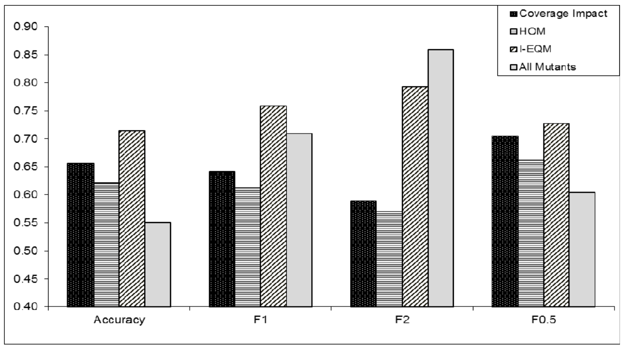

Figure 4. Mutant classifiers comparison

Finally, the last scenario, which limits the selection of equivalent mutants, weighs the precision metric higher than the recall one. For this case, studied by the F0.5 measure, the *I-EQM* classifier achieves better results than the rest classification approaches.

Conclusively, the *I-EQM* classification method provides better results than the *Coverage Impact* one with respect to the accuracy and all the considered F-measures. Therefore, it can be argued that the *I-EQM* constitutes a better mutant classifier than the rest of the examined ones.

V. Related Work

Isolating possible equivalent mutants with the aim of reducing their effects in the testing process dynamically, is a relatively new direction of the mutation testing research. One of the first attempts aiming at this issue is due to the work of Adamopoulos et al. [26]. In their attempt, genetic algorithms were employed in order to overcome the difficulties of the large numbers of the introduced mutants and equivalent ones. Their results suggested that it is possible to generate a small set of killable mutants, valuable to the testing process. However, contrary to the present approach this technique attempts to produce killable mutants and not to isolate equivalent ones in order to help assessing the test adequacy.

Measuring the mutants' impact as the way to assess mutations has been proposed by Schuler et al. [18] with the use of dynamic program invariants. In their study it was found that when mutants break dynamically generated invariants are more likely to be killable. The use of coverage impact as an assessment measure of mutations, was initially suggested by Grun et al. [19] and later extended by Schuler and Zeller [17]. Empirical evaluation [17] of the aforementioned approaches based on the Javalance [10] tool suggested that coverage impact is more efficient and effective at classifying killable mutants. Therefore, the proposed by the present paper approach *I-EQM* uses and extends the coverage impact approach with the aim of ameliorating the method's recall. Details about the coverage impact approach have been given in Section II.

Determining the program equivalence, such as the equivalent mutant identification, has been shown to be an "undecidable" problem in general [12]. However, approaches able to detect some of them do exist [27], [28]. In such an attempt Baldwin and Seyward [27] suggested the use of compiler optimization techniques. Since program optimization produces equivalent program versions, the original and the mutant programs can be optimized or de-optimized and eventually identify equivalent mutants. Offutt and Craft [28] developed such techniques and based on their empirical evaluation found that a 10% of the existing equivalent mutants could be automatically detected. This approach was later extended by Offutt and Pan [11] using path analysis and a constraint based testing technique. In their evaluation Offutt and Pan report that they achieved to detect on average the 45% of the existing equivalent mutants. Other related approaches make use of program slicing [29] or dependence analysis [30] to aid the tester identify equivalent mutants. All the equivalent mutant detection techniques, mentioned thus far, can be applied complementary to the mutant classification approaches [17] such as the ones proposed by the present paper. Thus, these approaches can identify equivalent mutants and then *I-EQM* can be applied to the remaining uncategorized mutants.

Mutation testing utilizing higher order mutants have been studied by Jia and Harman [22] who suggested the use of higher order mutants could be profitable. In view of this, they propose the use of search based optimization techniques in order to construct hard-to-kill, higher order mutants. To this end, practical problems of mutation testing could be answered by using only such higher order mutants. In a follow up work [31], in order to better simulate real faults, higher order mutants were constructed based on a multi objective evolutionary method. To this end, mutants that were both hard-to-kill and syntactic similar to the original program were produced. Further, automated tools that enable testing based on higher order mutants [32], [33] have also been suggested. A synopsis on higher order mutation and search based testing can be found in [34].

Second order mutation has been addressed in the literature as an alternative method to first order mutation [8], [23]. According to these approaches, equivalent mutants are considerably less likely to occur. Hence, they achieve to reduce their effects on the testing process. Generally, the higher order mutation approaches that appeared in the literature try to produce mutant sets with less equivalent mutants and not to isolate them. The approach presented in this paper is the first one, to the authors' knowledge, that uses higher order mutants in order to identify likely to be first order equivalent ones.

VI. Conclusions and Future Work

The practical application of mutation testing composes the primary aim of the present work. Towards this direction, a dynamic method, named *I-EQM*, able to isolate possible equivalent mutants was suggested. The originality of the proposed technique stems from the fact that it leverages higher order mutation and the mutants' coverage impact. Specifically, the classification scheme utilizes second order mutation to classify a given set of first order mutants as killable or not. The proposed method extends the Schuler and Zeller approach [17] to correctly classify a significantly greater number of killable mutants.

Conclusively, the present paper suggests **a)** a novel dynamic method based on higher order mutation, able to classify first order mutants as possible killable or possible equivalent ones. **b)** An effective integration with the approach proposed by Schuler and Zeller [17] in order to correctly classify a considerably higher number of mutations. Empirical evidence suggested that compared to the Schuler and Zeller approach, approximately 26% more killable mutants can be correctly classified with only a 4% loss on the method's precision. Finally, **c)** a case study on a manually classified mutant set indicating the effectiveness of the proposed approach was performed. The obtained results reveal that *I-EQM* achieves to correctly classify 82% of the killable mutants with a precision of 71%.

The research presented in this paper seeks practical solutions to the application of mutation testing. To achieve it, the work described here can be comprehended in the following directions. It can be extended by using a selective set of mutants. Such a set could provide even better and computationally cheap, classification results. Additionally, the combined use of search based optimization approaches with the coverage impact of the second order mutants could form a possible answer to the equivalent mutant problem. Further directions include conducting additional experiments by employing different mutant sets and subject programs to revalidate the findings of the present paper.

ACKNOWLEDGMENT

This research was partly funded by the Research Centre of the Athens University of Economics and Business.

REFERENCES

[1] A. J. Offutt and R. H. Untch, "Mutation 2000: Uniting the Orthogonal," in Mutation testing for the new century: Kluwer Academic Publishers, 2001, pp. 34-44.

[2] S. Kakarla, S. Momotaz, and A. S. Namin, "An Evaluation of Mutation and Data-flow Testing: A Meta Analysis " in Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on, Berlin, 2011, pp. 366 - 375

[3] P. Ammann and J. Offutt, Introduction to Software Testing: Cambridge University Press, 2008.

[4] N. Li, U. Praphamontripong, and A. J. Offutt, "An Experimental Comparison of Four Unit Test Criteria: Mutation, Edge-Pair, All-uses and Prime Path Coverage," in Proceedings of the 4th International Workshop on Mutation Analysis (MUTATION'09), Denver, Colorado, 2009, pp. 220-229.

[5] J. H. Andrews, L. C. Briand, and Y. Labiche, "Is Mutation an Appropriate Tool for Testing Experiments?," in Proceedings of the 27th International Conference on Software Engineering, St. Louis, MO, USA, 2005, pp. 402-411.

[6] J. H. Andrews, L. C. Briand, Y. Labiche, and A. S. Namin, "Using Mutation Analysis for Assessing and Comparing Testing Coverage Criteria," IEEE Trans. Softw. Eng., vol. 32, pp. 608-624, 2006.

[7] Y. Jia and M. Harman, "An Analysis and Survey of the Development of Mutation Testing," IEEE Trans. Softw. Eng., vol. 99, 2010.

[8] M. Papadakis and N. Malevris, "An Empirical Evaluation of the First and Second Order Mutation Testing Strategies," in Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on, 2010, pp. 90-99.

[9] A. J. Offutt, A. Lee, G. Rothermel, R. H. Untch, and C. Zapf, "An Experimental Determination of Sufficient Mutant Operators," ACM Trans. Softw. Eng. Methodol., vol. 5, pp. 99-118, 1996.

[10] D. Schuler and A. Zeller, "Javalanche: Efficient Mutation Testing for Java," 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, Amsterdam, The Netherlands, 2009, pp. 297-298.

[11] A. J. Offutt and J. Pan, "Automatically Detecting Equivalent Mutants and Infeasible Paths," Software Testing, Verification and Reliability, vol. 7, pp. 165-192, 1997.

[12] T. A. Budd and D. Angluin, "Two Notions of Correctness and their Relation to Testing," Acta Informatica, vol. 18, pp. 31-45, 1982.

[13] M. Papadakis and N. Malevris, "Automatically Performing Weak Mutation with the Aid of Symbolic Execution, Concolic Testing and Search-based Testing," Software Quality Journal, vol. 19, pp. 691-723, 2011.

[14] M. Papadakis and N. Malevris, "Automatic Mutation Test Case Generation via Dynamic Symbolic Execution," in Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on, 2010, pp. 121-130.

[15] G. Fraser and A. Zeller, "Mutation-driven Generation of Unit Tests and Oracles," in Proceedings of the 19th International Symposium on Software Testing and Analysis (ISSTA), 2010, pp. 147-158.

[16] B. Baudry, F. Fleurey, J.-M. Jézéquel, and Y. L. Traon, "From Genetic to Bacteriological Algorithms for Mutation-based Testing: Research Articles," Software Testing, Verification & Reliability, vol. 15, pp. 73-96, 2005.

[17] D. Schuler and A. Zeller, "(Un-)Covering Equivalent Mutants," in Software Testing, Verification and Validation (ICST), 2010 Third International Conference on, 2010, pp. 45-54.

[18] D. Schuler, V. Dallmeier, and A. Zeller, "Efficient Mutation Testing by Checking Invariant Violations," in Proceedings of the 18th International Symposium on Software Testing and Analysis, Chicago, IL, USA, 2009, pp. 69-80.

[19] B. J. M. Grun, D. Schuler, and A. Zeller, "The Impact of Equivalent Mutants," in Software Testing, Verification and Validation Workshops, 2009. ICSTW '09. International Conference on, 2009, pp. 192-199.

[20] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on Test Data Selection: Help for the Practicing Programmer," Computer, vol. 11, pp. 34-41, 1978.

[21] A. J. Offutt, "Investigations of the Software Testing Coupling Effect," ACM Trans. Softw. Eng. Methodol., vol. 1, pp. 5-20, 1992.

[22] Y. Jia and M. Harman, "Higher Order Mutation Testing," Information and Software Technology, vol. 51, pp. 1379-1393, 2009.

[23] M. Kintis, M. Papadakis, and N. Malevris, "Evaluating Mutation Testing Alternatives: A Collateral Experiment," in Software Engineering Conference, 17th Asia Pacific, 2010, pp. 300-309.

[24] G. Fraser and A. Zeller, "Generating Parameterized Unit Tests," in Proceedings of the 2011 International Symposium on Software Testing and Analysis, Toronto, Ontario, Canada, 2011, pp. 364-374.

[25] C. D. Manning, P. Raghavan, and H. Schütze, Introduction to Information Retrieval: Cambridge University Press, 2008.

[26] K. Adamopoulos, M. Harman, and R. M. Hierons, "How to Overcome the Equivalent Mutant Problem and Achieve Tailored Selective Mutation Using Co-evolution.," Springer, 2004, pp. 1338-1349.

[27] D. Baldwin and F. G. Sayward, "Heuristics for Determining Equivalence of Program Mutations," Yale University, New Haven, Connecticut 1979.

[28] A. J. Offutt and W. M. Craft, "Using Compiler Optimization Techniques to Detect Equivalent Mutants," Software Testing, Verification and Reliability, vol. 4, pp. 131-154, September 1994.

[29] R. M. Hierons, M. Harman, and S. Danicic, "Using Program Slicing to Assist in the Detection of Equivalent Mutants," Software Testing, Verification and Reliability, vol. 9, pp. 233-262, December 1999.

[30] M. Harman, R. M. Hierons, and S. Danicic, "The Relationship between Program Dependence and Mutation Analysis," San Jose, California, 2001, pp. 5-13.

[31] W. B. Langdon, M. Harman, and Y. Jia, "Efficient Multi-objective Higher Order Mutation Testing with Genetic Programming," Journal of Systems and Software, vol. 83, pp. 2416-2430, 2010.

[32] J. Yue and M. Harman, "MILU: A Customizable, Runtime-optimized Higher Order Mutation Testing Tool for the Full C Language," in Practice and Research Techniques, 2008. TAIC PART '08. Testing: Academic & Industrial Conference, 2008, pp. 94-98.

[33] 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, Szeged, Hungary, 2011, pp. 212-222.

[34] M. Harman, Y. Jia, and W. B. Langdon, "A Manifesto for Higher Order Mutation Testing," Paris, France, 2010, pp. 80-89.