Protocol for a Systematic Literature Review of Methods Dealing with Equivalent Mutant Problem

Wojciech Orzeszyna[a,b], Lech Madeyski[*,a], Richard Torkar[b,c]

[a] *Wrocław University of Technology, Wyb. Wyspiańskiego 27, 50370 Wrocław, Poland*
[b] *Blekinge Institute of Technology, S-371 79 Karlskrona, Sweden*
[c] *Certus Software V&V Center, Simula Research Laboratory, 1325 Lysaker, Norway*

1. Introduction

Testing is the key method to ensure quality of software. But how does one find out if the test suite is sufficiently covering all quality aspects? There are some established solutions for evaluating if the number of test cases is adequate, but there are still fewer ways to evaluate the quality of tests; mutation testing can be seen as one.

Mutation testing seeds artificial faults into an application (mutants) and checks whether a test suite can detect these faults. If these faults are not found, the test suite is considered as 'not good enough' [1]. There are also mutations, which keep the program semantics unchanged and thus cannot be detected by any test suite. Finding a way to select, or not select, these mutations is also known as the *equivalent mutant problem*.

The equivalent mutant problem has been increasingly studied since mutation testing was first proposed in the 1971 by Richard Lipton a student paper [2]. The growth of this field can also be dated in the late 1970s, when articles by DeMillo et al. [1], Hamlet [3] and Budd et al. [4] were published.

1.1. Objectives of the SLR

The overall aim of the study is to develop a new, more effective method for overcoming equivalent mutant problem or to enhance existing methods. To do that, a systematic literature review in the field of equivalent mutants problem is needed, to get to know the current state of knowledge and to have a good starting point. The following objectives are defined to meet the aim:

- Identify existing methods for dealing with equivalent mutants.
- Identify current state of development of existing methods for dealing with equivalent mutant.
- Classify those methods.
- Rank existing methods according to the number of detected equivalent mutations (percentage).
- Analyze possibilities to improve existing methods.

*Corresponding author: Lech Madeyski http://madeyski.e-informatyka.pl/

1.2. Research Questions

Research questions must determine the goal of the literature review and help to provide expected results [5–9]. The main objective of study is to develop a method which will detect all of equivalent mutants significantly faster than now, what means that the time for executing mutation testing will be measured in minutes instead of hours or even days for large programmes. Therefore in general there are two main factors [10, 11] - number of detected equivalent mutations (percentage) and duration of the detecting process (mutants per second). In relation to the mentioned factors, few questions were created for the literature review. For each of them a short description of expected outcomes is provided in the next paragraphs.

- **RQ1: What methods exist that try to solve the problem of equivalent mutants?**
  This is a very general question. In this case general ideas are also expected. Some of them might have been implemented and evaluated while some might be theoretical suggestions for further refinements.
- **RQ2: How can those methods be classified?**
  As a result, the classification of existing methods to some general domains and areas is expected.
- **RQ3: What is the maturity of existing methods?**
  All existing methods will be grouped by their maturity.
- **RQ4: What are the theoretical ideas on how to improve already empirically evaluated techniques?**
  In this case, everything that the authors mention in e.g. "Future work" is to be analysed. Any possibilities that would lead to an increase in the number of detected equivalent mutants are welcome.

2. Search Strategy

During the initial examination of the domain it was discovered that very little literature is likely to exist. Due to this reason the search process was made in two iterations. Primary search was automated, using search engines and digital libraries. Detailed description of the resources is provided in further paragraphs. After this process the manual search was conducted to scan the gray literature [7]. Second iteration includes checking reference lists from relevant primary studies, conference proceedings, work at progress and contacting all the authors asking them if they know of any unpublished results [6].

2.1. Search terms construction process

Following steps can be distinguished:

1. Develop search terms from the research questions. All possible terms which relate to the research questions were listed.
2. Synonyms of already collected terms were added to the list of search terms.

3. New search terms were collected by changing the plurals to singular forms and singular to plural forms.
4. New search terms were collected by gathering keywords from abstracts and conclusions from a sample of relevant research papers.
5. New search terms were collected by browsing through grey literature (technical reports, non peer reviewed articles, websites, etc.)
6. New search terms were constructed by using Boolean OR with synonyms of search keywords.
7. New search terms were constructed by using Boolean AND for combining different search terms.

2.2. Identifying search terms

For each research question related major terms were developed. The main set of terms is the same for all of the questions, so to minimizing duplicates if the term was listed for one research question, it will not be for next ones.

- RQ1: equivalent mutants, detection, methods, techniques, problem, mutation testing, mutation analysis, equivalence
- RQ2: classification, ranking
- RQ3: empirical evaluation, implemented, development
- RQ4: further improvement, improve

2.3. Finding synonyms, alternative spellings and forms

* - means zero or more letters

- mutation testing, mutation analysis
- equivalen* mutant*
- detect*, find*, recognize*, catch*
- method*, technique*
- problem*, issue*, question*
- classification*, ranking*, classified, categorisation*, categorization*, systematisation, type*, kind*
- empirical*, evaluat*, implement*, development, developed
- further, next, future, new
- improv*, progress*, enhanc*, refin*, increas*

2.4. Generic search terms

The title, abstract and keywords of the articles in the included electronic databases and conference proceedings will be searched according to the following search terms:

1. equivalen* AND mutant* AND (mutation OR testing OR analysis)
2. equivalen* AND mutant* AND (detect* OR find* OR recognize* OR catch*) AND (method* OR technique*)
3. equivalen* AND mutant* AND (problem* OR issue* OR question*)
4. equivalen* AND mutant* AND (method* OR technique*) AND (classification* OR ranking* OR classified OR categorisation* OR categorization* OR systematisation OR type* OR kind*)
5. equivalen* AND mutant* AND (method* OR technique*) AND (empirical* OR evaluat* OR implement* OR development OR developed)
6. equivalen* AND mutant* AND (method* OR technique*) AND (further OR next OR future OR new)
7. equivalen* AND mutant* AND (method* OR technique*) AND (improv* OR progress* OR enhanc* OR refin* OR increas*)

The detailed forms (due to differences in search capabilities between various databases) are presented in the Appendix A.

2.5. Resources to be searched

2.5.1. Automatic Search

The main resources to be searched in the first iteration (automated search) are electronic databases and conference proceedings. Access to them is provided directly from the web page or using JabRef [12].

- ACM Digital Library
- IEEE Xplore
- Science Direct
- Springer Link
- Wiley Online Library

These databases were selected, because they have been used as sources for other reviews in this area [2]. Also, we had a number of "key papers" [13–19] and we investigated if we could find all of them in the above databases, to check if we have a good set of data sources.

2.5.2. Grey Literature

To cover the most important part of grey literature some alternative sources were scanned. The manual search includes:

- Google scholar
  We used three search terms, for the first phase, and for all of them checked the first 200 results. The search terms were modified slightly in order to adopt them to Google scholar and to improve the effectiveness of the search process. The used search terms were as follows:
  - equivalen* AND mutant* AND (mutation OR testing OR analysis)
  - equivalen* AND mutant* AND (method* OR technique*)
  - equivalen* AND mutant* AND (problem* OR issue* OR question*)
- All the proceedings from *"Mutation: The International Workshop on Mutation Analysis"* (five editions: 2000–2010).
- Scanning lists of references in all primary studies (according to the snowball sampling method [20]).
- Checking personal websites of all authors of primary studies, in search of the other relevant sources (e.g. unpublished or latest results).
- Contacting all authors of primary studies. We are going to create a list of all the authors of relevant sources and contact them to see if they have anything to add. The authors will be contacted in order to make sure that all relevant material had been found by our search.

2.6. Search Procedure for Automatic Search

To avoid multiple unnecessary searching operations of same issue there should be search procedure.

1. Choose search term not already being searched,
2. For each database mentioned above type specific search term into the search engine,
3. If there is possibility choose publication category "Software engineering", "Computer Science" or similar,
4. Set the language of expected results to English, if search engine allows to do so,
5. Run search,
6. Download BibTeX data with abstract for all papers,

2.7. Documenting Search Results

Because of very little literature existing in this domain there is no need to create complicated directory tree for the results. It is better to keep the original names of downloaded files in case of need to re-search them.

For each found paper create table:.

Number	
Filename	
URL	
Title	
Author(s)	
Relevant to RQ1	
Relevant to RQ2	
Relevant to RQ3	
Relevant to RQ4	
Included / Excluded	

2.8. Results Selection Process

This section describes the selection criteria and the selection process used to choose only those publications in the search results that are relevant to this systematic review.

2.8.1. Inclusion Criteria

The following inclusion criteria were taken into account when selecting the primary studies:

- Describes at least one method for detecting, suggesting or avoiding equivalent mutants (this could include proof of concepts and empirically evaluated solutions, as well as theoretical ideas).
- Discusses any classification of the aforementioned methods.
- Evaluates, analyses or compares the aforementioned methods.
- Determines current state of maturity of the methods dealing with EMP (theoretical ideas/proofs of concept/empirically evaluated solutions).
- Proposes any theoretical ideas on how to improve the already evaluated methods dealing with EMP.
- Refers to other primary studies.

2.8.2. Exclusion Criteria

The following type of studies were excluded:

- Article's language is other than English.
- Article cannot be found in full-text.
- Article concerns mutations in other fields of study than software engineering or computer science.

3. Results Selection Process

This section describes the selection criteria and the selection process used to choose only those publications in the search results that are relevant to this systematic review.

3.1. Primary Study Selection Process

1. The title, abstract and conclusions (due to RQ4) of each article from each digital library will be reviewed against the inclusion and exclusion criteria and any papers that are clearly irrelevant will be excluded.
2. If paper meets any exclusion criteria then it will be marked as excluded.
3. Paper will be marked as included only when it touches the subject described by research questions and none of exclusion criteria are met.
4. Only articles marked as included should be considered for next phase – Quality Assessment.
5. All of included papers will be described in BibTeX file (specified in "Documenting selected papers").

3.2. Documenting Selected Papers

Each selected paper should be described in BibTeX file format as in the following example:

```
@inproceedings{gruen-mutation-2009,
    title = "The Impact of Equivalent Mutants",
    author = "Bernhard J.M. Gruen and David Schuler and Andreas
    Zeller",
    filename = "gruen-mutation-2009.pdf",
    year = "2009",
    month = "April",
    location = "Denver, Colorado, USA",
    url = "http://www.st.cs.uni-saarland.de/publications/files/
    gruen-mutation-2009.pdf",
    included = "yes"
}
```

4. Quality Assessment

In addition to general inclusion and exclusion criteria, it is important to assess the quality of primary studies [6]. Study quality assessment is adopted in order to determine the strength of the evidence and to assign grades to the recommendations generated by the systematic review [21]. The questionnaire used in this study was based on recommendations of [7, 21] with some specific questions according to the research question and the type of study. Quality assessment questionnaire is shown in Table 1.

All criteria will be summed and given in percentage evaluation system according to the equation:

$$Quality\ Note = \frac{Sum\ of\ points}{Points\ possible\ to\ get} \cdot 100\% \qquad (1)$$

Table 1: Quality assessment questionnaire

Property	**Points and Estimation Notes**
1. Topic focus	
1.1 To what degree does the paper topic covers research question issue?	scale: 0, 1 or 2
1.2 Paper body fully meets issues provided in its abstract (no unnecessary divagations)	0 - No; 2 - Yes
2. Analysis and Conclusions	
2.1 Could you replace study?	0 - No; 1 - Partly; 2 - Yes
2.2 Described methods were empirically evaluated	0 - No; 2 - Yes
2.3 What kind of projects were used for empirical evaluation?	0 - None; 0.5 - Own; 1 - Student's; 2 - Open source or commercial use
2.4 Are all study questions answered?	0 - No; 2 - Yes
2.5 Are the obtained numbers of detected equivalent mutants process greater than in previous reports? (only if described method was empirically evaluated and the results were given)	0 - No; 2 - Yes
2.6 Was there a control group with which to compare treatments?	0 - No; 2 - Yes
2.7 Are there any ideas for futher investigation presented?	0 - No; 2 - Yes
3. References	
3.1 Is the paper well referenced?	0 - No references; 1 - Document references unreliable sources or sources from one institution/author only, or number of sources is poor (less than 15); 2 - More sources from various authors and institutions
3.2 Paper contains links to previously selected resources	0 - No; 2 - Yes

Because of very little literature existing in this domain there is no need to design complicated data extraction and data synthesis processes. Collected quality notes should be used to assist primary study selection [5]. All of the included articles should be read precisely for the final phase – writing up the results of the review and finding the coming out well further investigations for the master thesis.

5. Reporting the review

5.1. Data extraction form

The data extraction forms were designed to collect all information necessary to address the issues of review and the study quality assessment. The data extracted includes the information needed to answer the research question and the criteria for assessing study quality. For each found paper the form as presented in Table 2 should be filled in.

Table 2: Data extraction form

Method name	
Described in article	
Summary	
Percentage of equivalent mutants detected	
Current state of development (theoretical idea, implemented project, empirically evaluated project)	
Method implemented for language	
Ideas on how to improve the method	
Quality Assessment result	

6. Data extraction results

6.0.1. *How to Overcome the Equivalent Mutant Problem and Achieve Tailored Selective Mutation Using Co-evolution*

Method name	Tailored Selective Mutation Using Co-evolution
Described in article	Adamopoulos2004 [22]
Summary	The authors proposed a method basing on genetic algorithms. They showed how to design the fitness function which will allow to avoid generation of equivalent mutants. "If S_i is the score of a mutant i and the individual consists of L mutants the fitness Mf of this individual is given by: $$Mf = \begin{cases} \frac{\sum_{i=1}^{L} S_i}{L} & \text{if } \forall i . S_i \neq 1. \\ 0 & \text{otherwise.} \end{cases}$$ If there exists i sucha that $S_i = 1$, then there is a mutant killed by no test cases"[22]. The proposed fitness function requires that for every generated mutant exists at least one test case which can kill it. For mutants which cannot be killed by any test case, the value of fitness is very low. This guarantees not to generate equivalent mutants.
Percentage of equivalent mutants detected	100% (equivalent mutants are not generated)
Current state of development (theoretical idea, implemented project, empirically evaluated project)	Theoretical idea. The real mutation testing tool is currently under development.
Method implemented for language	Not given
Ideas on how to improve the method	Not given
Quality Assessment result	72%

6.0.2. Heuristics for Determining Equivalence of Program Mutations

Method name	Using Compiler Optimization Techniques to Detect Equivalent Mutants
Described in article	Baldwin1979 [13]
Summary	The authors proposed an approach that uses compiler optimization techniques to detect equivalent mutants. The approach is based on the idea that the optimization procedure of source code will produce an equivalent application, so an equivalent mutant should be detected by optimization or reverse ("de-optimization") process. They proposed six types of heuristics: Constant Propagation, Invariant Propagation, Common Subexpression Elimination, Recognition of Loop Invariants, Hoisting and Sinking, Dead Code Detection.
Percentage of equivalent mutants detected	Not given
Current state of development (theoretical idea, implemented project, empirically evaluated project)	Theoretical idea
Method implemented for language	Fortran
Ideas on how to improve the method	Not given
Quality Assessment result	72%

6.0.3. Towards Mutation Analysis for Lustre Programs

Method name	Lesar model-checker used for eliminating equivalent mutant
Described in article	Bousquet2008 [23]
Summary	It is possible to construct proofs about the programs, since Lustre is based on mathematical foundation. The authors used LESAR [24, 25], a model-checker for Lustre which can be used prove the correctness of an application or to compare two programs. It needs "a verification program that is a comparison of the mutant and the original programs. When some environment description is provided with the original program, it is possible to consider the mutant-equivalency"[23].
Percentage of equivalent mutants detected	Not given
Current state of development (theoretical idea, implemented project, empirically evaluated project)	Implemented tool (Alien-V). Empirically evaluated only with 8 very small programs.
Method implemented for language	Lustre
Ideas on how to improve the method	Solve problems for some programs dealing with integers.
Quality Assessment result	95%

6.0.4. The Csaw C Mutation Tool: Initial Results

Method name	Using semantic differences in terms of running profile to detect non-equivalent mutants
Described in article	Ellims2007 [26]
Summary	This paper describes initial results from the research on mutation tool for C language. Suggestions for futer research are the most valuable part of this work from the perspective of equivalent mutants problem: "Firstly to look at possibilities for altering programs to prevent difficult-to-kill mutations being generated. Secondly to look at other external visible effects of mutants such as CPU usage, memory usage etc. as a means of detecting non-equivalent mutants"[26].
Percentage of equivalent mutants detected	Not given
Current state of development (theoretical idea, implemented project, empirically evaluated project)	Theoretical idea
Method implemented for language	Not Given
Ideas on how to improve the method	Not Given
Quality Assessment result	75%

6.0.5. The Impact of Equivalent Mutants on Coverage

Method name	The Impact of Equivalent Mutants on Coverage
Described in article	Gr'un2009 [14]
Summary	The authors proposed an approach which "measures changes in program behavior between the mutant and the original version. One aspect that is particularly easy to measure is control flow: If a mutation alters the control flow of the execution, different statements would be executed in a different order - an impact that is easy to detect using standard coverage measurement techniques. (...) By comparing the coverage of the original execution with the coverage of the mutated execution, we can determine the coverage difference. (...) This measure is motivated by the hypothesis that a mutation that has non-local impact on the coverage is more likely to change the observable behavior of the program. (...) if a mutation had impact on code coverage, it was more likely to be non-equivalent; if it did not have impact on code coverage, it was more likely to be equivalent."[14].
Percentage of equivalent mutants detected	- (does not detect equivalent mutants - only suggest them to the user)
Current state of development (theoretical idea, implemented project, empirically evaluated project)	Implemented in a tool (JAVALANCHE [27]), empirical evaluation only on one bigger project (JAXEN).
Method implemented for language	Java
Ideas on how to improve the method	Consider alternative impact measures
Quality Assessment result	100%

6.0.6. The Relationship Between Program Dependence and Mutation Analysis

Method name	Avoiding equivalent mutants generation using program dependence analysis
Described in article	Harman2001 [28]
Summary	There are three approaches to the way in which mutant can be inspected: strong (output-based), weak (state-based) and firm (compares programs in the probe point). An approach proposed in this paper uses firm mutation testing [29]. The authors assume that "mutants which fail to propagate 'corrupted data' to the inspection set at the probe point will be equivalent and should be avoided"[28].
Percentage of equivalent mutants detected	Not given
Current state of development (theoretical idea, implemented project, empirically evaluated project)	Theoretical idea
Method implemented for language	Not given
Ideas on how to improve the method	To use this technique in tandem with constraint-based techniques.
Quality Assessment result	89%

6.0.7. Using Program Slicing to Assist in the Detection of Equivalent Mutants

Method name	Using Program Slicing to Assist in the Detection of Equivalent Mutants
Described in article	Hierons1999 [15]
Summary	In this article a new technique has been proposed. “Instead of attempting to answer the question of equivalence, the approach presented here uses a program simplification process (program slicing), attempting to create the simplest program which denotes the question, this approach allows automation to be exploited in partially answering the question. The simplified program is an approximate answer; the greater the level of simplification, the closer the approximation. While this does not answer the question for the human analyst (...), it can reduce the effort involved in detecting equivalence. Where the mutant is not equivalent, it can help the tester find input that kills the mutant”[15].
Percentage of equivalent mutants detected	Not given, but the authors say that this method and constraint solving approach [17, 18] “are of equal power in detecting equivalent mutants. The difference between the two approaches lies in the way each handles cases where it is not possible to decide whether the mutant is equivalent, where amorphous slicing may offer additional assistance over that available through the constraint based approach”[15].
Current state of development (theoretical idea, implemented project, empirically evaluated project)	Theoretical idea
Method implemented for language	Not given
Ideas on how to improve the method	Using constraint solving approach and slicing together.
Quality Assessment result	89%

6.0.8. Java Exception Mutation

Method name	Java Exception Mutation
Described in article	Ji2009 [30]
Summary	The authors proposed a set of mutation operators only for Java exceptions. Because of that they "have provided a methodology for CBR (*Catch Block Replacement*) and CBI (*Catch Block Insertion*) to distinguish the equivalent mutant generated through semantic exception hierarchy"[30].
Percentage of equivalent mutants detected	100%
Current state of development (theoretical idea, implemented project, empirically evaluated project)	Theoretical idea
Method implemented for language	Java
Ideas on how to improve the method	Not given
Quality Assessment result	78%

6.0.9. Higher Order Mutation Testing

Method name	Higher Order Mutation Testing
Described in article	Jia2009 [31]
Summary	"This paper introduces a new paradigm for Mutation Testing, which is called Higher Order Mutation Testing (HOM Testing). Traditional Mutation Testing considers only first order mutants, created by the injection of a single fault. Often these first order mutants denote trivial faults that are easily killed. Higher order mutants are created by the insertion of two or more faults"[31]. The authors belive that High Order Mutation Testing has three major benefits: Increased subtlety, reduce test effort and, what is the most important from the perspective of equivalent mutants problem, reduce the number of generated equivalent mutants.
Percentage of equivalent mutants detected	Not given
Current state of development (theoretical idea, implemented project, empirically evaluated project)	Implemented (proof-of-concept)
Method implemented for language	C
Ideas on how to improve the method	Consider weak and firm High Order Mutation Testing.
Quality Assessment result	83%

6.0.10. An Analysis and Survey of the Development of Mutation Testing

Method name	-
Described in article	Jia2010 [2]
Summary	This technical report presents current state-of-art in the field of Mutation Testing. One of the sections provides a comprehensive summary of equivalent mutants detection techniques.
Percentage of equivalent mutants detected	-
Current state of development (theoretical idea, implemented project, empirically evaluated project)	-
Method implemented for language	-
Ideas on how to improve the method	-
Quality Assessment result	67%

6.0.11. Using a Fault Hierarchy to Improve the Efficiency of DNF Logic Mutation Testing

Method name	Using a Fault Hierarchy to Improve the Efficiency of DNF Logic Mutation Testing
Described in article	Kaminski2009 [32]
Summary	Using weak mutation testing (state-based inspection) and a set of logic mutation operators only allowed the authors to introduce a fault class hierarchy. In this hierarchy they have selected operators which do not create equivalent mutants.
Percentage of equivalent mutants detected	Not given (this technique does not generate equivalent mutants)
Current state of development (theoretical idea, implemented project, empirically evaluated project)	Empirically evaluated project (only on one project - TCAS)
Method implemented for language	Java
Ideas on how to improve the method	Not given
Quality Assessment result	85%

6.0.12. Evaluating Mutation Testing Alternatives: A Collateral Experiment

Method name	-
Described in article	Kintis2010 [33]
Summary	"In this paper several second order mutation testing strategies are introduced, assessed and compared along with weak mutation against strong. (...) The experimental assessment of weak mutation suggests that it reduces significantly the number of the produced equivalent mutants on the one hand and that the test criterion it provides is not as weak as is thought to be on the other"[33].
Percentage of equivalent mutants detected	Instead of detecting equivalent mutants, Higher Order Mutation Testing aims to reduce the number of generated equivalent mutants. This reduction varies from 65,5% for $HDom(50\%)$ to 86,8% for $SDomF$ strategy (both belongs to Hybrid Strategies) with the loss of test effectiveness from only 1,75% for $HDom(50\%)$ to 4,2% for $SDomF$.
Current state of development (theoretical idea, implemented project, empirically evaluated project)	Empirically evaluated project on fifteen small programs (from 11 to 47 lines of code)
Method implemented for language	Java
Ideas on how to improve the method	Not given
Quality Assessment result	91%

6.0.13. A Fault Model and Mutation Testing of Access Control Policies

Method name	Margrave's change-impact analysis
Described in article	Martin2007 [34]
Summary	In this paper Margrave [35], a change-impact analysis tool was used to detect equivalent mutants among generated mutants. The authors originally believed "equivalent mutant detection to be an important efficiency improvement though they found in practice that evaluating requests and comparing responses to be computationally cheaper than performing change-impact analysis with Margrave. Furthermore, limitations of Margrave prevented the detection of equivalent mutants for mutation operators on conditions and some combining algorithms"[34].
Percentage of equivalent mutants detected	Not given
Current state of development (theoretical idea, implemented project, empirically evaluated project)	Implemented project (only a proof-of-concept)
Method implemented for language	XACML policies
Ideas on how to improve the method	Not given
Quality Assessment result	78%

6.0.14. Efficiency of mutation operators and selective mutation strategies: An empirical study

Method name	Selective Mutation
Described in article	Mresa1999 [36]
Summary	The authors proposed a different type of selective mutation (reduction in the number of mutants by reducing the number of applied mutation operators). "Instead of trying to achieve a small loss of test effectiveness, they also took the cost of detecting equivalent mutants into consideration. In their work, each mutation operator is assigned a score which is computed by its value and cost. Their results indicated that it was possible to reduce the number of equivalent mutants while maintaining effectiveness"[2].
Percentage of equivalent mutants detected	Not given.
Current state of development (theoretical idea, implemented project, empirically evaluated project)	Empirically evaluated project (on 11 small programs)
Method implemented for language	Fortran 77
Ideas on how to improve the method	Not given
Quality Assessment result	85%

6.0.15. Investigations of the Software Testing Coupling Effect

Method name	Higher Order Mutation Testing
Described in article	Offutt1992 [37]
Summary	This is the first paper where higher order mutation testing was proposed. The author consider impact of coupling effect to support mutation testing. "The coupling effect hypothesizes that test data sets that detect simple types of faults are sensitive enough to detect more complex types of faults. (...) The major conclusion from this investigation is the fact that by explicitly testing for simple faults, we are also implicitly testing for more complicated faults, giving us confidence that fault-based testing is an effective way to test software"[37].
Percentage of equivalent mutants detected	Not Given, but from the generated 2-order mutants only from 0,53% to 1,4% were equivalent. Comparing to 1-order mutants it is significantly better result.
Current state of development (theoretical idea, implemented project, empirically evaluated project)	Empirically evaluated project (3 small programs - from 16 to 28 lines of code)
Method implemented for language	Fortran 77
Ideas on how to improve the method	Not Given
Quality Assessment result	82%

6.0.16. Using Compiler Optimization Techniques to Detect Equivalent Mutants

Method name	Complier optimizations to detect equivalent mutants
Described in article	Offutt1994 [16]
Summary	The authors proposed algorithms for determining classes of equivalent mutants. "These algorithms are based on data flow analysis and six compiler optimization techniques"[16]: Dead Code Detection, Constant Propagation, Invariant Propagation, Common Subexpression Detection, Loop Invariant Detection and Hoisting and Sinking. "The key intuition behind this approach is that many equivalent mutants are, in some sense, either optimizations or de-optimizations of the original program. The transformations produced from code optimizers result in equivalent programs. When an equivalent mutant satisfies a code optimization rule, algorithms can detect that the mutant is in fact equivalent"[16].
Percentage of equivalent mutants detected	About 10%, with 25% standard deviation
Current state of development (theoretical idea, implemented project, empirically evaluated project)	Empirically evaluated project (15 small programs - from about 5 to 52 executable statements)
Method implemented for language	Fortran 77
Ideas on how to improve the method	Treating elements of an array as individual data items, further analysis of loops and using program slicing.
Quality Assessment result	100%

6.0.17. Detecting Equivalent Mutants and the Feasible Path Problem

Method name	Detecting Equivalent Mutants and the Feasible Path Problem
Described in article	Offutt1996 [17], Offutt1997 [18]
Summary	"Constraint-based testing (CBT) [38] uses constraints for automatic test data generation. In CBT, a constraint represents the conditions under which a mutant will die. The technique in this paper uses the fact that if a test case kills the mutant, the constraint system will be true. If the constraint system cannot be true, then there is no test case that can kill the mutant and the mutant is equivalent. The general approach to using constraints to detect equivalent mutants is to look for infeasibility in constraint systems"[17, 18].
Percentage of equivalent mutants detected	47,63%
Current state of development (theoretical idea, implemented project, empirically evaluated project)	Implemented (proof-of-concept), empirically evaluated on 11 small programs (from 11 to 30 executable statements)
Method implemented for language	Fortran 77
Ideas on how to improve the method	Recognising infeasible constraints, having better constraints and/or analysing the execution after the mutated statement
Quality Assessment result	100%

6.0.18. The Class-Level Mutants of MuJava

Method name	Using Equivalency Conditions to Eliminate Equivalent Mutants for Object Oriented Mutation Operators
Described in article	Offutt2006 [39]
Summary	"This paper introduces specific techniques for Java class mutation operators that are adapted from constraint solving approaches. Instead of running in a 'post-processing mode', after mutants are generated,(...) MuJava integrates the equivalent mutation analysis with mutant generation, as suggested by Hierons, Harman and Danicic [15]. MuJava implements specific, focused, heuristics that avoid equivalent mutants for specific mutation operators. This approach is based on equivalency conditions for mutation operators, which in turn is based on the conditions under which mutants are killed"[39]. The authors defined equivalency conditions for sixteen mutation class-level operators.
Percentage of equivalent mutants detected	Not given
Current state of development (theoretical idea, implemented project, empirically evaluated project)	Implemented (as a part of MuJava [10]), applied on 866 classes from six applications.
Method implemented for language	Java
Ideas on how to improve the method	Identifying equivalency conditions for other class-level mutation operators.
Quality Assessment result	95%

6.0.19. An Empirical Evaluation of the First and Second Order Mutation Testing Strategies

Method name	-
Described in article	Papadakis2010 [40]
Summary	"This paper presents an empirical study for using mutation testing and its first and second order mutation variants. (...) The results obtained indicate that first order strategies are generally more effective at detecting faults, than their second order rivals however, at a greater cost. Second order strategies can drastically decrease the number of equivalent mutants introduced and provide significant savings to both numbers of produced mutants and required test cases. The results suggest that a reduction of approximately 80% to 90% of the equivalent mutants generated by second order strategies can be tackled. Moreover, second order strategies can accomplish reductions of roughly 30% of the required test cases with approximately 10% or less on the loss of their fault detection ability compared to strong mutation. Randomly selecting a percentage of first order mutants results in a fault loss ranging from 26% to 6% for the methods Rand 10% to 60%. Their test reductions range from 60% to 17%"[40].
Percentage of equivalent mutants detected	Reduction of approx. 80% to 90% of the equivalent mutants generated by second order strategies.
Current state of development (theoretical idea, implemented project, empirically evaluated project)	Empirical evaluation on eight medium programs (from 137 to 513 lines of code).
Method implemented for language	C
Ideas on how to improve the method	Not given
Quality Assessment result	91%

6.0.20. Efficient Mutation Testing by Checking Invariant Violations

Method name	Impact of Dynamic Invariants
Described in article	Schuler2009 [41]
Summary	Instead of detecting equivalent mutants directly, the authors proposed a technique which suggest the tester mutants which are probably non-equivalent. They "asses the impact of mutations by checking dynamic invariants. (...) For each learned invariant, they insert statements into the bytecode that check for invariant violations before and after a method. If an invariant is violated, this is reported and the run resumes. (...) If a mutant violates even very simple invariants, it is more likely to be detectable by an actual test. When improving test suites, test managers therefore should focus on those surviving mutations that have the greatest impact on invariants"[41].
Percentage of equivalent mutants detected	Not given
Current state of development (theoretical idea, implemented project, empirically evaluated project)	Empirically evaluated project (on six programs). Implemented in JAXEN [27].
Method implemented for language	Java
Ideas on how to improve the method	Alternative impact measures, consider impact as similarity measure.
Quality Assessment result	100%

6.0.21. (Un-)Covering Equivalent Mutants

Method name	Mutation Impact
Described in article	Schuler2010 [19]
Summary	"Equivalent mutants are defined as having no observable impact on the programs output. This impact of a mutation can be assessed by checking the program state at the end of a computation, as tests do. However, we can also assess the impact of a mutation while the computation is being performed. In particular, we can measure changes in program behavior between the mutant and the original version. The idea is that if a mutant impacts internal program behavior, it is also more likely to change external program behavior - and thus impacts the semantics of the program. If we focus on mutations with impact, we would thus expect to find fewer equivalent mutants"[19].
Percentage of equivalent mutants detected	Not given. This approach only suggest (non-)equivalent mutants. "If the mutation changes coverage, it has a 75% chance to be non-equivalent"[19].
Current state of development (theoretical idea, implemented project, empirically evaluated project)	Empirically evaluated project (on seven programs, from 5,000 to 100,000 lines of code). Implemented in JAXEN [27].
Method implemented for language	Java
Ideas on how to improve the method	Not given
Quality Assessment result	91%

6.0.22. Bayesian-Learning Based Guidelines to Determine Equivalent Mutants

Method name	Bayesian-Learning Based Guidelines to Determine Equivalent Mutants
Described in article	Vincenzi2002 [42]
Summary	"This paper aims at reducing the effort needed to analyze the live mutants instead of providing a way to automatic detect the equivalents. The idea presented here is to provide guidelines to ease the determination of equivalent mutants and also the identification of non-equivalents, which is useful to improve the test set. Based on historical data (...) the approach, named Bayesian Learning-Based Equivalent Detection Technique (BaLBEDeT), uses the Brute-Force algorithm to estimate which is the most promising group of mutants that should be analyzed"[42].
Percentage of equivalent mutants detected	Not given
Current state of development (theoretical idea, implemented project, empirically evaluated project)	Empirically evaluated project (on 5 small programs)
Method implemented for language	C
Ideas on how to improve the method	Consider the frequency of execution.
Quality Assessment result	100%

A. Generic Search Terms

A.1. ACM Digital Library and IEEE Xplore

1. equivalen* AND mutant* AND (mutation OR testing OR analysis)
2. equivalen* AND mutant* AND (detect* OR find* OR recognize* OR catch*) AND (method* OR technique*)
3. equivalen* AND mutant* AND (problem* OR issue* OR question*)
4. equivalen* AND mutant* AND (method* OR technique*) AND (classification* OR ranking* OR classified OR categorisation* OR categorization* OR systematisation OR type* OR kind*)
5. equivalen* AND mutant* AND (method* OR technique*) AND (empirical* OR evaluat* OR implement* OR development OR developed)
6. equivalen* AND mutant* AND (method* OR technique*) AND (further OR next OR future OR new)

7. equivalen* AND mutant* AND (method* OR technique*) AND (improv* OR progress* OR enhanc* OR refin* OR increas*)

A.2. Science Direct

1. TITLE-ABSTR-KEY(equivalen* AND mutant* AND (mutation OR testing OR analysis))[All Sources(Computer Science)]
2. TITLE-ABSTR-KEY(equivalen* AND mutant* AND (detect* OR find* OR recognize* OR catch*) AND (method* OR technique*))[All Sources(Computer Science)]
3. TITLE-ABSTR-KEY(equivalen* AND mutant* AND (problem* OR issue* OR question*))[All Sources(Computer Science)]
4. TITLE-ABSTR-KEY(equivalen* AND mutant* AND (method* OR technique*) AND (classification* OR ranking* OR classified OR categorisation* OR categorization* OR systematisation OR type* OR kind*))[All Sources(Computer Science)]
5. TITLE-ABSTR-KEY(equivalen* AND mutant* AND (method* OR technique*) AND (empirical* OR evaluat* OR implement* OR development OR developed))[All Sources(Computer Science)]
6. TITLE-ABSTR-KEY(equivalen* AND mutant* AND (method* OR technique*) AND (further OR next OR future OR new))[All Sources(Computer Science)]
7. TITLE-ABSTR-KEY(equivalen* AND mutant* AND (method* OR technique*) AND (improv* OR progress* OR enhanc* OR refin* OR increas*))[All Sources(Computer Science)]

A.3. Springer Link

1. ab:(equivalen* AND mutant* AND (mutation OR testing OR analysis))' with filters: Computer Science Software Engineering
2. ab:(equivalen* AND mutant* AND (detect* OR find* OR recognize* OR catch*) AND (method* OR technique*))' with filters: Computer Science Software Engineering
3. ab:(equivalen* AND mutant* AND (problem* OR issue* OR question*))' with filters: Computer Science Software Engineering
4. ab:(equivalen* AND mutant* AND (method* OR technique*) AND (classification* OR ranking* OR classified OR categorisation* OR categorization* OR systematisation OR type* OR kind*))' with filters: Computer Science Software Engineering
5. ab:(equivalen* AND mutant* AND (method* OR technique*) AND (empirical* OR evaluat* OR implement* OR development OR developed))' with filters: Computer Science Software Engineering
6. ab:(equivalen* AND mutant* AND (method* OR technique*) AND (further OR next OR future OR new))' with filters: Computer Science Software Engineering
7. ab:(equivalen* AND mutant* AND (method* OR technique*) AND (improv* OR progress* OR enhanc* OR refin* OR increas*))' with filters: Computer Science Software Engineering

A.4. Wiley Online Library

1. equivalen* AND mutant* AND (mutation OR testing OR analysis) in Abstract OR equivalen* AND mutant* AND (mutation OR testing OR analysis) in Article Titles OR equivalen* AND mutant* AND (mutation OR testing OR analysis) in Keywords AND 1099-1689 in ISSN
2. equivalen* AND mutant* AND (detect* OR find* OR recognize* OR catch*) AND (method* OR technique*) in Abstract OR equivalen* AND mutant* AND (detect* OR find* OR recognize* OR catch*) AND (method* OR technique*) in Article Titles OR equivalen* AND mutant* AND (detect* OR find* OR recognize* OR catch*) AND (method* OR technique*) in Keywords AND 1099-1689 in ISSN
3. equivalen* AND mutant* AND (problem* OR issue* OR question*) in Abstract OR equivalen* AND mutant* AND (problem* OR issue* OR question*) in Article Titles OR equivalen* AND mutant* AND (problem* OR issue* OR question*) in Keywords AND 1099-1689 in ISSN
4. equivalen* AND mutant* AND (method* OR technique*) AND (classification* OR ranking* OR classified OR categorisation* OR categorization* OR systematisation OR type* OR kind*) in Abstract OR equivalen* AND mutant* AND (method* OR technique*) AND (classification* OR ranking* OR classified OR categorisation* OR categorization* OR systematisation OR type* OR kind*) in Article Titles OR equivalen* AND mutant* AND (method* OR technique*) AND (classification* OR ranking* OR classified OR categorisation* OR categorization* OR systematisation OR type* OR kind*) in Keywords AND 1099-1689 in ISSN
5. equivalen* AND mutant* AND (method* OR technique*) AND (empirical* OR evaluat* OR implement* OR development OR developed) in Abstract OR equivalen* AND mutant* AND (method* OR technique*) AND (empirical* OR evaluat* OR implement* OR development OR developed) in Article Titles OR equivalen* AND mutant* AND (method* OR technique*) AND (empirical* OR evaluat* OR implement* OR development OR developed) in Keywords AND 1099-1689 in ISSN
6. equivalen* AND mutant* AND (method* OR technique*) AND (further OR next OR future OR new) in Abstract OR equivalen* AND mutant* AND (method* OR technique*) AND (further OR next OR future OR new) in Article Titles OR equivalen* AND mutant* AND (method* OR technique*) AND (further OR next OR future OR new) in Keywords AND 1099-1689 in ISSN
7. equivalen* AND mutant* AND (method* OR technique*) AND (improv* OR progress* OR enhanc* OR refin* OR increas*) in Abstract OR equivalen* AND mutant* AND (method* OR technique*) AND (improv* OR progress* OR enhanc* OR refin* OR increas*) in Article Titles OR equivalen* AND mutant* AND (method* OR technique*) AND (improv* OR progress* OR enhanc* OR refin* OR increas*) in Keywords AND 1099-1689 in ISSN

References

[1] R. DeMillo, R. Lipton, and F. Sayward, "Hints on test data selection: Help for the practicing programmer," *Computer*, vol. 11, pp. 34–41, Apr 1978.

[2] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Transactions on Software Engineering*, vol. 99, no. PrePrints, 2011.

[3] R. Hamlet, "Testing programs with the aid of a compiler," *Software Engineering, IEEE Transactions on*, vol. SE-3, pp. 279–290, Jul 1977.

[4] T. A. Budd, R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "The design of a prototype mutation system for program testing," in *Proceedings of the AFIPS National Computer Conference*, (Anaheim, New Jersey), pp. 623–627, Jun 1978.

[5] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. E. Emam, and J. Rosenberg, "Preliminary guidelines for empirical research in software engineering," *IEEE Transactions on Software Engineering*, vol. 28, no. 8, pp. 721–734, 2002.

[6] B. Kitchenham, "Procedures for performing systematic reviews," tech. rep., Keele University and NICTA, 2004.

[7] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Tech. Rep. EBSE 2007-001, Keele University and Durham University Joint Report, 2007.

[8] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering - a systematic literature review," *Information and Software Technology*, vol. 51, no. 1, pp. 7–15, 2008.

[9] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, "Lessons from applying the systematic literature review process within the software engineering domain," *Journal of Systems and Software*, vol. 80, no. 4, pp. 571–583, 2007.

[10] Y.-S. Ma, J. Offutt, and Y. R. Kwon, "MuJava: an automated class mutation system," *Software Testing, Verification and Reliability*, vol. 15, no. 2, pp. 97–133, 2005.

[11] J. W. Wilkerson, *Closing the defect reduction gap between software inspection and test-driven development: applying mutation analysis to iterative, test-first programming.* PhD thesis, University of Arizona, Tucson, AZ, USA, 2008. Adviser-Nunamaker, Jay F.

[12] JabRef, "JabRef - reference manager." http://jabref.sourceforge.net, Jan 2011.

[13] D. Baldwin and F. G. Sayward, "Heuristics for determining equivalence of program mutations," techreport 276, Yale University, New Haven, Connecticut, 1979.

[14] B. J. M. Grün, D. Schuler, and A. Zeller, "The impact of equivalent mutants," in *Proceedings of the IEEE International Conference on Software Testing, Verification, and Validation Workshops*, (Denver, Colorado, USA), pp. 192–199, IEEE Computer Society, 2009.

[15] R. Hierons, M. Harman, and S. Danicic, "Using program slicing to assist in the detection of equivalent mutants," *Software Testing, Verification and Reliability*, vol. 9, no. 4, pp. 233–262, 1999.

[16] A. J. Offutt and W. M. Craft, "Using compiler optimization techniques to detect equivalent mutants," *Software Testing, Verification and Reliability*, vol. 4, no. 3, pp. 131–154, 1994.

[17] A. J. Offutt and J. Pan, "Detecting equivalent mutants and the feasible path problem," in *Proc. Eleventh Annual Conf. 'Systems Integrity Computer Assurance COMPASS '96 Software Safety. Process Security'*, pp. 224–236, 1996.

[18] A. J. Offutt and J. Pan, "Automatically detecting equivalent mutants and infeasible paths," *Software Testing, Verification and Reliability*, vol. 7, no. 3, pp. 165–192, 1997.

[19] D. Schuler and A. Zeller, "(Un-)covering equivalent mutants," in *Proceedings of the 3rd International Conference on Software Testing Verification and Validation (ICST'10)*, (Paris, France), pp. 45–54, Apr 2010.

[20] L. Goodman, "Snowball sampling," *The Annals of Mathematical Statistics*, vol. 32, pp. 148–170, Mar 1961.

[21] K. S. Khan, G. T. Riet, J. Glanville, A. J. Sowden, and J. Kleijnen, "Undertaking systematic reviews of research on effectiveness CRD s guidance for those carrying out or commissioning reviews," Tech. Rep. 4, University of York, 2001.

[22] K. Adamopoulos, M. Harman, and R. M. Hierons, "How to overcome the equivalent mutant problem and achieve tailored selective mutation using co-evolution," in *In GECCO (2), volume 3103 of Lecture Notes in Computer Science*, vol. 3103 of *Lecture Notes in Computer Science*, pp. 1338–1349, Springer Berlin / Heidelberg, 2004.

[23] L. du Bousquet and M. Delaunay, "Towards mutation analysis for Lustre programs," *Electronic Notes in Theoretical Computer Science*, vol. 203, no. 4, pp. 35–48, 2008.

[24] N. Halbwachs, D. Pilaud, F. Ouabdesselam, and A.-C. Glory, "Specifying, programming and verifying real-time systems using a synchronous declarative language," in *Proceedings of the international workshop on Automatic verification methods for finite state systems*, (New York, NY, USA), pp. 213–231, Springer-Verlag New York, Inc., 1990.

[25] N. Halbwachs, F. Lagnier, and C. Ratel, "Programming and verifying real-time systems by means of the synchronous data-flow language Lustre," *IEEE Transactions on Software Engineering*, vol. 18, pp. 785–793, Sep 1992.

[26] M. Ellims, D. Ince, and M. Petre, "The Csaw C mutation tool: initial results," in *Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION*, (Washington, DC, USA), pp. 185–192, IEEE Computer Society, 2007.

[27] D. Schuler and A. Zeller, "Javalanche: efficient mutation testing for Java," in *ESEC/FSE '09: Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ESEC/FSE '09, (New York, New York, USA), p. 297, ACM Press, 2009.

[28] M. Harman, R. Hierons, and S. Danicic, "The relationship between program dependence and mutation analysis," in *Mutation testing for the new century* (W. E. Wong, ed.), pp. 5–13, Norwell, MA, USA: Kluwer Academic Publishers, 2001.

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

[30] C. Ji, Z. Chen, B. Xu, and 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, pp. 556–561, 2009.

[31] Y. Jia and M. Harman, "Higher order mutation testing," *Information and Software Technology*, vol. 51, pp. 1379–1393, Oct 2009.

[32] G. Kaminski and 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*, pp. 386–395, 2009.

[33] M. Kintis, M. Papadakis, and N. Malevris, "Evaluating mutation testing alternatives: A collateral experiment," in *Proc. 17th Asia Pacific Software Engineering Conf. (APSEC)*, pp. 300–309, 2010.

[34] E. Martin and T. Xie, "A fault model and mutation testing of access control policies," in *Proceedings of the 16th international conference on World Wide Web*, WWW '07, (New York, New York, USA), pp. 667–676, ACM Press, 2007.

[35] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz, "Verification and change-impact analysis of access-control policies," in *Proceedings of the 27th international conference on Software engineering*, ICSE '05, (New York, NY, USA), pp. 196–205, ACM, 2005.

[36] E. S. Mresa and L. Bottaci, "Efficiency of mutation operators and selective mutation strategies: an empirical study," *Software Testing, Verification and Reliability*, vol. 9, no. 4, pp. 205–232, 1999.

[37] A. J. Offutt, “Investigations of the software testing coupling effect,” *ACM Trans. Softw. Eng. Methodol.*, vol. 1, pp. 5–20, Jan 1992.

[38] R. A. DeMillo and A. J. Offutt, “Constraint-based automatic test data generation,” *IEEE Transactions on Software Engineering*, vol. 17, pp. 900–910, Sep 1991.

[39] J. Offutt, Y.-S. Ma, and Y.-R. Kwon, “The class-level mutants of MuJava,” in *Proceedings of the 2006 international workshop on Automation of software test - AST '06*, AST '06, (New York, New York, USA), pp. 78–84, ACM Press, 2006.

[40] M. Papadakis and 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*, ICSTW '10, pp. 90–99, IEEE Computer Society, 2010.

[41] D. Schuler, V. Dallmeier, and A. Zeller, “Efficient mutation testing by checking invariant violations,” in *ISSTA '09: Proceedings of the Eighteenth International Symposium on Software Testing and Analysis*, ISSTA '09, (New York, New York, USA), pp. 69–80, ACM Press, 2009.

[42] A. M. R. Vincenzi, E. Y. Nakagawa, J. C. Maldonado, M. E. Delamaro, and R. A. F. Romero, “Bayesian-learning based guidelines to determine equivalent mutants,” *International Journal of Software Engineering and Knowledge Engineering*, vol. 12, no. 6, pp. 675–690, 2002.