Computer Science Review 25 (2017) 29–48

Contents lists available at ScienceDirect

Computer Science Review

journal homepage: www.elsevier.com/locate/cosrev

Review Article

Higher order mutation testing: A Systematic Literature Review

Ahmed S. Ghiduk [a,b,*], Moheb R. Girgis [c], Marwa H. Shehata [b]

[a] College of Computers and IT, Taif University, Saudi Arabia
[b] Department of Math. and CS, Faculty of Science, Beni-Suef University, Egypt
[c] Department of Computer Science, Faculty of Science, Minia University, Egypt

ARTICLE INFO

Article history:
Received 1 July 2016
Received in revised form 8 June 2017
Accepted 15 June 2017
Available online 4 August 2017

Keywords:
Mutation testing
Higher-order mutation testing
First-order mutants
Higher-order mutants

ABSTRACT

Mutation testing is the process whereby a fault is deliberately inserted into a software system, in order to assess the quality of test data, in terms of its ability to find this fault. Mutation testing is also used as a way to drive the test data development process. Traditionally, faults were inserted one by one into a software system, but more recently there has been an upsurge of interest by the area of higher-order mutation, in which multiple faults are inserted into the system at once. Originally, this was thought to be too expensive, as there was already a concern that the size of the pool of mutants for traditional mutation was already too large to handle. However, following a seminal publication in 2008, it was realized that the space of higher-order mutants (*HOMs*) could be searched for useful mutants that drive testing harder, and to reduce the overall test effort, by clever combination of first-order mutants. As a result, many authors examined the way in which *HOM* testing could find subtle hard to kill faults, capture partial fault masking, reduce equivalent mutants problem, reduce test effort while increasing effectiveness, and capture more realistic faults than those captured by simple insertion of first-order mutants. Because of the upsurge of interest in the previous issues, this paper presents the first Systematic Literature Review research specifically targeted at a higher-order mutation. This Systematic Literature Review analyzes the results of more than one hundred sixty research articles in this area. The current paper presents qualitative results and bibliometric analysis for the surveyed articles. In addition, it augments these results with scientific findings and quantitative results from the primary literature. As a result of this work, this *SLR* presents an outline for many future work.

Contents

* Corresponding author at: College of Computers and IT, Taif University, Saudi Arabia.
E-mail addresses: asaghiduk@tu.edu.sa (A.S. Ghiduk), moheb.girgis@mu.edu.eg (M.R. Girgis), marwahashem88@yahoo.com (M.H. Shehata).

http://dx.doi.org/10.1016/j.cosrev.2017.06.001
1574-0137/

1. Introduction

1.1. Software Testing (ST)

ST is a process to identify the differences between current condition of software and desired condition. The aim of ST is to develop the software through determine whether the software achieves its requirements, and to ensure that the software does not contain any errors. In fact, most of the resources of a software project (more than 50%) are invested in the detection and correction of faults [1,2]. Despite this massive investment, it is widely known that the complexity of software makes it impossible to detect all the faults of software [1–3].

ST is very important to increase the confidence in the software system or application because they may contain many hidden errors. Some of these errors are unimportant, but some of them are expensive or dangerous. Therefore, the software system or application should be checked to obtain a high efficiency software. There are many types of *ST* such as black-box testing, white-box testing, component-based testing, and integration testing. This study focuses on mutation testing particularly higher-order mutation testing (*HOMT*) as a type of white-box testing.

ST depends on generating test cases which are collection of test actions that are executed to verify a specific attribute or function of the software application. There are many testing techniques used to design the test cases. Some of these techniques work on designing highly effective test cases that detects more errors with less effort and time [1–3]. These techniques can be categorized according to source code knowledge such as black-box, white-box, gray-box techniques [4–6], used information source knowledge such as program-based, specification-based, interface-based techniques [4,5,7], or testing level knowledge such as unit-level, integration-level, system-level techniques [2,7].

1.2. Mutation Testing (MT)

MT is developed by DeMillo et al. [8] and Hamlet [9]. In *MT*, a change is made in the source code of the software. Then, the software is executed to check if the test cases are able to detect the errors. The changes in the faulty program do not have an effect on the aim of the program. A goal of *MT* is to evaluate the quality of the test cases in revealing the mutants. *MT* was developed to find test inputs to kill the mutants in the tested program [9]. *MT* requires creating faults (errors) in the tested program so it is called a fault-based testing technique. The process of *MT* is accomplished in three steps as follow:

- First, inserting faults into the original program by making simple syntactic changes to produce a set of faulty programs called mutants; each mutant contains a different syntactic change.
- Second, applying set of test cases on the original program and also on all mutants. Then the ability of these test cases on detecting errors are assessed by calculating the mutation score (*MS*). *MS* measures the effectiveness of a test set in terms of its ability to detect faults.
- Third, comparing the results of the original program and the mutated one. If the result of executing a mutant differs from the result of executing the original program for any test cases in the input test set, the fault denoted by the mutant is killed or detected; otherwise it is said to be survived.

The *MS* is the ratio of the number of killed mutants to the difference between total number of the mutants and the number of equivalent mutants. The value of *MS* is between *"0"* and *"1"*. If *MS* has a lower value, this means that mutants cannot be detected accurately by the test set. If MS has a higher value, we say that most of the mutants were killed with this test set. When *MS = 0* that refers to the mutants cannot be killed by any test set, and when *MS = 1* that means the mutants are killed easily. The *MS* is calculated by the following formula: $MS = \frac{Number\ of\ Killed\ mutants}{Total\ number\ of\ mutants - Number\ of\ equivalent\ mutants}$.

Mutants can be categorized into two types: first-order mutants (*FOMs*) and higher-order mutants (*HOMs*). *FOMs* are created by applying mutation operators only once. *HOMs* are created by applying mutation operators more than once. Mutation operators are set of modifications that are applied on the program to generate mutants. Depend on the used programming languages, there are many mutation operators such as statement deletion, logic or arithmetic operator replacement and variable replacement. Table 1 provides examples of *FOMs* and *HOMs*.

MT can take place in many activities such as evaluating the quality of the test set to produce a high quality and stable system. On the other side, it has a number of obstacles. The first challenge of *MT* is the enormous number of mutants. Whereas, not only the original program is considered but each mutant is also executed by set of test cases, which means high cost and high effort are needed. The second challenge is the realism problem. Mutants are created by inserting single and simple changes in the original program; so the realistic faults are not denoted. Also, there is no any guarantee that most of real faults have killed [10]. The third challenge is the equivalent mutant problem. Some mutation operators generate mutants semantically similar to the original program. These mutants are called equivalent mutants and their detection are very difficult and require more human effort. Several approaches have been introduced to provide considerable solutions for these problems such as do smarter approach [11,12], do faster approach [13,14], do fewer approach [15], and incremental mutation testing [16]. MT is applied not only to common programming languages but also to other domains such as *SQL*, aspect-oriented programs, network protocols, web services, etc [17].

HOMT is a new model of *MT* which studies *HOMs*. It was proposed by Jia and Harman [18]. *HOMs* are considered as more

Table 1
Examples of FOMs and *HOMs*.

Original Program	Mutated Program p'		
	First-Order Mutant	Higher-Order Mutant	
		Second-Order Mutant	Third-Order Mutant
sum = $x + y$ avg = sum/n	sum = $x * y$ avg = sum/n	sum = $x * y$ avg = sum − n	sum = $x * y + +$ avg = sum − n

complex faults [19]. They are generated by adding two or more errors (faults) in the original program [18]. *HOMs* can be generated by combining *FOMs*. *HOMT* can solve some *MT* problems [18–20]. *HOMT* may reduce the number of created mutants and also can generate *HOMs* harder to kill than the *FOMs* such as subsuming HOMs [10,18]. In addition, *HOMT* can limit unrealistic and avoid generating equivalent mutants [18,20–22].

Because of the upsurge of interest in *HOMT*, this paper presents the first survey paper specifically targeted at a higher-order mutation. Although there have been previous surveys of mutation testing, and testing more generally, this is the first survey that specifically targets higher-order mutation. So far here is just about sufficient interest and activity in this area that it now warrants a survey, and therefore it is timely to publish a systematic literature review on the topic of higher-order mutation. The contributions of this paper are:

- Inspection of more than one hundred sixty research articles in the higher-order mutation testing area.
- Presenting qualitative results and bibliometric analysis for the surveyed articles.
- Augmenting these results with scientific findings and quantitative results from the primary literature.
- Summarizing the work which has been done to overcome the key obstacles of *HOMT*.
- As a result of this work, this *SLR* presents an outline for many open work in *HOMT*.
- Achieving the following research questions:
  - *RQ1*: How expensive is higher-order mutation testing?
  - *RQ2*: For what extent can higher-order mutation testing improve the subtlety of mutants?
  - *RQ3*: For what extent can higher-order mutation testing overcome the equivalent mutant?
  - *RQ4*: For what extent can higher-order mutation testing reduce the cost of effective testing?
  - *RQ5*: For what extent can higher-order mutants simulate real-fault?

This systematic literature review (*SLR*) is performed according to the processes defined by Kitchenham and Charters [23]. This SLR is organized as follows: Section 2 describes the planning and the method used to conduct the systematic review and reports the results of this review. Section 3 introduces the results of this *SLR* and answers the research questions. Section 4 introduces the findings of this *SLR* especially the work which has been done, the work to do and the threats to validity. Section 5 presents the conclusion of this *SLR* and the future work.

2. Review methodology

This Systematic Literature Review (*SLR*) analyzes and appraises all available works related to our research questions. According to the guideline of Kitchenham and Charters [23], the goal of *SLR* is to characterize best practices by means of specific procedures, technologies, methods or tools by aggregating information from comparative studies. This *SLR* contains three steps: (1) planning the review, (2) conducting the review, and (3) reporting the review, which will discuss in details throughout this section.

2.1. Planning the review

The planning phase is concerned with developing the review protocol.

Research questions (RQs) **:** The main objective of this SLR is analyzing the work which has been done to overcome the key challenges of *HOMT* such as equivalent mutant, realism, explosion of number of mutants, *HOMT* approaches, *HOMs* generation techniques, and test data generation techniques for killing *HOMs*. The following research questions are proposed to achieve these objectives.

- *RQ1*: How expensive is higher-order mutation testing?
- *RQ2*: For what extent can higher-order mutation testing improve the subtlety of mutants?
- *RQ3*: For what extent can higher-order mutation testing overcome the equivalent mutant?
- *RQ4*: For what extent can higher-order mutation testing reduce the cost of effective testing?
- *RQ5*: For what extent can higher-order mutants simulate real-fault?

2.2. Conducting the review

This phase is divided into four main steps as follow:

1. ***Search strategy:*** To answer on the research questions, this *SLR* uses some keywords to search for the primary studies in some search resources as follow:

Search keywords: The coverage landscape of this SLR is the area of higher-order mutation testing. The set of search terms were devised in a systematic and iterative fashion, i.e., the search starts with an initial set of keywords and iteratively improved this set until no further relevant papers could be found to improve the pool of primary studies. By taking all of the above aspects into account, the search query is formulated as follows: {equivalent || subtle || higher order || second order || mutation testing || fault based} && { mutation || mutant || tools || testing || techniques || methods || problems || analysis || approaches}. The *SLR* searched the whole text of the studies if the search engine supported full text search. If the search engine does not support full text search, the title, abstract and keywords of the studies are included in the search process.

Search Resources: To provide a comprehensive *SLR* covered all the publications related to *HOMT*, more than 200 papers on mutation testing from 1977 to 2017 are downloaded. *SLR* selected the papers on second-order mutation testing (*SOMT*) in particular and *HOMT* in general which were published between 1992 and 2017. The *SLR* searched for these papers in the following databases: *IEEE* Explore, *ACM* Digital Library, Google Scholar, Web of Science, Science Direct, Springer Library, Elsevier Online Library, Microsoft Academic Search. Also we depended on the references that found in our primary studies during the process of searching.

2. ***Studies Selection:*** This section discusses the criteria for inclusion or exclusion the primary studies that are related to this work.

Inclusion criteria: when selecting the primary studies that provide us the answers to research questions, the studies that achieve the following criteria are included:

Table 2
Summary of primary SOM and *HOM* testing studies by publication year and publication type.

Study Reference (Publication Type: J.: Journal; C.: Conference; B. Ch.: Book Chapter, Ph.D.: Ph.D. thesis)	Publication Year	Number of Publications	Two-year duration	Two-year progress	Average of two-year progress%
Offutt [24] (J.)	1992	1	1992	1	50%
Polo et al. [35] (J.); Jia and Harman [28,36] (C.)	2008	3	1992 to 2008	4	200%
Jia and Harman [18] (J.); Langdon et al. [27] Schuler and Zeller [12] (C.)	2009	3	2008 to 2009	6	300%
Langdon et al. [19](J.); Harman et al. [10], Kintis et al. [22], Papadakis and Malevris [20] (C.)	2010	4	2009 to 2010	7	350%
Kapoor [37](J.); Harman et al. [38], Blanco-Muñoz [39] (C.)	2011	3	2010 to 2011	7	350%
Akinde [25](J.); Kintis et al. [26], Omar and Ghosh [40](C.)	2012	3	2011 to 2012	6	300%
Mateo et al. [41](J.); Omar et al. [29](C.); Jia [42] (Ph.D.)	2013	3	2012 to 2013	6	300%
Madeyski et al. [21], Ghiduk [43](J.); Derezińska and Hałas [44], Omar et al. [32,45], Harman et al. [33](C.); Nguyen and Madeyski [46](B. Ch.)	2014	7	2013 to 2014	10	500%
Jia et al. [47] (C.), [48] (J.), Nguyen and Madeyski [30](B. Ch.)	2015	3	2014 to 2015	10	500%
Nguyen and Madeyski [49,50], Ghiduk [51](J.); Nguyen and Madeyski [52](B. Ch.); Tokumoto et al. [53], Lima et al. [54], Wu et al. [55] (C.)	2016	7	2015 to 2016	10	500%
Omar et al. [31] (J.), Nguyen and Madeyski [56](J.)	2017	2	2016 to FQ2017	9	450%

Table 3
The extracted data from the primary studies.

Research Issue	No. of studies	Ratio of studies
Reducing number of *HOMs*	10	55.6%
Constructing hard-to-kill *HOMs*	4	22.2%
Identifying equivalent *HOMs*	3	16.7%
Constructing more realistic *HOMs*	1	5.6%
Test generation for *HOMs*	3	16.7%
Genetic Improvement	1	5.6%
Total number of studies = 18 (some studies consider more than one research issue)		
Used technique for constructing *HOMs*	Frequency	Most used technique (frequency)
Search based techniques	19	Genetic algorithm (10)(52.6%)
Data flow based techniques	1	Definition-use associations (1) (0.053%)
Systematic based techniques	25	Last To First (7) (36.8%)
Dynamic symbolic execution	1	Dynamic symbolic execution (1)(0.053%)
Random based techniques	8	Fair random (8)(42.1%)
Total number of used techniques = 19 (some techniques have been used more than one time)		

- *IC1*: Must be published in journal issue.
- *IC2*: Must be published in conference proceedings
- *IC3*: Focus on *HOMT*.
- *IC4*: Discuss *HOMT* approaches and their advantages.
- *IC5*: Discuss test data generation techniques and their ability for killing *HOMs*.
- *IC6*: Discuss the difference between *HOMT* and traditional mutation testing.
- *IC7*: Answer one or more of our research questions.

Exclusion criteria: A study is excluded if it meets one of the following criteria:

- *EC1*: Study is not available in hard or electronic format.
- *EC2*: Duplicate studies reporting similar results.
- *EC3*: Study is not written in English.
- *EC4*: Study is not a full paper, short paper, MSc. thesis or Ph.D. thesis (e.g., posters, and tutorials).
- *EC5*: Study does not relate to higher-order mutation testing topic.

Table 2 presents a summary for the primary studies that were surveyed and achieved the inclusion criteria (details are given in Tables 9–11). The primary studies are organized according to the publication year. Form Table 2, one can determine that *SOMT* was first proposed in 1992 by Offutt [24]. Then, the development of *SOMT* and *HOMT* was between 2008 and 2017 (first quarter of 2017: FQ2017). In addition, Table 2 presents two-year publications progress (the total number of publications in each two consecutive years) and the percentage of the average of the two-year progress in the number publications. Fig. 1 shows the percentage of the average of the two-year progress in the number of publications and the trend line of the publications. The publications trend line shows that the size of publications is directly proportional with the time.

From Table 2 and Fig. 1, one can conclude that in the last decade the researchers perception about higher-order mutation testing has been changed. The following points summarizes these new perspectives and claims:

- *Overcoming equivalent mutant problem*: One of the problems that has dumped a mutation testing since its inception has been the problem of equivalent mutants. This problem is essentially un-decidable. It is one that has been hard to overcome. Recently, researchers claimed that higher-order mutation testing can address this problem [21,25,26].
- *Creating real faults*: higher-order mutation testing can create mutants that are more like a real faults than first-order mutation testing [19,27].
- *Finding subtle faults*: higher-order mutant subsumption increases the effectiveness of mutation based testing, by making new mutants are harder to kill than any of the individual mutants from which they are constructed [28–33].
- *HOMT is not expensive*: it is not grossly expensive to compute higher-order mutants. Although the space is enormous, the valuable ones are sparse and can be found using techniques such as search based software engineering [10].
- *Reducing the effort required to achieve effective testing*: efficiency has proved to be a bugbear of mutation testing in general. Higher-order mutation testing increases the efficiency of mutation based testing by combining first-order mutants into a single higher-order mutant [11–15,34].

3. ***Data Extraction***: The aim of this step is introducing the obtained data from each selected study such as study title, study result, and study objectives. These extracted data help us to answer the *RQs*. We deduced from the primary studies the data given in Table 3. Table 3 presents Research issue, number of studies in each issue, ratio of studies, used technique for constructing *HOMs*, frequency, and most used technique and its frequency.

From these data, we can inferred that most studies (55.6%) concentered on reducing number of *HOMs*. While there are a very few number of studies concentered on constructing more realistic *HOMs* (5.6%), and genetic improvement (5.6%). Systematic-based methods (100%) and search-based methods (100%) are the most used in *HOMT*. Genetic algorithm (52.6%), last to first (36.8%), and fair random (42.1%) are the most frequency techniques.

4. ***Study Quality Assessment*:** This section assesses each selected study based on set of quality criteria. The quality criteria (*QC*) were based on some quality assessment (*QA*) questions as in Table 4. The used questions in this study was based on recommendations of Kitchenham and Charters [57] and Khan et al. [58] with some specific questions according to the proposed research questions and the type of this study. The questions were answered with *Yes = 1 "Y"* or *Partly = 0.5 "P"* or *No = 0 "N"* as in Table 4. These quality assessments are applied on each selected study in the surveyed studies and evaluated the quality score (*QS*) for each study through the answering on 7 questions. The value of *QS* is between 0 (very poor) and 7 (excellent). Table 5 gives an example for the process of evaluating *QS* for only five studies form the selected studies. After calculating the score for each selected study, we summed the number of the studies that are equal in the score and recorded the *QS* percentage for them as in Table 6.

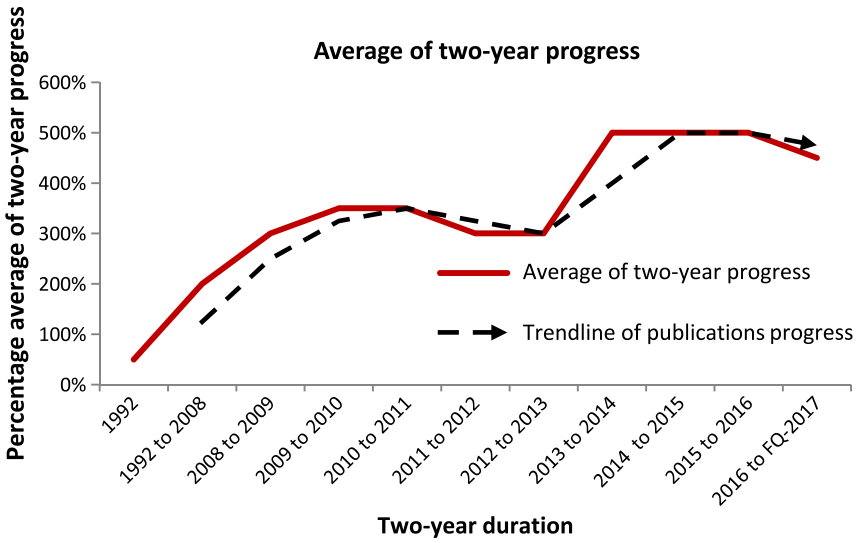

Fig. 1. Percentage average of two-year progress.

Table 4
Quality assessment questions.

No	Questions
QA1	Does the study topic cover the answers to the SLR questions?
QA2	Is there a clear statement of the objectives of the research?
QA3	Do the study results have been evaluated?
QA4	Is the study topic mentioned in previous studies?
QA5	Is the paper well referenced?
QA6	Does the paper contain links to previously selected resources?
QA7	Are the used methods in the study described?

2.3. Reporting the review

This section reports the analysis of the surveyed studies and introduces set of classifications for the *HOMT* studies based on their features.

1. *HOMT* approaches classification based on mutant degree

The surveyed higher-order mutation testing (*HOMT*) approaches given in Table 7 are classified into second-order mutation testing (*SOMT*) and *HOMT* approaches. The employed techniques and their results are introduced as follow:

SOMT Approaches: Polo et al. [35] proposed three algorithms, last to first (*L2F*), different operators (*DiffOp*), and random mix (*RMix*), to generate *SOMs*. The *L2F* algorithm uses the list of n *FOMs* and generates *SOMs*. This can be obtained by the combination of the first *FOM* in the list with the last *FOM* in the list and so on. The *DiffOp* algorithm generates *SOMs* by combining two *FOMs* each of them was created by different mutation operator. The *RMix* algorithm combines any two randomly selected *FOMs* into *SOM*. The result of this study showed that the number of *SOMs* is reduced into half the number of *FOMs*, and the mean ratio of equivalent *SOMs* is only about 5% compared with 18.66% of equivalent *FOMs* [35].

Papadakis and Malevris [20] suggested five techniques to generate *SOMs* by combining *FOMs*. These techniques are last to first (*L2F*), same node (*SNode*), same unit (*SUnit*), *SU_F2Last* and *SU _DiffOp*. Their empirical study deduced that the number of generated *SOMs* is reduced to half and this reduced the execution cost.

Kintis et al. [22] introduced two categories of *SOMT* strategies. The first category includes two second order strategies *RDomF* and *SDomF*. The second category includes two hybrid strategies *HDom* (20%) and *HDom* (50%). The paper results showed that the number of generated equivalent mutants of weak mutation, *RDomF*, *SDomF*, *HDom* (20%) and *HDom* (50%) strategies reduced about 73%, 85.4%, 86.8%,81.4% and 65.5% in turn compared with the number of generated equivalent mutants of strong mutation. Also the mutation scores are 99.94%, 99.99%, 99.91%, 99.91% for the *RDomF*, *SDomF*, *HDom* (20%) and *HDom* (50%) strategies, respectively. In 2012, Kintis et al. [26] proposed *I-EQM* technique. This technique is able to dynamically isolate first-order equivalent mutants using *SOMs*. The proposed approach [26] applied three classification techniques (*HOM* classifier, *I-EQM* classifier, Coverage Impact Classifier) on *FOMs*. The objective of these techniques is to classify first-order

Table 5
Example on the process of evaluating quality score for 5 selected studies.

Study Ref	QA1	QA2	QA3	QA4	QA5	QA6	QA7	Total Score
Jia and Harman [18]	P = 0.5	Y = 1	Y = 1	N = 0	Y = 1	Y = 1	Y = 1	5.5
Polo et al. [35]	Y = 1	P = 0.5	Y = 1	P = 0.5	Y = 1	Y = 1	Y = 1	6
Jia and Harman [28]	P = 0.5	P = 0.5	N = 0	N = 0	Y = 1	Y = 1	N = 0	3
Myers and Sandler [2]	N = 0	Y = 1	N = 0	Y = 1	P = 0.5	Y = 1	N = 0	3.5
Ghiduk [43]	Y = 1	Y = 1	Y = 1	P = 0.5	Y = 1	Y = 1	P = 0.5	6
Kitchenham and Charters [57]	N = 0	Y = 1	N = 0	P = 0.5	P = 0.5	P = 0.5	N = 0	2.5

Table 6
Quality scores for the all selected studies.

Description	Quality scores				
	Very poor	poor	Good	Very good	Excellent
QS	QS <2	2 ≤ QS <3.5	3.5 ≤ QS <5	5 ≤ QS <6	6 ≤ QS <7
QS percentage	QS % <28%	28%≤ QS % <50%	50% ≤ QS % <71%	71% ≤ QS % <85%	85% ≤ QS % <100%
Number of Studies	10	18	35	19	27

Table 7
SOMs and *HOMs* generation techniques.

Year	Technique	Degree	Ref	No. of SOMs Techniques	No. of *HOMs* Techniques
2008	LTF, DiffOp and RM	*SOMs*	Polo [35]	1	1
	GR algorithm, GA and HC Algorithm	*HOMs*	Jia and Harman [28]		
2009	GR algorithm, GA and HC Algorithm	*HOMs*	Jia and Harman [18]	0	2
	GP	*HOMs*	Langdon et al. [27]		
2010	LTF , S Node, SUnit , SU_LTF and SU_DiffOp	*SOMs*	Papadakis and Malevris [20]	2	1
	i. Second Order Strategies category(RDomF and SDomF) ii. Hybrid Strategies category (HDom(20%) and HDom(50%)	*SOMs*	Kintis et al. [22]		
	Monte Carlo sampling, GA and GP	*HOMs*	Langdon et al. [19]		
2011	MiLu (*HOMs* generation and assessment tool for c language)	*HOMs*	Harman et al. [38]	0	1
2012	MILU tool	*HOMs*	Akinde [25]	0	1
2013	Each-Choice (ECH) algorithm and Between-Operators (BTO) algorithm	*SOMs*	Mateo et al. [41]	1	1
	G A , Local Search algorithm, and Random Search algorithm	*HOMs*	Omar et al. [29]		
2014	JudyDiffOp algorithm and NeighPair algorithm	*SOMs*	Madeyski et al. [21]	1	2
	CIA and RNA	*HOMs*	Ghiduk [43]		
	Each-Choice (ECH), Between-Operators (BTO) , LTF, and Random (RND) algorithms	*HOMs*	Derezińska and Hałas [44]		
2015	Multi-objective Optimization Algorithm	*HOMs*	Nguyen and Madeyski [30]	0	1
2016	Multi-objective Optimization Algorithms	*HOMs*	Nguyen and Madeyski [49]	0	2
	Multi-objective Optimization Algorithms	*HOMs*	Nguyen and Madeyski [52]		

mutants as possibly killable or possibly equivalent ones. This study was compared to the Schuler and Zeller approach [59]. The approach results showed that *I-EQM* achieves a classification precision score is 71% and a classification recall score is 81% .

Madeyski et al. [21] proposed *NeighPair* algorithm and *JudyDiffOp* algorithm. The idea of *JudyDiffOp* algorithm is a modulation of the different operators algorithm [11]. The two algorithms are based on creating *SOMs* by combining *FOMs* but each algorithm combines the mutants in different way. These algorithms are experimented with *RMix* and *L2F* [35]. The experiments showed that the number of created *SOMs* was reduced to half comparing with created *FOMs*. The number of equivalent mutants was decreased with average 47%, 58.5%, 66% for the *RMix*, L2F and *JudyDiffOp*, respectively. While *NeighPair* algorithm is unhelpful for reducing number of equivalent mutants.

***HOMT Approaches*:** Jia and Harman [18] applied three search based algorithms (*GR* Algorithm, *GA* and *HC* Algorithm) to generate a good *HOMs* called subsuming *HOMs* (*SSHOMs*). *SSHOM* is harder to kill than the first-order mutants from which it is created. The three algorithms were applied on ten subject programs. The results of this study mentioned that there exist many *SSHOMs* in each studied program. Also the results deduced that genetic algorithm is able to generate *SSHOMs* with percentage 80%, while the greedy algorithm with 50% and hill climbing algorithm with 58%. Harman et al. [38] used *MiLu* (*HOM* generation tool for C) to generate *FOMs* and *HOMs*. They also applied the method of combining *DSE* and *SBST* techniques to create a high effective test data able to detect both *FOMs* and *HOMs*. The study results showed that the generated test data are able to kill *FOMs* with a high percentage than any other previous test data generation approach. Also this technique is considered the first technique in generating test data for killing *SOMs*.

Akinde [25] applied MiLu tool for generating *FOMs*, SOMs and *HOMs*. The study results showed that the number of equivalent mutants when generating *SOMs* less than *FOMs*. Similarly, number of equivalent mutants when generating *HOMs* closes to zero percentage.

Ghiduk [43] applied genetic algorithms based technique to generate the test inputs for killing *HOMs*. This paper used MuJava tool to generate *FOMs* of the program. Then, it proposed two algorithms circular incremental algorithm (*CIA*) and random N algorithm (*RNA*) based on MuJava to find the *HOMs*. Also, this paper applied genetic algorithm to generate set of test inputs that killed *FOMs* and *HOMs*.

2. *HOMT* approaches classification based on mutant generation

For the practicability and efficiency of *HOMT*, approaches to generate good (strong) *HOMs* that are harder to kill are needed. As given in Table 9, there are two types of approaches were introduced to find strong *HOMs* as follow:

***Single Objective (SObj)*:** Jia and Harman [18,28] applied single objective search based optimization techniques to find *SSHOMs*. They defined new fitness function to find *SHOMs*. Firstly, they calculate the fragility function for the *FOMs* and *HOMs* after that they calculate the fitness function for *HOMs*. The fragility function is defined as follow: $fragility(\{MT_1, \ldots, MT_n\}) = \frac{|\bigcup_{i=1}^{n} kill(MT_i)|}{|TC|}$, where *TC* is the set of test cases, $MT = \{MT_1, \ldots.., MT_n\}$ is a set of mutants and the function $kill(\{MT_1, \ldots, MT_n\})$ represents the set of test cases that kills mutants. The fragility value is between *"0"* and *"1"*. If the value is *"0"* this refers to the mutant could not be killed by this set of test cases. If the value is greater than zero this means that the mutant is weaker. When the value reaches *"1"* this leads to the mutant can be killed by the test cases. If we considered the sets from MT_1 *to* MT_n represent the *HOM* consisting of the FOMs FT_1 to FT_n. The fitness function for a *HOM* is: $fitness(MT_{1\ldots n}) = \frac{fragility(\{MT_{1\ldots n}\})}{fragility(\{FT_{1,\ldots,}FT_n\})}$. So the fitness of a *HOM* is the ratio of the fragility of its *HOM* to the fragility of the *FOMs* which it was created from them. If the fitness is greater than *"1"*, the *HOM* is said to be a weak mutant. When the fitness decreases from *"1"* to *"0"*, the *HOM* becomes strong mutant. If the fitness value equals *"0"*, it is called equivalent *HOM*.

***Multi Objective (MObj)*:** Langdon et al. [27,60] used *GP* to search for *HOMs*. Also in 2010, they [19] used Monte Carlo sampling, *GA* and *GP* to find the strong *HOMs*. The two studies [19,27] used *GP* for finding strong *HOMs*. The *GP* used two fitness functions to formulate the mutants. These functions are semantic difference and syntactic difference. The syntactic distance function is defined as the summation of the number of changes measured by the actual difference. Besides, the semantic distance function is calculated as the number of test cases for which a mutant and original program act differently. The result of this study demonstrated that the *GP* approach is able to find *HOMs* hardly killing.

Nguyen and Madeyski [30,52] applied multi objective optimization algorithms (*NSGA-II* algorithm) to create *HOMs* and using their new objective and fitness functions to search for valuable *SSHOMs*. The results showed that this technique have many benefits in searching for strongly *SSHOMs* but the number of equivalent *HOMs* is still large.

3. Test data generation approaches

This classification concentrates on categorizing the test data generation approaches and techniques that were used for killing *FOMs* and *HOMs* and showing their effectiveness. One of the major challenges in mutation testing is generating test data to kill a large number of mutants. An effective test case is the test case that kills a large number of mutants than another. Furthermore, a test suite is effective if it contains a few number of test cases. To kill *FOM*, a test input needs to satisfy three conditions are reachability, infection and propagation [61,62].

Table 8 presents *FOMs* and *HOMs* test data generation approaches, the employed techniques, the type of killed mutants and their ability for killing the mutants. The results of this *SLR* showed that many scholars proposed approaches to reduce computational costs and optimize the test data generation for mutation testing. These approaches aim to construct test data that can detect non-equivalent mutants in a shorter period, by reducing the costs of test data generation and improving the quality of the resulting test data set. In addition the results of this *SLR* showed that most of the previous approaches on mutation test data generation used SBST techniques such as hill climbing [76], ant colony optimization [77], evolutionary algorithms [78,79], and genetic algorithm [80].

From Table 8, we can deduce that most of previous test data generation approaches aim to kill *FOMs* and the work on killing *HOMs* is very limited. Harman et al. [38] introduced a new approach for generating test data approach that combines dynamic symbolic execution (*DSE*) and search based techniques. The results of this approach showed that this new technique can kill 38% of *FOMs* using reachability and infection and kill 36% of the mutants using reachability alone. In addition, the results showed that the technique kills 48% of the *SOMs* using reachability and infection, which in turn kills 41% of the mutants using reachability alone.

Ghiduk [43] introduced GA-Based *HOMT* system (*GAMTS*) that used *GA* technique to generate set of test inputs for killing *FOMs* and *HOMs*. The results showed that the used technique killed 81.8% of the *FOMs*, 90% of the *SOMs*, and 93% of the third-order mutants of the total number of mutants in all subject programs. This study showed that *GA* has high effective in killing non-equivalent mutants of orders one, two, and three.

4. The used programming language

This subsection concentrates on identifying programming languages that were used by *HOMT* approaches. *MT* is used to test both the specification of the program (specification mutation) [81] and the source code of the program (program mutation) [82].

Table 8
FOMs and *HOMs* test data generation approaches.

Reference	Technique	Degree of mutant	Mutation Score(MS)
Harman et al. [38]	DSEand SBST	*HOMs*	Unknown
Ghiduk [43]	GA	*HOMs*	60%<MS<95%
DeMillo and Offutt [61]	Constraint-Based Testing (CBT)	*FOMs*	MS >90%
Hanh et al. [63]	GA and simulated annealing (SA) algorithm	*FOMs*	MS = 85.7% for the GA and MS = 85.5% for the SA
Papadakis and Malevris [64]	HC algorithm	*FOMs*	MS ≈96%
Louzada et al. [65]	Elitist Genetic Algorithm (EGA)	*FOMs*	MS = 92.2%
Malhotra and Garg [66]	GA	*FOMs*	MS ≈ 88.33%
Papadakis et al. [67]	Enhanced Control Flow Graph (ECFG)	*FOMs*	MS = 86.3%
Rad et al. [68]	Bacteriological Algorithms (BA)and GA	*FOMs*	MS = 87.5% for BA and MS = 90.6% for GA
Fraser and Zeller [69]	GA	*FOMs*	58.61% <MS <82.95%.
Papadakis and Malevris [70]	Dynamic Symbolic Execution (DSE)	*FOMs*	MS = 63%
Mishra et al. [71]	EGA	*FOMs*	Un known
Zhang et al. [72]	DSE	*FOMs*	MS >80%
May et al. [73]	Immune Inspired Algorithm(IIA)	*FOMs*	MS = 89.2%
Ayari et al. [74]	Ant Colony Optimization(ACO)	*FOMs*	MS = 89%
Liu et al. [75]	Improved Iterative Relaxation Method (IIRM)	*FOMs*	MS = 94.2%

Specification mutation is a black box testing, in which errors are inserted into program specifications, while for program mutation is a white box testing where errors are inserted into source code. In program mutation, *MT* is applied on program source code and this code is written in many programming languages. These language such as Fortran language [83–85], Ada language [86,87], C language [88–90], Java language [91–93], and C# language [94–96]. From the previous *HOMT* approaches, more than 66.6% of the studies used Java, while 27.8% used C languages. Table 9 presents the used programming languages for *SOM* and *HOM* testing approaches.

5. Higher-order mutant generation tools

This subsection identifies the mutants generation tools that were applied by *HOMT* approaches and the programming languages which are supported by each tool and the availability status of the tool. *MT* is the process of generating set of software tests and evaluating efficiency of these tests. This process is based on two main steps as follow: First step is mutant generation phase. In this step a mutant is generated by adding a single change into the original program using a mutation operator. Second step is test input generation. After generating the mutants, the test suite is generated and run on each of these mutants.

If a large number of mutation operators are used, it will create many number of mutants. The used mutation operator in the mutants generation process must be different to avoid the creation of equivalent mutants. There are many studies that interest in designing the mutation operators [97–101]. The first set of the mutation operators were created for the Mothra tool [102]. This tool was used to generate the mutants for Fortran 77 programs. Ahmed et al. [103] introduced a survey on object-oriented mutation operators. In the past years, the researchers focused on designing new mutation operators for some purposes such as security problems targeted [104] or the mutation operators for specific language [105].

In the recent years, the mutation operators are mostly used in MT tools. Using of MT in industry depends on applying a fully automated mutation tool. In 1970, it was the year of proposing MT. Consequently many mutation tools were created to generate the mutants. Stages of developing MT tools are three stages. Firstly, between 1977 and 1981 four MT tools were built such as PIMS [84], EXPER [106], and FMS.3 [107] for Fortran and CMS.1 [34] for Cobol. Secondly, from 1982 to 1999 four tools were built in this period such as MOTHRA for FORTRAN [108], PROTEUM and TUMS for C [109]. These three tools are considered as academic tools. The fourth tool is called INSURE++ for C/C++ [110] and is considered as industry tool. The two academic tools MOTHRA and PROTEUM were widely used in this stage. These two tools were mostly used in the advanced mutation techniques, such as weak mutation [111], selective mutation [98], mutant sampling [15], and interface mutation [92]. Thirdly, in the period from 2000 to 2017, there are increasing in the development of MT tools. More than ten tools were built in this period such as Jester and Pester [112], MuJava [113] for Java, Nester [114] for C#, JDAMA [115] for SQL, MUSIC [116] for SQL, MILU [36] for C, and HOMAJ [45] for AspectJ and Java. In the third stage, some of the generated tools are used for FOMs generation such as MuJava [22] and the others are used for FOMs and *HOMs* generation such as MILU [18] and HOMAJ [45]. We noted that MuJava and MILU tools are the most used tools in the previous *HOMT* approaches [18,28,43]. Table 9 presented the MT tools that were used by the previous *HOMT* approaches, tools availability, and the supported programming languages by each tool.

3. Results and discussion

To explore the research questions given in Section 2.1, this section answers the research questions and introduces the threats to validity.

3.1. The exploration of RQ1

RQ1 is designed to investigate how expensive is higher-order mutation testing. Harman et al. [10] showed that *HOMT* is too expensive. They argued that search based techniques such as genetic algorithms can introduce a solution for this problem. According to the results of this SRL given in Table 9 search-based techniques have been successfully used in more than 53% of the techniques used in generating higher-order mutants. In addition as given in Table 10 and Fig. 2, search-based techniques have been used in approximately 100% of the techniques used to generate hard to kill *HOMs*. Search-based techniques can easily explore huge size domains (2000k of mutants) to find the required mutants. As given in Fig. 3, search-based techniques have been successfully used in more than 33% of techniques used to reduce the number of *HOMs*.

3.2. The exploration of RQ2

RQ2 is designed to investigate the efficiency of higher-order mutation testing in generating subtle mutants which are hard to kill. According to this *SLR*, there are three research groups published some works in this topic. The first research group is Harman, Jia, Langdon and others at King's College of London, UK. This group published many papers in the key problems of higher-order mutation testing [10,18,19,27,28,33,36,38]. The second research group includes Nguyen and Madeyski at Faculty of Computer Science and Management, Wroclaw University of Technology, Poland. This

Table 9
Details of extracted data from the primary studies.

Study	Team work	#Publications and Ref.	Year [20xx]	Research Issue and(no. of objectives)	Used technique	FOM generation Tool (availability)	Programming Language	Order of Mutants
S#1	Jia, Harman, Langdon	4[10,18,28,36]	08, 08,09,10	Constructing hard-to-kill *HOMs*. (Single objective)	Greedy Algorithm (GR), Genetic Algorithm (GA), and Hill Climbing Algorithm (HC)	Milu (yes)	C	≥ 2
S#2	Langdon, Harman, Jia	2[19,27]	09, 10	Finding realistic and hard-to-kill *HOMs*. (Multi-objective)	1. A multi-objective Pareto optimal approach using Monte Carlo sampling, genetic algorithms and genetic programming	Milu (yes)	C	≥ 2
S#3	Nguyen, Madeyski	5[30,46,49,50,56]	14, 15, 16, 16, 17	Constructing hard-to-kill *HOMs*. (Single-objective)	Nondominated Sorting Genetic Algorithm Version II (NSGA-II) Two Extension Version of NSGA-II (NSGA-III and eNSGA_II) Steady State Multi-Objective Evolutionary Algorithm (eMOEA)	Judy (yes)	Java	≥ 2
S#4	Nguyen, Madeyski	1[52]	16	Reducing *HOMs* based on test cases (Multi-objective)	Nondominated Sorting Genetic Algorithm Version II (NSGA-II) Two Extension Version of NSGA-II (NSGA-III and eNSGA_II) Steady State Multi-Objective Evolutionary Algorithm (eMOEA)	Judy (yes)	Java	≥ 2
S#5	Omar, Ghosh, Whitly	5[29,31,32,40,45]	12, 13, 14, 14, 17	Constructing hard-to-kill *HOMs*. (Single-objective)	Genetic Algorithm (GA), Local Search (LS), Data-Interaction Guided Local Search (DIGLS), Test-Case Guided Local Search (TCGLS), Restricted Enumeration Search (RES), Restricted Random Search(RRS).	HOMAJ (no)	Java,AspectJ	≥ 2
S#6	Madeysk, Orzeszyn,Torkar, Józala	1[21]	14	Reducing *HOMs* Overcoming equivalent mutants problem (Single-objective)	NeighPair algorithm JudyDiffOp algorithm. RandomMix Last2First	Judy (yes)	Java	= 2
S#7	Polo, Piattini, Garcia-Rodriguez	1[35]	08	Reducing the number of *HOMs*. (Single-objective)	LastToFirst DifferentOperators RandomMix	muJava (yes)	Java	= 2
S#8	Ghiduk	1[43]	14	Reducing the number of *HOMs*. (Single-objective)	Circular Incremental Algorithm (CIA) Random N Algorithm (RNA)	muJava (yes)	Java	≥ 2
S#9	Ghiduk	1[51]	16	Reducing the number of HOMs. (Single-objective)	LastToFirst DifferentOperators Data flow	muJava (yes)	Java	≥ 2
S#10	Lima et al.	1[54]	16	Reducing the number of *HOMs*. (Single-objective)	GA DifferentOperators Each-Choice LastToFirst RandomMix	muJava (yes)	Java	≥ 2
S#11	Papadakis, Malevris, Kintis	2[20,22]	10,10	Reducing of *HOMs*. Comparing weak mutation against strong mutation (Single-objective)	Last2First SameNode SameUnit SU_F2Last SU_DiffOp RMix DiffOp	muJava (yes)	Java	= 2
S#12	Derezińska, Hałas	1[44]	14	Reducing of HOMs. (Single-objective)	Between-Operators (BTO) Each-Choice (ECH) FirstToLast (FTL) Random (RND)	Mutpy(no)	Python	2 and 3
S#13	Mateo, Us-aola,Aleman	1[41]	13	Reducing the number of *HOMs* at system level. (Single-objective)	FirstToLast Each-Choice (ECH) Between-Operators (BTO) Random	Bacterio(no)	Java	= 2
S#14	Akinde	1[25]	12	Reducing equivalent mutants (Single-objective)	NA	Milu (yes)	C	≥ 2
S#15	Harman, Jia,Langdon	1[38]	11	Test data generation (Single-objective)	Dynamic symbolic execution Hill climbing	SHOM(no)	C	≥ 2
S#16	Harman, Jia, Mateo, Polo	1[33]	14	Reducing number of *HOMs* and the number of test cases (Multi-objective)	1. Genetic Algorithm	Bacterio(no)	Java	≥ 2
S#17	Jia, Wu, Harman, Krinke	2[47,55]	15, 16	Using *HOMs* for improving non-functional properties of programs (Multi-objective)	1. NSGA-II	Milu(yes)	C	≥ 2
S#18	Kintis, Papadakis, Malevris	1[26]	12	Using second-order mutants to identify first-order equivalent mutants. (Single-objective)	1. I-EQM considers the execution behavior of both first and second-order mutants, to isolate likely to be first-order equivalent mutants.	JavaLanche(yes)	Java	= 2

Table 10
Details of the studies.

Study	S#1 and S#2	S#3	S#5
No. of subjects	10	3	10
Range of LOC	50 ≤ LoC ≤ 6000	5925 ≤ LoC ≤ 23996	121 ≤ LoC ≤ 14388
Total LOC	14850	43493	19146
Non trivial FOMs	46606	5896	2944
# of Explored HOMs	1000000	as many as possible	1500000
Degree of HOMs	2 ≤ DHOM ≤ 13	2 ≤ DHOM ≤ 15	2 ≤ DHOM ≤ 7+
Overall Average Subtle HOMs	15%	8.74%	14%

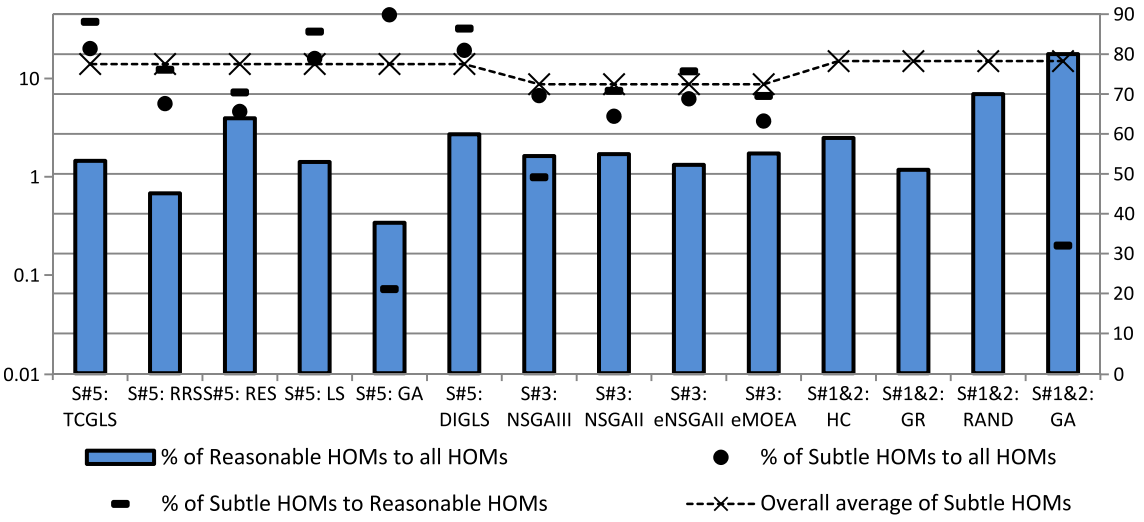

Fig. 2. Algorithms comparison.

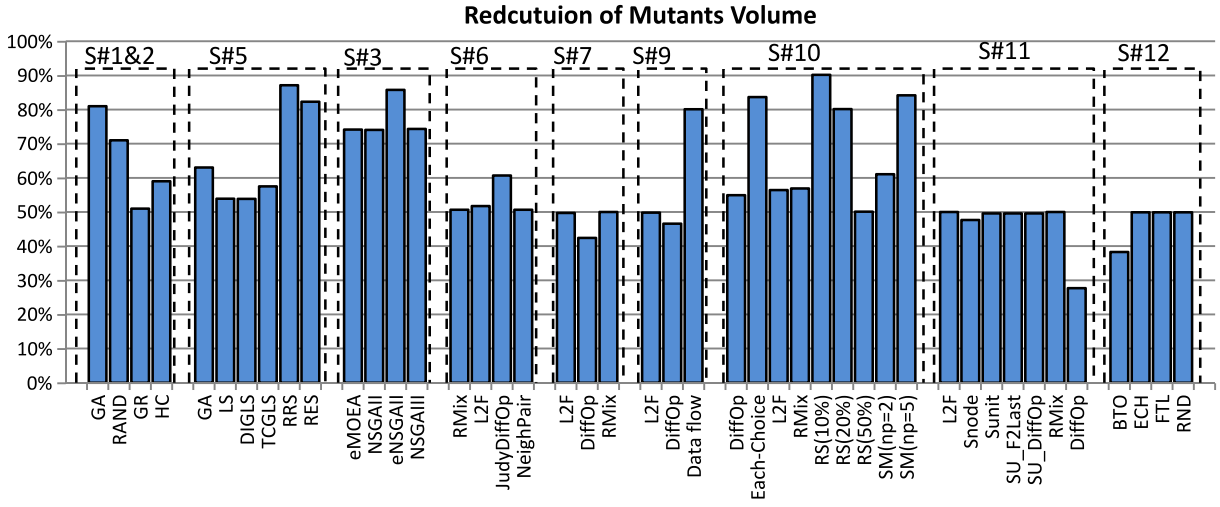

Fig. 3. Comparison of mutants reduction studies.

group presented a new classification of *HOMs* and used optimization techniques for finding subtle *HOMs* [30,46,49,50,52,56]. The third group consists of Omar, Ghosh and Whitly at Colorado State University, USA. This group published some papers in generating subtle higher-order mutants [29,31,32,40,45].

These three research groups recommended different terminologies for mutants that are hard to kill such as subtle *HOMs*, Strong Subsuming *HOMs*, valuable *HOMs*, and difficult to kill *HOMs*. The three research groups applied many search-based techniques for finding hard to kill mutants. The first group applied three metaheuristic algorithms: greedy algorithm (*GR*), genetic algorithm (*GA*), and hill climbing algorithm (*HC*) to find subsuming *HOMs*. The second research group used six techniques: genetic algorithm (*GA*), local search (*LS*), data-interaction guided local search (*DIGLS*), test-case guided local search (*TCGLS*), restricted enumeration search (*RES*), restricted random search (*RRS*). The third research group used four algorithm: nondominated sorting genetic algorithm version II (*NSGA-II*) , two extension version of *NSGA-II* (*NSGA-III* and

eNSGA_II), a steady state multi-objective evolutionary algorithm (*eMOEA*).

Table 10 introduces a comparison between three primary studies (*S#1&2, S#3, and S#5*) interested in constructing hard to kill mutants which is given in Table 9. This comparison shows the total lines of codes of the subject programs in each study, number of non-trivial *FOMs* used to generate *HOMs*, total number of *HOMs* explored by each study, the maximum degree of *HOMs* generated by each study, and the overall average of subtle *HOMs*. The comparison shows that the study S#1&2 is the most effective one in generation subsuming *HOMs*. The mean ratio of subtle *HOMs* to all subsuming *HOMs* generated by studies S#1&2, S#3, S#5 are 15%, 8.74%, and 14%, respectively.

Fig. 2 presents a comparison between fourteen techniques used in the three studies (*S#1&2, S#3*, and *S#5*) according to four criteria: mean ratio of the reasonable *HOMs* to all *HOMs*, mean ratio of subtle *HOMs* to all *HOMs*, mean ratio of subtle *HOMs* to all reasonable *HOMs*, and overall ratio of subtle *HOMs*. In Fig. 2, the first three criteria are referenced to the left axis and the fourth criterion is referenced to the right axis. As a result, the genetic algorithm technique proposed in study S#1&2 is the most effective technique among all techniques in generating subsuming *HOMs* and subtle *HOMs*. The second promising technique is *S#3*: *eNSGAI* I. Although the successful of generating subtle *HOMs*, but the ratio of the generated *HOMs* is small where it is not more than 15%. Therefore, more research work is required to introduce other definitions for subtle mutants instead of subsumed mutants and more effective techniques to generate it. We believe that combining *GA* algorithm proposed in *S#1&2* and *eNSGAII* algorithm proposed in S#3 can enhance the generation of subtle *HOMs*.

3.3. The exploration of RQ3

RQ3 is designed to investigate the equivalent mutant problem which is considered one of the key problem of mutation testing. Equivalent mutant problem is considered as the most explored mutation testing problem [10,18,21,22,24–26,59,117–151]. According to our SLR 43 articles and these are identified. Madeyski et al. [21,129] introduced a survey on equivalent mutant problem. In addition, they studied the impact of second-order mutants on equivalent mutant problem. They classified the equivalent mutant overcoming techniques into four categories: equivalent mutant detection (*EMD*), suggesting equivalent mutant (*SEM*) or impact of equivalent mutant (*IEM*) , avoiding equivalent mutant generation (*AEM*), and higher-order equivalent mutants (*HOEM*). To answer *RQ3*, our *SLR* extended Madeyski's review by augmenting it by the recent published work. Based on the same classification proposed by Madeyski et al. [21,129], we classified the surveyed articles and systems according to these four categories. Table 11 presents brief for the equivalent mutant overcoming techniques with focusing on higher-order mutants techniques. As given in Table 11, *HOMT* have been successfully used in avoiding generation of *EM*, detection of first order *EM*, and reduction of the number of *EM*. Techniques for detection *HOEM* are required.

3.4. The exploration of RQ4

RQ4 is designed to investigate the efficiency of *HOM* testing in reducing the cost of mutation testing. One of the key problems of mutation testing in general and *HOM* testing especially is the explosion of the number of mutants. Where *HOMs* are created by merging different *FOMs*. Suppose $m_{1...n}$ be the number of mutation operators which can be applied at $P_{1...n}$ places in the tested program. Therefore, the number of *FOMs* is $\sum_{i=0}^{n} m_i$; the number of *HOMs* is $\sum_{i=2}^{n} \binom{i}{n} m^i$ [18].

So far there is a significant number of studies which have been performed to reduce the number of *HOMs*. Table 9 reports the primary studies for reducing the cost of higher-order mutation testing. There are nine primary studies (*S#1&2, S#3, S#5, S#6, S#7, S#9, S#10, S#11, and S#12*) focused on reducing the number of higher-order mutants. These studies applied three different methodologies for reducing the number of mutants: i) reducing number of mutation operators ($m_{1...n}$) for example *S#6* [21], *S#7* [35], *S#10* [54], *S#11* [20,22] and *S#12* [44]; ii) selecting the valuable set of *HOMs* for example *S#1&2* [18,28], *S#5* [31], and *S#3* [30,49]; iii) reducing the number of mutated places ($P_{1...n}$) for example *S#9* [51].

These studies applied number of different techniques to reduce the number of *HOMs*. *S#1&2* employed three meta-heuristic algorithms: greedy algorithm (*GR*), genetic algorithm (*GA*), hill climbing algorithm (*HC*) and random search (*RAND*). *S#5* used six techniques: genetic algorithm (*GA*), local search (*LS*), data-interaction guided local search (*DIGLS*), test-case guided local search (*TCGLS*), restricted enumeration search (*RES*), and restricted random search (*RRS*). *S#3* used four algorithm: nondominated sorting genetic algorithm version II (*NSGA-II*), two extension version of *NSGA-II* (*NSGA-III* and *eNSGA _II*), and a steady state multi-objective evolutionary algorithm (*eMOEA*). *S#6* proposed *NeighPair* algorithm and *JudyDiffOp* algorithm. The idea of *JudyDiffOp* algorithm is a modulation of the different operators algorithm [11]. The two algorithms are based on creating *SOMs* by combining *FOMs* but each algorithm combines the mutants in different way. These algorithms are experimented with random search (*RMix*) and (*L2F*) [35]. *S#7* used three techniques last to first (*L2F*) and different operators algorithm (*DiffOp*) and *RMix*. *S#9* used three techniques *L2F* and *DiffOp* and data flow algorithms. *S#10* used six techniques *DiffOp*, L2F, each choice, *RMix*, random search (with ratio 10%, 20%, 50%) *RS*, and selective mutants *SM*. *S#11* used three techniques *DiffOp*, *L2F*, and *RMix*. *S#11* applied the used techniques in single unit or single node. *S#12* used four techniques between operators algorithm (*BTO*), each choice (*ECH*), first to last (*FTL*), and random search (*RND*).

Fig. 3 presents the results of all techniques in each study. The results given in Fig. 3 shows that the reduction ratios of algorithms *GA* in *S#1&2*, *eNSGAII* in *S#3*, *RRS* in *S#5*, *RES in S#5*, data flow in *S#9*, each choice in *S#10*, *RS* in *S#10*, and *SM* in *S#10* are the highest reduction ratios which are 81%, 85.75%, 87.13%, 82.3%, 80.07%, 83.64%, 90.18%, and 84.17%, respectively. Table 12 presents the number of first-order mutants, highest reduction ratio, lowest reduction ratio, and mean reduction ratio for each study.

From the results given in Table 12, we can conclude that studies *S#1&2*, *S#11, S#6*, *S#3* and *S#12* are the most effective studies based on the number of first-order mutants which is in direct propositional with higher-order mutants. By considering mean reduction ratio and number of FOMs, we can conclude that S#1&2 and *S#3* are the most effective studies proposed to reduce the number of generated *HOMs* where 66% and 77.05% of *HOMs* are reduced by the techniques of *S#1&2* and *S#3*, respectively. Fig. 4 presents a comparison between the mutant reduction studies according to number of *FOMs* referenced to right axis and highest, lowest, and mean reduction ratios referenced to left axis.

3.5. The exploration of RQ5

RQ5 is designed to investigate the efficiency of *HOMs* in finding more realistic mutants. To the best of our knowledge, there is no any research work studies the relation between *HOMs* and real faults. Langdon et al. [19,27] considered the complex higher-order mutants which need many changes to fix as real faults. According to this idea, Langdon et al. [19,27] used genetic programming to generate set of higher-order mutants (*2nd, 3rd,* and *4th* order)

Table 11
Equivalent mutant overcoming techniques.

Category	Technique	Reference	Technique Efficiency/Findings
EMD	Compiler based technique	Baldwin and Sayward 1979[138]	NA
		Offutt and Craft 1994[117]	≈ 10%
		Papadakis et al. 2015[131]	≈ 30%
	Constraints based technique	Pan 1994[132], Offutt and Pan 1996, 1997[126,127]	≈ 50%
		Nica and Wotawa 2012[133]Nica 2011[137]	≈ 40%
		Ueshiba and Haga 2014 [121]	Average 63.1% (13% to 100%)
	Program slicing based technique	Hierons et al. 1999 [118]	≈ 47.63%
	Semantic based technique	Ellims et al. 2007 [139]	NA
	Margrave's change-impact analysis	Martin and Xie 2007 [140]	NA
	Lesar model-checker for eliminating equivalent mutants	Bousquet and Delaunay 2008 [141]	NA
SEM/IEM	Code similarity	Kintis and Malevris 2013 [151], Kintis 2016 [122]	≈ 50%
	Static analysis	Kintis and Malevris 2015 [124]Kintis 2016 [122]	≈ 56%
	Data flow patterns	Kintis and Malevris 2014 [134]Kintis 2016 [122]	≈ 70%
	State infection conditions	Just et al. 2013 [136]	≈ 88.9% (8 out of 9 equivalent mutants in a case study)
	Bayesian-learning based technique	Vincenzi et al. 2002 [147]	NA
	impact of EM on coverage	Grün et al. 2009 [130]	Suggests (non-)equivalent mutants
	Impact of dynamic invariants	Schuler et al. 2009 [148]	NA
	Changes in coverage to find EM	Schuler and Zeller 2010, 2013 [59,149]	25% to 70%
	Software Anomaly Detection	Arcaini et al 2015, 2017 [128] [119]	detecting static anomalies
AEM	Estimate the ability of mutation operators to find equivalent mutants	Umar 2006 [120]	Find the mutation operators which can generate equivalent mutants
	Selective mutation	Mresa and Bottaci 1999 [142]	NA
	Program dependence analysis	Harman et al. 2001 [143]	NA
	Co-evolutionary search techniques	Adamopoulos et al. 2004 [123]	Avoids equivalent mutant generation
	Equivalency conditions	Offutt et al. 2006 [144]	NA
	Fault hierarchy	Kaminski and Ammann 2009 [145]	Avoids equivalent mutant generation
	Semantic exception hierarchy	Chen et al. 2009 [146]	Proposed mutation operators for Java exception handling constructs
	Classification of mutants	Papadakis et al. 2014 [125]	Kill 92% of all the killable mutants.
HOMs	Avoiding Generation of Equivalent Mutant	Jia and Harman 2009 [18]	NA
		Kintis et al. 2010 [22], Kintis 2016 [122]	EM is reduced from 65.6% to 86.8%
		Offutt 1992 [24]	≈ 0.53% to 1.4% 2nd HOMs are EM.
		Papadakis and Malevris 2010 [20]	EM is reduced from 80% to 90%
		Akinde 2012 [25]	1st Order 7.6% to 28.8% 2^{nd} HOM 3.5% to 4.5%
	Using HOMT to isolate FOEM	Kintis et al. 2012 [26], Kintis et al. 2015 [150], Kintis 2016 [122]	Precision score of 71% and a recall value of 81%

Table 12
Highest, lowest, and mean reduction ratio for each study.

Study	No. of FOMs	Highest reduction ratio	Lowest reduction ratio	Mean reduction ratio
S#1&2	94493	81%	51%	66%
S#5	2944	87.13%	53.84%	66.28%
S#3	5896	85.75%	74.03%	77.05%
S#6	46619	60.67%	50.67%	53.44%
S#7	1026	50.00%	42.40%	47.37%
S#9	1114	80.07%	46.59%	58.83%
S#10	1883	85.4%	54.9%	67.4%
S#11	62714	49.99%	27.68%	46.30%
S#12	4876	49.9%	38.3%	47.0%

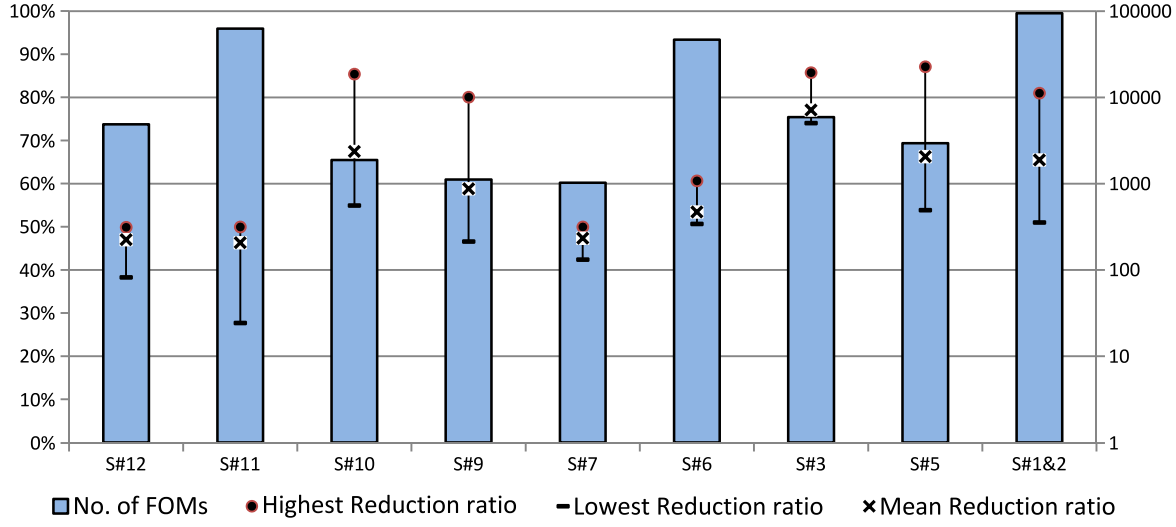

Fig. 4. Highest, lowest, and mean reduction ratio for each study.

which are hard to kill (killed by one test case). This set of mutants satisfies two objectives: semantic distance (number of test cases which cause a mutant and original program behave in a different way) and syntactic distance (sum the number of changes in the logical control structure). In triangle program, the number of mutants of *1st, 2nd, 3rd*, and *4th* are 85, 3400, 85000, and 1487500, respectively. The number of hardest to kill mutants of *1st, 2nd*, *3rd*, and *4th* are 18, 325, 2615, and 12363, respectively.

A few of studies in traditional mutation testing (1st order) concentrated on some issues related to realism problem [152–156]. Though, these studies are out the scope of attention of this *SLR*, we summarize the results of these studies to be a road map for similar studies in *HOMT*. Daran and Thévenod-Fosse [155] introduced the first experimental study to compare programs errors created by real faults and those created by 1st order mutations. The study included 1458 errors created by real faults and 2272 created by mutations. The results of the experiments showed that 85% of 2272 errors created by mutations were also created by real faults. Andrews et al. [153] explored the relation between mutants, hand-seeded faults, real faults. They concluded that mutants are better substitution for real faults than hand-seeded faults. Namin and Kakarla [156] replicated the work of Andrews et al. [153] and concluded that the correlation between the mutation score and fault detection is weak for one of the subjects. Namin and Kakarla [156] studied the properties of mutants and real faults. They concluded that mutants and real faults sometimes behave in different way (e.g. mutant detection ratio is 0.512, while fault detection ratio 0.686). Just et al. [152] studied coupling effect between mutants and real faults. They concluded that coupling effect ratio is 73%. Gopinath et al. [154] studied competent programmer hypothesis. They concluded that a typical-change modifies about three to four tokens or ten tokens especially if at least 80% of the real faults are included. They claimed that understanding of the competent programmer hypothesis may be incorrect. Table 13 introduces a comparison of studies that explored the relation between first-order mutants and real faults.

4. The findings of this SLR

Through the answers of the research questions and the reporting review section, we come up with the following findings.

4.1. Work have been done

4.1.1. Overcoming the high cost and expensiveness of HOMT

The high cost and expensiveness of higher-order mutation testing is due to the huge number of the generated *HOMs* which can be created by combining the *FOMs*. This numerous amount of mutants makes the process of finding good mutants is very costly. According to the results of this SLR, overcoming this problem can be done through two different strategies. The first strategy is using search based techniques to find good representative small set of *HOMs*; where search based techniques such as genetic algorithm, greedy algorithm, and hill climbing algorithm, and random search as well have been successfully used to find subtle *HOMs* among more than 2000000 mutants (see Table 9 and Table 10). The second strategy is using tactics to reduce the number of *HOMs*. According to the results of this *SLR*, the researchers used three different tactics to reduce the number of *HOMs*: (1) reducing the number of mutation operators which consequently reduces the number of *FOMs* and number of *HOMs*; (2) selecting subset of *HOMs* instead of all mutants such as subtle set (e.g. strong subsuming *HOMs* which is considered the most valuable set of *HOMs*); (3) reducing the

Table 13
Comparison of studies that explored the relation between mutants and real faults.

Study	Research issueMs: mutantsRF: real fault	No. and categories of mutation operators	No. of RF versions	No. of 1st Ms	Ratio of mutant evaluated	Results
Daran and Thévenod-Fosse [155]	Similarity in behavior between Ms and RF.	24 operators3 categories: replacing constant, identifier, or operator	1458	2272	1%	85% of errors created by mutations were created by real faults
Andrews et al. [153]	Whether Ms or hand-seeded faults are representative of RF.	32 operators 4 categories: replacing constant, or operator, negate branch condition, or delete statement	38space program	11379	10%	Mutation score and real faults detection rate is very close.
Namin and Kakarla [156]	Similarity in properties between Ms and RF.	The same in [153]	38space program	301400	10%	Mutant detection ratio is 0.512, while fault detection ratio 0.686.
Just et al. [152]	Coupling effect between Ms and RF.	NA operators4 categories: replacing constant, or operator, deleting statement or modifying branch condition	357	230000	100%	Coupling effect ratio is 73%
Gopinath et al. [154]	Competent programmer hypothesis.	≈ 77 operators9 categories: add, replace, or remove tokens, add or remove +/- 1, change in constant value, changing a constant to a variable, changing a variable to another variable, changing a binary operator to another, negation of a value.	240000 patches			A typical-change modifies about three to four tokens which maybe ten if 80% of the real faults are included.

mutated locations in the original programs using some techniques such as data flow analysis.

4.1.2. Coping with the realism problem

Although, there is a number of researchers studied the key issues of realism problem of traditional mutation testing such as coupling effect between FOMs and real faults, competent programmer hypothesis, similarity in behavior and properties between FOMs and real faults (see Table 13 for more details), these issues are not studied for higher-order mutation testing. Only complex property of higher-order mutants and real faults has been studied by Langdon et al. [19,27].

4.1.3. Solution of the equivalent mutant problem

According to the results of our SLR, there are a wide number of articles concentrated on equivalent mutant problem for traditional mutation testing and higher-order mutation testing as well. The equivalent mutant overcoming techniques for traditional mutation testing focused on equivalent mutant detection, suggesting equivalent mutant or studying the impact of equivalent mutant, and avoiding equivalent mutant generation. The equivalent mutant overcoming techniques for higher-order mutation testing concentrated mainly on avoiding generation of higher-order equivalent mutants (for more details see Table 11).

4.2. Work to be done

Although there is a significant number of articles has been published in the area of higher-order mutation testing, there are many key issues are not considered yet. The following subsections discus the work to be done in *HOMT* field.

4.2.1. Overcoming the high cost of HOMT

According to the results of this *SLR*, there are additional work to be done for overcoming the high cost of *HOMT*. From the preceding discussion, the high cost of *HOMT* can be decreased by using search-based techniques or by reducing the number of mutants. Many issues need investigation such as determining the most effective search based technique for generating *HOMs* (which can be done by empirical comparison between the common techniques such as genetic, particle swarm, ant colony, and bat algorithms), suggesting another representative subsets of *HOMs* rather than subsuming set to be the target of *HOMT*, and suggesting another methodologies for reducing number of mutants using criteria such as paths, branches, or data flow etc.

4.2.2. Coping with the realism problem

There are many key issues needed to be investigate in the realism problem of higher-order mutation testing such as coupling effect between *HOMs* and real faults, competent programmer hypothesis, similarity in behavior and properties between *HOMs* and real faults.

4.2.3. Solution of the equivalent mutant

Although there are some techniques have been proposed to avoid generation of higher-order equivalent mutants, there are some open works such as (1) detecting higher-order equivalent mutants, (2) employing first-order equivalent mutants to generate killable higher-order mutants.

4.2.4. Test data generation

Techniques for killing the different types of *HOMs* are needed where only one research was presented to find test data for killing *SHOMs*.

4.3. Threats to validity

4.3.1. Difficulties in finding all the studies that are related to our SLR

This problem is considered one of the major problems of *SLRs* [157]. We used databases that were used before in [158] to search for the sources. We also used keywords during searching to select the primary studies about *HOMT*. If these studies do not describe their objectives about *HOMT*, these studies may have been removed. We applied the inclusion and exclusion criteria on the selected papers to determine which paper would be the best for our *SLR*.

4.3.2. The difficulty in classifying the studies

We tried to classify the approaches according to *HOMs* generation approaches and test input generation approaches. We found the majority of *HOMT* testing approaches were designed to generate *HOMs* and the work on generating test case for killing *HOMs* are so little.

4.3.3. The difficulty in data extracting

When we wanted to introduce the effectiveness of some techniques that were used in selected approaches, we found some of these papers did not give a complete description to the technique. Due to this limitation, we were unable to compare these techniques and offer a complete view of their effectiveness.

4.3.4. Writing languages of some papers

Some publications are written by languages which are not considered in this *SLR* in which English language is the basic language for selecting studies.

5. Conclusion and future work

Mutation testing is the process which can be used to drive the test data development process and assess the quality of these data, in terms of its ability to find faults. Mutation testing can be divided into two categories first and higher-order mutation testing. Since 2008, there has been an upsurge in interest in *HOMT*, with many authors examining the way in which *HOMT* could find subtle hard to kill faults, capture partial fault masking, reduce equivalent mutants problem, reduce test effort while increasing effectiveness, and capture more realistic faults than those captured by simple insertion of first-order mutants. Because of this upsurge of interest in the previous activities, this paper presented the first systematic literature review specifically targeted at a higher-order mutation. This SLR analyzed the results of more than one hundred sixty research articles in this area. This SLR presented qualitative results and bibliometric analysis for the surveyed articles. It augmented these results with scientific findings and quantitative results from the primary literature. In addition, it summarized the work which has been done to overcome the key obstacles of *HOMT*. This *SLR* deduced the following results:

1. The trend of publications in *HOMT* is directly proportional with the time.
2. Most studies (55.6%) concentered on reducing number of *HOMs*. While there are a very few number of studies concentered on constructing more realistic *HOMs* (5.6%), and genetic improvement (5.6%). Systematic-based methods (100%) and search-based methods (100%) are the most used strategies in *HOMT*. Genetic algorithm (52.6%), last to first (36.8%), and fair random (42.1%) are the most frequency techniques in *HOMT*.
3. Most test data generation approaches aim to kill FOMs and the work on killing *HOMs* is very limited.

4. More than 66.6% of the studies used Java, while 27.8% used C languages.
5. *MuJava* and *MiLu* tools are the most used tools in *HOMT*.
6. Search-based techniques can easily explore huge size domains (2000k) of mutants to find the required mutants.
7. The mean ratio of subtle *HOMs* to all subsuming *HOMs* generated by the primary studies are between 8.74%, and 15%.
8. The ratio of subtle *HOMs* is small where it is not more than 15%. Therefore, more work is required to introduce other definitions for subtle mutants instead of subsumed mutants and more effective techniques to generate it.
9. *HOMT* have been successfully used in avoiding generation of EM, detection of first order EM, and reduction of the number of EM. Techniques for detection HOEM are required.
10. Mean reduction ratio of the number of generated *HOMs* is between 66% and 77.05%.
11. There is no any research work studies the relation between *HOMs* and real faults.

As a result of this work, this SLR presented an outline for many work to be done in *HOMT*. Our future work will concentrate on doing meta-analysis for the results of this *SLR* and doing some statistical comparisons among the primary studies in each key issue of *HOMT*.

References

[1] B. Bezier, Software Testing Techniques, second ed., Van Nostrand Reinhold Co., New York, NY, USA, 1990.
[2] G.J. Myers, C. Sandler, The Art of Software Testing, John Wiley & Sons, 2004.
[3] L. Copeland, A Practitioner's Guide To Software Test Design, Artech House, Inc., Norwood, MA, USA, 2003.
[4] R.V. Binder, Testing Object-Oriented Systems: Models, Patterns, and Tools, Addison Wesley Longman Publishing Co. Inc., Boston, MA, USA, 1999.
[5] G. Kapfhammer, The Computer Science Handbook, Chapter Software Testing, edition 2nd., CRC Press, 2004.
[6] S. Schach, Testing: Principles and practice, ACM Comput. Surv. 28 (1) (1996) 277–279.
[7] H. Zhu, P. Hall, J. May, Software unit test coverage and adequacy, ACM Comput. Surv. 29 (4) (1997) 366–427.
[8] R.A. DeMillo, R.J. Lipton, F.G. Sayward, Hints on test data selection: Help for the practicing programmer, Computer 11 (4) (1978) 34–41.
[9] R.G. Hamlet, Testing programs with the aid of a compiler, IEEE Trans. Softw. Eng. 3 (4) (1977) 279–290.
[10] M. Harman, Y. Jia, W.B. Langdon, A manifesto for higher order mutation testing, in: Third International Conference on Software Testing, Verification, and Validation Workshops, ICSTW, 2010, pp. 80–89.
[11] W. Howden, Weak mutation testing and completeness of test sets, IEEE Trans. Softw. Eng. 8 (4) (1982) 371–379.
[12] D. Schuler, A. Zeller, Javalanche: Efficient mutation testing for java, in: Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, ESEC/FSE'09, New York, NY, USA, 2009, pp. 297–298.
[13] A.J. Offutt, R.H. Untch, Mutation Testing for the New Century, Chapter Mutation 2000: Uniting the Orthogonal, Kluwer Academic Publishers, Norwell, MA, USA, 2001, pp. 34–44.
[14] R. DeMillo, E. Krauser, A. Mathur, Compiler-integrated program mutation, in: Computer Software and Applications Conference, COMPSAC'91, Proceedings of the Fifteenth Annual International, 1991, pp. 351–356.
[15] A.P. Mathur, W.E. Wong, An empirical comparison of mutation and data low based test adequacy criteria, 1993.
[16] M.A. Cachia, M. Micallef, C. Colombo, Towards incremental mutation testing, in: Electronic Notes in Theoretical Computer Science, Proceedings of the 2013 Validation Strategies for Software Evolution, VSSE Workshop, 2013, pp. 2–11.
[17] Y. Jia, M. Harman, An analysis and survey of the development of mutation testing, IEEE Trans. Softw. Eng. 37 (5) (2011) 649–678.
[18] Y. Jia, M. Harman, Higher order mutation testing, J. Inf. Softw. Technol. 51 (10) (2009) 1379–1393.
[19] W.B. Langdon, M. Harman, Y. Jia, Efficient multi-objective higher order mutation testing with genetic programming, J. Syst. Softw. 83 (2010) 2416–2430.
[20] M. Papadakis, N. Malevris, An empirical evaluation of the first and second order mutation testing strategies, in: Proceedings of the 2010 Third International Conference on Software Testing, Verification, and Validation Workshops, Ser. ICSTW'10, IEEE Computer Society, 2010, pp. 90–99.
[21] L. Madeyski, W. Orzeszyna, R. Torkar, M. Józala, Overcoming the equivalent mutant problem: A systematic literature review and a comparative experiment of second order mutation, IEEE Trans. Softw. Eng.. 40 (1) (2014) 23–44.
[22] M. Kintis, M. Papadakis, N. Malevris, Evaluating mutation testing alternatives: A collateral experiment, in: Proc. 17th Asia Pacific Soft. Eng. Conf., APSEC, 2010.
[23] B. Kitchenham, S. Charters, Guidelines for performing systematic literature reviews in software engineering, in: Evidence-Based Software Engineering, 2007.
[24] A.J. Offutt, Investigations of the software testing coupling effect, ACM Trans. Softw. Eng. Methodol. 1 (1) (1992) 5–20.
[25] A.O. Akinde, Using higher order mutation for reducing equivalent mutants in mutation testing, Asian J. Comput. Sci. Inf. Technol. 2 (3) (2012) 13–18.
[26] M. Kintis, M. Papadakis, N. Malevris, Isolating First Order Equivalent Mutants via Second Order Mutation, in: IEEE Fifth International Conference on Software Testing, Verification and Validation, 2012, pp. 701–710.
[27] W.B. Langdon, M. Harman, Y. Jia, Multi objective mutation testing with genetic programming, in: 4th Testing Academia and Industry Conference — Practice and Research Techniques, TAIC PART'09, IEEE press, Windsor, UK, 2009, pp. 21–29.
[28] Y. Jia, M. Harman, Constructing Subtle Faults Using Higher Order Mutation Testing, in: Proceedings of the 8th International Working Conference on Source Code Analysis and Manipulation, SCAM'08, Beijing, China, 2008, pp. 249–258.
[29] E. Omar, S. Ghosh, D. Whitely, Constructing subtle higher order mutants for Java and AspectJ programs, in: Proceedings of the 24th IEEE International Symposium on Software Reliability Engineering, ISSRE, 2013, , pp. 340–349.
[30] Q.V. Nguyen, L. Madeyski, Searching for strongly subsuming higher order mutants by applying multi-objective optimization algorithm, in: H.A. Le Thi, N.T. Nguyen, T.V. Do (Eds.), Advanced Computational Methods for Knowledge Engineering, in: Vol. 358 of Advances in Intelligent Systems and Computing, Springer, 2015, pp. 391–402. http://dx.doi.org/10.1007/978-3-31917996-4_35.
[31] E. Omar, S. Ghosh, D. Whitley, Subtle higher order mutants, Inf. Softw. Technol. 81 (2017) 3–18.
[32] E. Omar, S. Ghosh, D. Whitley, Comparing search techniques for finding subtle higher order mutants, in: GECCO'14, Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, 2014, pp. 1271–1278.
[33] M. Harman, Y. Jia, P. Reales Mateo, M. Polo, Angels and monsters: An empirical investigation of potential test effectiveness and efficiency improvement from strongly subsuming higher order mutation, in: Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering, ASE'14, ACM, New York, NY, USA, 2014, pp. 397–408.
[34] A.T. Acree, On Mutation, Georgia Inst. Of Technology, 1980, Ph.D. Thesis.
[35] M. Polo, M. Piattini, I. Garcia-Rodriguez, Decreasing the cost of mutation testing with second-order mutants, Softw. Test. Verif. Reliab. 19 (2) (2008) 111–131.
[36] Y. Jia, M. Harman, MILU: A customizable, runtime-optimized higher order mutation testing tool for the full c language, in: Proceedings of the 3rd Testing: Academic and Industrial Conference Practice and Research Techniques, TAIC PART'08, IEEE Computer Society, Windsor, UK, 2008, pp. 94–98.
[37] S. Kapoor, Test case effectiveness of higher order mutation testing, Int. J. Comput. Technol. Appl. 2 (5) (2011) 1206–1211.
[38] M. Harman, Y. Jia, 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'11, 2011, pp. 212–222.
[39] E. Blanco-Muñoz, A. García-Domínguez, J.J. Domínguez-Jiménez, I. Medina-Bulo, Towards higher-order mutant generation for WS-BPEL, in: Proceedings of the International Conference on e-Business, 2011, pp. 1–6.
[40] E. Omar, S. Ghosh, An Exploratory Study of Higher Order Mutation Testing in Aspect-Oriented Programming, in: Proceedings of the 23rd IEEE International Symposium on Software Reliability Engineering, ISSRE 2012, 2012, pp. 1–10.
[41] P.R. Mateo, M.P. Usaola, J.L.F. Aleman, Validating 2nd-order mutation at system level, IEEE Trans. Softw. Eng. 39 (4) (2013) 570–587.
[42] Y. Jia, Higher Order Mutation Testing, University College London (UCL), 2013, Doctoral Thesis.
[43] A.S. Ghiduk, Using evolutionary algorithms for higher-order mutation testing, IJCSI Int. J. Comput. Sci. 11 (2) (2014) 93–104.
[44] A. Derezińska, K. Hałas, Experimental evaluation of mutation testing approaches to python programs, in: Proc. of 7th IEEE Inter. Conf. on Software Testing Verification and Validation Workshops, ICSTW, IEEE Comp. Soc, 2014, pp. 156–164.

[45] E. Omar, S. Ghosh, D. Whitley, HOMAJ: A Tool for Higher Order Mutation Testing in AspectJ and Java, Software Testing, in: Verification and Validation Workshops, ICSTW, IEEE Seventh International Conference on, 2014, pp. 165–170.
[46] Q.V. Nguyen, L. Madeyski, Problems of mutation testing and higher order mutation testing, in: Advanced Computational Methods for Knowledge Engineering, in: Vol. 282 of the series Advances in Intelligent Systems and Computing, 2014, pp. 157–172.
[47] Y. Jia, F. Wu, M. Harman, J. Krinke, Genetic Improvement using Higher Order Mutation, in: GECCO Companion'15: Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation, 2015, pp. 803–804.
[48] Y Jia, M Merayo, M Harman, Introduction to the special issue on mutation testing, Softw. Test. Verif. Reliab. 25 (5–7) (2015) 461–463.
[49] Q.V. Nguyen, L. Madeyski, Empirical evaluation of multi-objective optimization algorithms searching for higher order mutants, in: Cybernetics and Systems — Smart Experience and Knowledge Engineering for Optimization, Learning, and Classification/Recommendation Problems, vol. 47, 2016, pp. 48–68.
[50] Q. Vu Nguyen, L. Madeyski, On the relationship between the order of mutation testing and the properties of generated higher order mutants, in: Ngoc Thanh Nguyen, Bogdan Trawiński, Hamido Fujita, Tzung-Pei Hong (Eds.), Intelligent Information and Database Systems, ACIIDS 2016, in: Lecture Notes in Artificial Intelligence, vol. 9621, Springer-Verlag, Berlin Heidelberg, 2016.
[51] A.S. Ghiduk, Reducing the number of higher-order mutants with the aid of data flow, e-Inform. Softw. Eng. J. 10 (2016) 31–49.
[52] Q.V. Nguyen, L. Madeyski, Higher order mutation testing to drive development of new test cases: An empirical comparison of three strategies, in: N.T. Nguyen, B. Trawiński, H. Fujita, T.-P. Hong (Eds.), Intelligent Information and Database Systems, ACIIDS 2016, in: vol. 9621 of Lecture Notes in Artificial Intelligence, Springer, 2016, pp. 235–244. http://dx.doi.org/10.1007/978-3-662-49381-6_23.
[53] S. Tokumoto, H. Yoshida, K. Sakamoto, S. Honiden, MuVM: Higher Order Mutation Analysis Virtual Machine for C, in: 2016 IEEE International Conference on Software Testing, Verification and Validation, ICST, Chicago, IL, 2016, pp. 320–329.
[54] J.A.P. Lima, G. Guizzo, S.R. Vergilio, A.P.C. Silva, H.L. Jakubovski Filho, H.V. Ehrenfried, Evaluating Different Strategies for Reduction of Mutation Testing Costs, in: Proceedings of the 1st Brazilian Symposium on Systematic and Automated Software Testing, Article No. 4, 2016.
[55] F. Wu, M. Harman, Y. Jia, J. Krinke, HOMI: Searching Higher Order Mutants For Software Improvement, in: Symposium on Search-Based Software Engineering, Raleigh, NC, USA, October 2016.
[56] Q.V. Nguyen, L. Madeyski, Addressing mutation testing problems by applying multi-objective optimization algorithms and higher order mutation, J. Intell. Fuzzy Systems 32 (2017) 1173–1182.
[57] B. Kitchenham, S. Charters, Guidelines for performing systematic literature reviews in software engineering, Technical Report EBSE, 2007.
[58] K.S. Khan, G.Ter. Riet, J. Glanville, A.J. Sowden, J. Kleijnen, Undertaking systematic reviews of research on effectiveness CRDs guidance for those carrying out or commissioning reviews, Technical Report, 2001.
[59] D. Schuler, A. Zeller, (Un-) Covering Equivalent Mutants, in Software Testing, Verification and Validation, in: ICST, 2010 Third International Conference on, 2010, pp. 45–54.
[60] W.B. Langdon, Genetic Programming and Data Structures: Genetic Programming Data Structures = Automatic Programming!, Boston. 1998.
[61] R. DeMillo, A.J. Offutt, Constraint-based automatic test data generation, IEEE Trans. Softw. Eng. 17 (9) (1991) 900–910.
[62] L.J. Morell, A theory of fault-based testing, IEEE Trans. Softw. Eng. 16 (8) (1990) 844–857.
[63] L.T.M. Hanh, K.T. Tung, N.T. Binh, Mutation-based test data generation for simulink models using genetic algorithm and simulated annealing, Int. J. Comput. Inf. Technol. 3 (2014) 763–771.
[64] M. Papadakis, N. Malevris, Searching and generating test inputs for mutation testing, J. Springer Plus 2 (2013) 1–12.
[65] J. Louzada, C.G. Camilo-Junior, A. Vincenzi, C. Rodrigues, An elitist evolutionary algorithm for automatically generating test data, in: World Congress on Computational Intelligence, WCCI'12, IEEE, 2012, pp. 1–8.
[66] R. Malhotra, M. Garg, An adequacy based test data generation technique using genetic algorithms, J. Inf. Process. Syst. 7 (2) (2011) 363–384.
[67] M. Papadakis, N. Malevris, M. Kallia, Towards automating the generation of mutation tests, in: 5th Workshop on Automation of Software Test, 2010, pp. 111–118.
[68] M. Rad, F. Akbari, A. Bakht, Implementation of common genetic and bacteriological algorithms in optimizing testing data in mutation testing, in: Computational Intelligence and Software Engineering, CiSE, 2010 International Conference on, 2010, pp. 1–6.
[69] G. Fraser, A. Zeller, Mutation-driven generation of unit tests and oracles, in: International Symposium on Software Testing and Analysis, ISSTA'10, 2010, pp. 147–157.
[70] M. Papadakis, N. Malevris, Automatic mutation test case generation via dynamic symbolic execution, in: 21st International Symposium on Software Reliability Engineering, ISSRE'10, 2010, pp. 121–130.
[71] K.K. Mishra, S. Tiwari, A. Kumar, A. Misra, An approach for mutation testing using elitist genetic algorithm, in: International Conference on Computer Science and Information Technology, IEEE, 2010, pp. 426–429.
[72] L. Zhang, T. Xie, N. Tillmann, J. Halleux, H. Mei, Test generation via dynamic symbolic execution for mutation testing, in: International Conference on Software Maintenance, ICSM'10, 2010.
[73] P. May, J. Timmis, K. Mander, Immune and evolutionary approaches to software mutation testing, in: Proceedings of the 6th International Conference on Artificial Immune Systems. ICARIS'07, Springer-Verlag, 2007, pp. 336–347.
[74] K. Ayari, S. Bouktif, G. Antoniol, Automatic mutation test input data generation via ant colony, in: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, GECCO'07, ACM, 2007, pp. 1074–1081.
[75] M.H. Liu, Y.F. Gao, J.H. Shan, J.H. Liu, L. Zhang, J.S. Sun, An approach to test data generation for killing multiple mutants, in: Proceedings of the 22nd IEEE International Conference on Software Maintenance. ICSM'06, IEEE Computer Society, 2006, pp. 113–122.
[76] S.J. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, second edn, Pearson Education, 2003, Chapter 4.
[77] M. Dorigo, G. Di Caro, The ant colony optimization meta-heuristic, in: New Ideas in Optimization, 1999, pp. 11–32.
[78] T. Back, Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms, Oxford University Press, New York, 1996.
[79] D. Whitley, An overview of evolutionary algorithms: Practical issues and common pitfalls, Inf. Softw. Technol. 43 (14) (2001) 817831.
[80] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, first ed., Addison-Wesley Longman Publishing Co. Inc., Boston, MA, USA, 1989.
[81] A.S. Gopal, T.A. Budd, Program Testing By Specification Mutation, Technical Report TR,, University of Arizona, Tucson, Arizona, 1983.
[82] R.A. DeMillo, Program Mutation: An Approach To Software Testing, Technical Report, Georgia Institute of Technology, 1983.
[83] Centre for Reviews and Dissemination, What are the criteria for the inclusion of reviews on DARE? 2007. Available at http://www.york.ac.uk/inst/crd/faq4.htm.
[84] T.A. Budd, R.A. DeMillo, R.J. Lipton, F.G. Sayward, The design of a prototype mutation system for program testing, in: Proceedings of the AFIPS National Computer Conference, Vol. 74, ACM, Anaheim, New Jersey, 1978, pp. 623–627.
[85] R.J. Lipton, F.G. Sayward, The Status of Research on Program Mutation, in: Proceedings of the Workshop on Software Testing and Test Documentation, 1978, pp. 355–373.
[86] A.J. Offutt, J. Voas, J. Payn, Mutation Operators for Ada, Technique Report ISSE, George Mason University, Fairfax, Virginia, 1996 PP.96-09.
[87] J.H. Bowser, Reference Manual for Ada Mutant Operators, Technique Report GITSERC- 88/02, Georgia Institute of Technology, Atlanta, Georgia, 1988.
[88] H. Agrawal, R.A. DeMillo, B. Hathaway, W. Hsu, W. Hsu, E.W. Krauser, R.J. Martin, A.P. Mathur, E. Spafford, Design of Mutant Operators for the C Programming Language, Technical Report SERC-TR-41-P, Purdue Univ., 1989.
[89] M.E. Delamaro, J.C. Maldonado, A.P. Mathur, Interface mutation: An approach for integration testing, IEEE Trans. Softw. Eng. 27 (3) (2001) 228–247.
[90] H. Shahriar, M. Zulkernine, Mutation-Based Testing of Buffer Overflow Vulnerabilities, in: Proceedings of the 2nd Annual IEEE International Workshop on Security in Software Engineering, Turku, Finland, 2008, pp. 979–984.
[91] M.E. Delamaro, J.C. Maldonado, Interface Mutation: Assessing Testing Quality at Inter procedural Level, in: Proceedings of the 19thInternational Conference of the Chilean Computer Science Society, SCCC'99, Talca, Chile,1999, pp. 78–86.
[92] M.E. Delamaro, J.C. Maldonado, A.P. Mathur, Integration Testing Using Interface Mutation, in: Proceedings of the seventh International Symposium on Software Reliability Engineering, ISSRE'96, White Plains, New York, 1996, pp. 112–121.
[93] S. Kim, J.A. Clark, J. A. McDermid, Class Mutation: Mutation Testing for Object-oriented Programs, in: Proceedings of the Net. Object Days Conference on Object-Oriented Software Systems. 2000.
[94] A. Derezinska, Object-oriented Mutation to Assess the Quality of Tests, in: Proceedings of the 29th Euro micro Conference, Belek, Turkey, 2003, pp. 417–420.
[95] A. Derezinska, Advanced Mutation Operators Applicable in C# Programs, Technique Report, Warsaw University of Technology, Warszawa, Poland, 2005.

[96] A. Derezinska, Quality Assessment of Mutation Operators Dedicated for C# Programs, in: Proceedings of the 6th International Conference on Quality Software, QSIC'06, Beijing, China, 2006.
[97] A.S. Namin, J.H. Andrews, D. Murdoch, Sufficient mutation operators for measuring test effectiveness. in: Software Engineering, ICSE'08 ACM/IEEE 30th International Conference on, 2008, pp. 351–360.
[98] A.J. Offutt, A. Lee, G. Rothermel, R.H. Untch, C. Zapf, An experimental determination of sufficient mutant operators, ACM Trans. Softw. Eng. Methodol. 5 (2) (1996) 99–118.
[99] Y.-S. Ma, A.J. Offutt, Y.-R. Kwon, Inter-class mutation operators for java, in: Software Reliability Engineering, Proceedings. 13th International Symposium on, 2002, pp. 352–363.
[100] H.Y. Chen, S. Hu, Two New Kinds of Class Level Mutants for Object-Oriented Programs, 2006.
[101] A. Derezinska, M. Rudnik, Quality evaluation of object-oriented and standard mutation operators applied to c# programs, in: Proceedings of the 50th International Conference on Objects, Models, Components, Patterns, TOOLS'12, Springer-Verlag, Berlin, Heidelberg, 2012, pp. 42–57.
[102] K.N. King, A.J. Offutt, A fortran language system for mutation-based software testing, Softw. - Pract. Exp. 21 (7) (1991) 685–718.
[103] Z. Ahmed, M. Zahoor, I. Younas, Mutation Operators for Object-Oriented Systems: A Survey, 2010.
[104] H. Shahriar, M. Zulkernine, Mutation-Based testing of format string bugs, in: Proceedings of the 2008 11th IEEE High Assurance Systems Engineering Symposium, HASE'08, IEEE Computer Society, Washington, DC, USA, 2008, pp. 229–238, Sch. of Comput. Queen's Univ. Kingston, ON.
[105] J.S. Bradbury, J.R. Cordy, J. Dingel, Mutation operators for concurrent java, j2se 50, in: Proceedings of the Second Workshop on Mutation Analysis, MUTATION'06, IEEE Computer Society, Washington, DC, USA, 2006, Sch. of Comput. Queen's Univ. Kingston, ON.
[106] T.A. Budd, R. Hess, F.G. Sayward, EXPER Implementor'S Guide, Technique Report, Yale University, New Haven, Connecticut, 1980.
[107] A. Tanaka, Equivalence Testing for FORTRAN Mutation System using Data Flow Analysis, Georgia Institute of Technology, Atlanta, Georgia, 1981, Master Thesis.
[108] R.A. DeMillo, D.S. Guindi, K.N. King, W.M. McCracken, A.J. Offutt, An extended overview of the mothra software testing environment, in: Proceedings of the 2nd Workshop on Software Testing, Verification, and Analysis, TVA'88, IEEE Computer Society, Banff Alberta, Canada, 1988, pp. 142–151.
[109] M.E. Delamaro, Proteum — a Mutation Analysis Based Testing Environment, University of Sao Paulo, Sao Paulo, Brazil, 1993, Master Thesis.
[110] Parasoft, Parasoft Insure++, 2006, http://www.parasoft.com/jsp/products/home.jsp?product=Insure.
[111] A.J. Offutt, S. Lee, An empirical evaluation of weak mutation, IEEE Trans. Softw. Eng. 20 (5) (1994) 337–344.
[112] I. Moore, Jester and Pester, 2001, http://jester.sourceforge.net/.
[113] Y.-S. Ma, A.J. Offutt, Y.-R. Kwon, MuJava: An automated class mutation system, Softw. Test. Verif. Reliab. 15 (2) (2005) 97–133.
[114] SourceForge, Nester, 2002, http://nester.sourceforge.net/.
[115] C. Zhou, P. Frankl, Mutation Testing for Java Database Applications, in: Proceedings of the 2nd International Conference on Software Testing Verification and Validation, ICST'09, Davor Colorado, 2009, pp. 396–405.
[116] H. Shahriar, M. Zulkernine, MUSIC: Mutation-based SQL Injection Vulnerability Checking, in: Proceedings of the 8th International Conference on Quality Software, QSIC'08, Oxford, UK, 2008, pp. 77–86.
[117] A.J. Offutt, W.M. Craft, Using compiler optimization techniques to detect equivalent mutants, Softw. Test. Verif. Reliab. 4 (3) (1994) 131–154.
[118] R. Hierons, M. Harman, S. Danicic, Using program slicing to assist in the detection of equivalent mutants, Softw. Test. Verif. Reliab.. (1999).
[119] P. Arcaini, A. Gargantini, E. Riccobene, P. Vavassori, A novel use of equivalent mutants for static anomaly detection in software artifacts, Inf. Softw. Technol. 81 (2017) 52–64, January 2017.
[120] M Umar, An Evaluation of Mutation Operators for Equivalent Mutants, Department of Computer Science, King's College, London, 2006, M.Sc. Thesis.
[121] T. Ueshiba, H. Haga, Detecting equivalent mutants using symbolic computation, in: Proceedings of the International Conference on Electrical, Electronics, Computer Engineering and their Applications, Kuala Lumpur, Malaysia, 2014, pp. 6–11.
[122] M. Kintis, Effective Methods To Tackle the Equivalent Mutant Problem when Testing Software with Mutation, Department of Informatics, Athens University of Economics and Business, 2016, Ph.D. Thesis.
[123] K. Adamopoulos, M. Harman, R.M. Hierons, How to overcome the equivalent mutant problem and achieve tailored selective mutation using co-evolution, in: K. Deb (Ed.), Genetic and Evolutionary Computation—GECCO 2004, in: Lecture Notes in Computer Science, Vol 3103, Springer, Berlin, Heidelberg, 2004, pp. 1338–1349 GECCO 2004.
[124] M. Kintis, N. Malevris, MEDIC: A static analysis framework for equivalent mutant identification, Inf. Softw. Technol. 68 (2015) 1–17.
[125] M. Papadakis, M. Delamaro, Y. Le Traon, Mitigating the effects of equivalent mutants with mutant classification strategies, J. Sci. Comput. Program. 95 (P3) (2014) 298–319.
[126] A.J. Offutt, J. Pan, Automatically detecting equivalent mutants and infeasible paths, Softw. Test. Verif. Reliab. 7 (1997) 165–192.
[127] A.J. Offutt, J. Pan, Detecting equivalent mutants and the feasible path problem, in: Computer Assurance. 1996 COMPASS'96, Systems Integrity. Software Safety. Process Security. Proceedings of the Eleventh Annual Conference on, Gaithersburg, MD, 1996, pp. 224–236.
[128] P. Arcaini, A. Gargantini, E. Riccobene, P. Vavassori, Rehabilitating equivalent mutants as static anomaly detectors in software artifacts, in: 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops, ICSTW, Graz, 2015, pp. 1–6.
[129] W. Orzeszyna, Solutions To the Equivalent Mutants Problem: A Systematic Review and Comparative Experiment, School of Computing, Blekinge Institute of Technology, Sweden, 2011, M.Sc. Thesis.
[130] B.J.M. Grün, D. Schuler, A. Zeller, The Impact of Equivalent Mutants, in: International Conference on Software Testing, Verification, and Validation Workshops, Denver, CO, 2009, pp. 192–199.
[131] M. Papadakis, Y. Jia, M. Harman, Y. Le Traon, Trivial compiler equivalence: A large scale empirical study of a simple, fast and effective equivalent mutant detection technique, in: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Florence, 2015, pp. 936–946.
[132] J. Pan, Using Constraints To Detect Equivalent Mutants, ISSE Department George Mason University, 1994, M.Sc. Thesis..
[133] S. Nica, F. Wotawa, Using Constraints for Equivalent Mutant Detection. WS-FMDS, 2012, pp. 1–8.
[134] M. Kintis, N. Malevris, Using data flow patterns for equivalent mutant detection, 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops, Cleveland, OH, 2014, pp. 196–205.
[135] M. Patrick, M. Oriol, J.A. Clark, MESSI: Mutant evaluation by static semantic interpretation, in: Proceedings of the 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation, ICST'12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 711–719.
[136] R. Just, M.D. Ernst, G. Fraser, Using state infection conditions to detect equivalent mutants and speed up mutation analysis, in: Proceedings of the Dagstuhl Seminar 13021: Symbolic Methods in Testing. 2013.
[137] S. Nica, M. Nica, F. Wotawa, Detecting equivalent mutants by means of constraint systems, in: VALID 2011: The Third International Conference on Advances in System Testing and Validation Lifecycle, 2011, pp. 21–24.
[138] D. Baldwin, F.G. Sayward, Heuristics for Determining Equivalence of Program Mutations, Techreport, Yale University, New Haven, Connecticut, 1979, p. 276.
[139] M. Ellims, D. Ince, M. Petre, The Csaw C mutation tool: Initial results, in: Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques- MUTATION, IEEE Computer Society, Washington, DC, USA, 2007, pp. 185–192.
[140] E. Martin, T. Xie, A fault model and mutation testing of access control policies, in: Proceedings of the 16th International Conference on World Wide Web, Ser. WWW'07, ACM Press, New York, New York, USA, 2007, pp. 667–676.
[141] L. du Bousquet, M. Delaunay, Towards mutation analysis for Lustre programs, Electron. Notes Theor. Comput. Sci. 203 (4) (2008) 35–48.
[142] E.S. Mresa, L. Bottaci, Efficiency of mutation operators and selective mutation strategies: An empirical study, Softw. Test. Verif. Reliab. 9 (4) (1999) 205–232.
[143] M. Harman, R. Hierons, S. Danicic, The relationship between program dependence and mutation analysis, in: W.E. Wong (Ed.), Mutation Testing for the New Century, Kluwer Academic Publishers, Norwell, MA, USA, 2001, pp. 5–13.
[144] J. Offutt, Y.-S. Ma, Y.-R. Kwon, The class-level mutants of mujava, in: Proceedings of the 2006 International Workshop on Automation of Software Test - AST'06, Ser. AST'06, ACM Press, New York, New York, USA, 2006, pp. 78–84.
[145] G. Kaminski, P. Ammann, Using a fault hierarchy to improve the efficiency of DNF logic mutation testing, in: Proc. Int. Conf. Software Testing Verification and Validation ICST'09, 2009, pp. 386–395.
[146] C. Ji, Z. Chen, B. Xu, Z. Wang, A new mutation analysis method for testing Java exception handling, in: Proc. 33rd Annual IEEE Int. Computer Software and Applications Conf. COMPSAC'09, vol. 2, 2009, pp. 556–561.
[147] A.M.R. Vincenzi, E.Y. Nakagawa, J.C. Maldonado, M.E. Delamaro, R. A.F. Romero, Bayesian-learning based guidelines to determine equivalent mutants, Int. J. Softw. Eng. Knowl. Eng. 12 (6) (2002) 675–690.
[148] D. Schuler, V. Dallmeier, A. Zeller, Efficient mutation testing by checking invariant violations, in: ISSTA'09: Proceedings of the Eighteenth International Symposium on Software Testing and Analysis, Ser. ISSTA'09, ACM Press, New York, New York, USA, 2009, pp. 69–80.

[149] D. Schuler, A. Zeller, Covering and uncovering equivalent mutants, Softw. Test. Verif. Reliab. 23 (2013) 353–374.
[150] M. Kintis, M. Papadakis, N. Malevris, Employing second-order mutation for isolating first-order equivalent mutants, Softw. Test. Verif. Reliab. 25 (5–7) (2015) 508–535, STVR, Special Issue on Mutation Testing.
[151] M. Kintis, N. Malevris, Identifying more equivalent mutants via code similarity, in: Proceedings of the 20th Asia-Pacific Software Engineering Conference, APSEC, 1, 2013, pp. 180–188, December 2013.
[152] R. Just, D. Jalali, L. Inozemtseva, M.D. Ernst, R. Holmes, G. Fraser, Are mutants a valid substitute for real faults in software testing? in: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2014, ACM, New York, NY, USA, 2014, pp. 654–665.
[153] J.H. Andrews, L.C. Briand, Y. Labiche, Is mutation an appropriate tool for testing experiments? in: Proceedings of the 27th International Conference on Software Engineering, ICSE'05, ACM, New York, NY, USA, 2005, pp. 402–411.
[154] R. Gopinath, C. Jensen, A. Groce, Mutations: How close are they to real faults? in: Proceedings of the 2014 IEEE 25th International Symposium on Software Reliability Engineering, ISSRE'14, IEEE Computer Society, Washington, DC, USA, 2014, pp. 189–200.
[155] M. Daran, P. Thévenod-Fosse, Software error analysis: A real case study involving real faults and mutations, in: Steve J. Zeil, Will Tracz (Eds.), Proceedings of the 1996 ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA'96, ACM, New York, NY, USA, 1996, pp. 158–171.
[156] A.S. Namin, S. Kakarla, The use of mutation in testing experiments and its sensitivity to external threats, in: Proceedings of the 2011 International Symposium on Software Testing and Analysis, ISSTA'11, ACM, New York, NY, USA, 2011, pp. 342–352.
[157] B. Kitchenham, R. Pretorius, D. Budgen, O. Pearl Brereton, M. Turner, M. Niazi, S. Linkman, Systematic literature reviews in software engineering: A tertiary study, Inf. Softw. Technol. 52 (2010) 792–805.
[158] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, S. Linkman, Systematic literature reviews in software engineering - a systematic literature review, Inf. Softw. Technol. 51 (1) (2008) 7–15.