Model-Based Higher-Order Mutation Analysis

Fevzi Belli[1], Nevin Güler[2], Axel Hollmann[1], Gökhan Suna[3], and Esra Yildiz[3]

[1] University of Paderborn, Department of Electrical Engineering and Information Technology, Paderborn, Germany
[2] University of Mugla, Department of Statistics, Mugla, Turkey
[3] Izmir Institute of Technology, Department of Computer Engineering, Izmir, Turkey
{belli,ngueler,hollmann}@adt.upb.de,
gokhansuna44@hotmail.com, esrayildiz@gmail.com

Abstract. Mutation analysis is widely used as an implementation-oriented method for software testing and test adequacy assessment. It is based on creating different versions of the software by seeding faults into its source code and constructing test cases to reveal these changes. However, in case that source code of software is not available, mutation analysis is not applicable. In such cases, the approach introduced in this paper suggests the alternative use of a model of the software under test. The objectives of this approach are (i) introduction of a new technique for first-order and higher-order mutation analysis using two basic mutation operators on graph-based models, (ii) comparison of the fault detection ability of first-order and higher-order mutants, and (iii) validity assessment of the coupling effect.

Keywords: Software Testing, Higher-Order Mutation Analysis, Coupling Effect, Basic Mutation Operators, Event Sequence Graphs.

1 Introduction and Related Work

One of the major goals of software testing is to reveal defects/bugs by comparing the observed behavior of the software with its expected behavior. There are many approaches used in software testing. Mutation analysis (MA), originally proposed by DeMillo et al. [4] and Hamlet [7], is a method for analyzing the adequacy of a test set to reveal faults [17], [20]. It bases on the idea of seeding artificial defects (mutations) into the system under test (SUT).

MA has two main steps. In the first step, faults are injected into the software by applying mutation operators to the original software. The mutation operators cause syntactic changes and represent certain fault models. Each of the faulty versions of the software generated is called a "mutant". In the second step, a test set is executed against the mutants. A mutant is considered "killed" if the expected output predicted by the test set differs from the actual output of the mutant. Otherwise, the mutant is "live". In that case, the fault(s) could not be revealed or the mutant is equivalent to the original version. If all mutants are killed by a test set, the test set is considered to be adequate in detecting the faults that were injected into the mutants.

T.-h. Kim et al. (Eds.): ASEA 2010, CCIS 117, pp. 164–173, 2010.

MA relies on two assumptions: the *competent programmer hypothesis* and the *coupling effect* [4], [15]. The competent programmer hypothesis claims that experienced programmers tend to implement software that is almost correct. The coupling effect assumes that a test set that detects all simple faults defined by first-order mutants (FOM) is also able to detect complex faults defined by higher-order mutants (HOM) [4]. The coupling effect has been studied by many researchers. Offutt ([15], [16]) described empirical investigations about the coupling effect over a specific class of software faults and showed that the coupling effect hypothesis is valid. Wah [19] presented a theoretical study of fault coupling, based on a simple model of fault-based testing. He shows that fault coupling occurs infrequently, at least when only two faults are involved. In another study, Wah [18] dealt with the coupling effect from a theoretical standpoint and he found out that the hypothesis of a coupling effect is largely valid.

Although many studies have shown the validity of the coupling effect, Harman et al. [8] argued that HOMs are potentially better able to simulate real faults. However, higher-order mutation analysis has been believed to be expensive and impractical. Therefore, they argue that higher-order mutation analysis can be available using a process that searches fitter mutants from the space of all possible mutants. Jia et al. ([10], [11]) also investigated HOMs. They introduced the concept of a subsuming HOMs that denote subtle fault combinations. The objective of that study is to seek valuable HOMs using automated search-based optimization techniques.

MA can be very expensive since there is a great number of different mutation operators for generating mutants. Therefore, another important issue of MA is the selection of appropriate mutation operators. Several studies have been carried out to find a smaller set of mutation operators without significant loss of test effectiveness. This idea was proposed by Mathur [12]. In his study, two mutation operators, array reference for scalar variable replacement (ASR) and scalar variable replacement (SVR) that generated most of the mutants are omitted to reduce the number of generated mutants. This idea was extended by omitting four mutation operators (4-selective mutation) and omitting six mutation operators (6-selective mutation). Offutt et al. [14] have shown that selective mutation is an effective alternative to non-selective mutation by presenting empirical results that support the hypothesis. Mresa and Bottaci [13] proposed a different selective mutation based on assigning a score to each mutation operator.

In this paper we suggest reducing the broad variety of mutation operators known from literature to two basic operators and their combination. MA is usually applied to the source code of SUT. However, its source code is not always available or easily modifiable in case of hardware. To overcome this problem, this paper applies mutation analysis to a model that describes the relevant aspects of SUT. Belli et al. [1] introduced event sequence graphs (ESGs) for modeling the behavior of an SUT. ESGs consist of vertices (nodes) representing externally observable phenomena ("events") and directed edges (arcs) defining allowed sequences among events. For our approach, an ESG model is mutated by using mutation operators such as insertion and omission of directed edges or vertices. Thus, fault models of SUT are obtained. For each mutant a model-based test case generation algorithm is used to generate a test set. Concluding, SUT is executed against each test set and the mutants are classified as killed or live.

This paper focuses on an investigation of FOMs and HOMs based on mutants generated from ESG and comparison of higher-order mutation operators with basic mutation operators. The primary objective of this paper is to answer the following questions:

- *Do test sets that detect all FOMs detect most of the HOMs, i.e., does the coupling effect hold?*
- *Are basic mutation operators sufficient for measuring test effectiveness of higher-order mutation analysis?*

Now, the question might arise why ESG is used for modeling, and not UML diagrams. Formally speaking, ESG is directed graph enriched by elementary semantics to represent sequential processes, having the same representative power as finite-state automata, which are equivalent to type-3 (regular) grammars [1], [3]. The formalism used in ESG notation enables the direct adoption of mathematical results known from graph theory, automata theory, etc., as needed by our approach.

The rest of this paper is organized as follows: Section 2 presents modeling with ESG. Section 3 introduces basic mutation operators for ESG and a process for ESG-based mutation analysis. A case study in Section 4 validates the approach using a large commercial web-based system. Section 5 concludes the paper summarizing the results and outlines future work.

2 Modeling with Event Sequence Graphs

An event is an externally observable phenomenon, such as an environmental or a user stimulus, or a system response punctuating different stages of system activity. ESGs [1] represent system behavior and user-system interaction focusing on events.

Definition 1. An *event sequence graph* $ESG=(V, E, \Xi, \Gamma)$ is a directed graph with V as a finite set of vertices (events), $E \subseteq V \times V$ a finite set of *arcs*, and $\Xi, \Gamma \subseteq V$ as finite sets of distinguished vertices called *entry* events and *exit* events.

The semantics of an ESG is as follows: Any $v \in V$ represents an event. For two events $v,v' \in V$, the event v' must be enabled after the execution of v if $(v, v') \in E$. $\Xi(ESG)$, $\Gamma(ESG)$ represent the *entry* and *exit* events of a given ESG. To mark the entry and exit events, every $\xi \in \Xi$ is preceded by a pseudo event $[\notin V$ and every $\gamma \in \Gamma$ is followed by $] \notin V$. For each $v \in V$ there is at least one sequence of vertices $(\xi, v_1, \ldots, v_m)$ from $\xi \in \Xi$ to $v_m = v$ and one sequence of vertices $(v_1, \ldots, v_m, \gamma)$ from $v_1 = v$ to $\gamma \in \Gamma$ with $(v_i, v_{i+1}) \in E$, for $i=1,\ldots,m\text{-}1$ and $v \neq \xi$ and $v \neq \gamma$. Fig. 1 shows an example of an ESG with $V=\{a,b,c\}$, $E=\{(a,b), (a,c), (b,c), (c,b)\}$, $\Xi=\{a\}$, and $\Gamma=\{b\}$.

Definition 2. Let $ESG=(V, E, \Xi, \Gamma)$ be an ESG. Any sequence of vertices $(v_1,\ldots,v_m)$ is called an *event sequence* (ES) if $(v_i,v_{i+1}) \in E$, for $i=1,\ldots,m\text{-}1$.

Let α and ω be the functions to determine the entry vertex and exit vertex of an ES. For example, given $ES=(v_1,\ldots,v_m)$, the entry and exit vertices are $\alpha(ES)=v_1$ and $\omega(ES)=v_m$, respectively. Note that the pseudo vertices [,] are not included in the *ES*. An $ES = (v_i, v_j)$ of length 2 is called an *event pair* (EP).

The function l (*length*) determines the number of vertices of an ES. In particular, if $l(ES)=1$ then it is an event sequence of length 1. The pseudo vertices [and] are not included in any ES. Neither are they considered to determine the entry and exit vertices, or to compute the length of an ES.

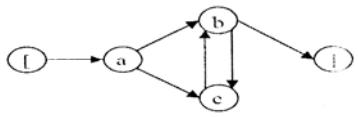

Fig. 1. Example of simple event sequence graph

Definition 3. An ES is a *complete event sequence* (*CES*) if $\alpha(ES)\in \Xi$ and $\omega(ES)\in \Gamma$.

Each CES represents a *walk* from an entry of the ESG to an exit realized by the form: (initial) user inputs→ (interim) system responses → ... → (final) system response. Based on the notion of CES a test case generation algorithm Φ is described by Algorithm 1. Applying algorithm Φ to an ESG (denoted by $\Phi(ESG)$) delivers a test set T that is an ordered pair of (*input to SUT, expected output of SUT*) to be executed against SUT. Details of an algorithm as well as minimization of test sets have been published in previous work ([1]).

Algorithm 1. Test case generation Φ

Input: *ESG* = (*V*, *E*, Ξ, Γ), *len* := maximum length of CES to be covered
Output: Test report of succeeded and failed test cases

FOR *i* := 1 **TO** *len*
BEGIN
  Cover all ESs of *ESG* of length *i* by means of CESs;
END
Apply the test cases given by CESs to SUT; Observe output of SUT;

3 ESG-Based Mutation Analysis

Based on ESG this section defines basic ([2]) and higher-order mutation operators to generate faulty models (mutants) and defines a mutation analysis process using ESG and these operators.

3.1 Basic Mutation Operators

A given ESG specifies the expected, desirable behavior of SUT. An ESG can be changed by manipulating either the arcs or the events resulting in a faulty model ESG*. There are two basic mutation operators that can be defined for different model elements – *insertion* (*I*) and *omission* (*O*).

Definition 4. Basic Mutation operators.

- An *arc insertion operator* (*aI*) extends an ESG by inserting a new arc $\alpha\notin E$ into the given ESG: $aI(ESG, \alpha) := (V, E\cup\{\alpha\}, \Xi, \Gamma)$.

- An *arc omission operator* (aO) removes an arc $\alpha \in E$ from the given ESG: $aO(ESG, \alpha) := (V, E \backslash \{\alpha\}, \Xi, \Gamma)$.
- An *event insertion operator* (eI) extends an ESG by inserting an event e into the given ESG: $eI(ESG, e) := (V \cup \{e\}, E, \Xi, \Gamma)$.
- An *event omission operator* (eO) removes an event $e \in E$ from the given ESG: $eO(ESG, e) := (V \backslash \{e\}, E, \Xi, \Gamma)$.

The eI-operator requires adding extra arcs to/from the inserted event from/to other nodes. After applying an eO-operator, adjacent arcs of event e have to be removed.

The set of first-order basic mutation operators is thereby given by $\Delta_1 := \{aI, aO, eI, eO\}$. Higher-order mutants are constructed as follows:

Definition 5. The set of all n-order ($n>1$) mutation operators are given by $\Delta_n := \Delta_1^n$

Traditional mutation operators as defined in the literature (e.g. [5],[6]) can be directly represented by the basic operators or as a combination. As an example, the operators manipulating the events of an FSM as defined in [6]) can be represented as follows: "event-missing" by basic operator $eO \in \Delta_1$, "event-extra" by basic operator $eI \in \Delta_1$, and "event-exchanged" as a second-order operator $(eO, eI) \in \Delta_2$. The direction of an arc of an ESG is changed by a second-order operator $cdA := (aO, aI)$.

3.2 ESG-Based Mutation Analysis

It is assumed that test cases generated by test case algorithm Φ (Algorithm 1) based on ESG do no reveal any more faults in SUT and that the source code of SUT is not available. Therefore, mutants are constructed based on the model and not the system itself. The goal is to evaluate the fault detection effectiveness of algorithm Φ and its test cases generated, i.e. its capability to distinguish the mutants from SUT. Algorithm 2 describes an ESG-based mutation analysis.

Algorithm 2. ESG-based mutation analysis

Input: *ESG* that describes SUT, $\Delta \subseteq \Delta_1 \cup \ldots \cup \Delta_n$ (set of mutation operators)
k := maximum number of mutants to be generated by each operator
Φ (test generation algorithm)
Output: Killed and live mutants

```
FOREACH δ∈Δ
BEGIN
  FOR i := 1 TO k
  BEGIN
    ESG*δ,i := δ(ESG);
    T*δ,i := Φ(ESG*δ,i);
    Execute SUT against T*δ,i;
    Compare expected output contained in T*δ,i with actual output of SUT;
  END
END
```

An ESG representing SUT and a set of mutation operators $\Delta \subseteq \Delta_1 \cup \ldots \cup \Delta_n$ is used to generate up to k mutants for each mutation operator $\delta \in \Delta$. For each mutant $ESG^*_{\delta,i}$ test case generation algorithm Φ is used to generate a test set $T^*_{\delta,i}$. SUT is executed against this set to compare the predicted output contained in $T^*_{\delta,i}$ with the output observed by SUT. If the output differs, the mutant $ESG^*_{\delta,i}$ is killed. Otherwise, the mutant remains live or is equivalent to the original model.

Definition 6. Given an ESG, a set of mutation operators Δ, the number of mutants k, and a test case generation algorithm Φ, the *test generation mutation score* is defined as: $TGMS(ESG, \Delta, k, \Phi)$:= No. of killed mutants / (all mutants – no. of equivalent mutants).

4 Case Study

To analyze the practicability, characteristic features, and limitations of our approach, an experimental case study based on a commercial web-based system has been conducted. Our main focus is to answer the research questions from the introduction.

4.1 Modeling, Test Generation and Execution

SUT is a commercial web portal (ISELTA – Isik's System for Enterprise-Level Web-Centric Tourist Applications [9]) for marketing touristic services. ISELTA enables hotels and travel agencies to create individual search masks. These masks are embedded in existing homepages of hotels as an interface between customers and the system. Visitors of the website can then use these masks to select and book services, e.g., hotel rooms or special offers. The system also provides other search mask to book arrangements. This part of the system will be used within the case study. Arrangements part of system consists of two parts: additional services and special offers. To set up an additional service and special offer, the dedicated website as shown in Fig. 1 and Fig. 2, respectively, are used. Examples of ESG model for additional services and special offer are given Fig. 3 and Fig. 4, respectively.

Fig. 1. Website to set up additional service

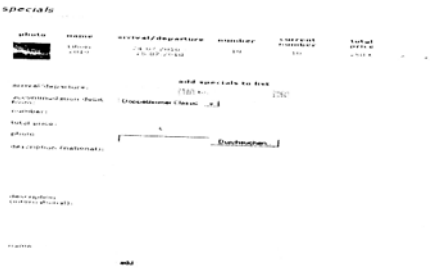

Fig. 2. Website to set up special offer

Test cases were generated for each mutant using the algorithm Φ (as described in Algorithm 1) for *len* = 2, that is, to cover all events and pairs of events by CES. Equivalent mutants were not observed in the case study.

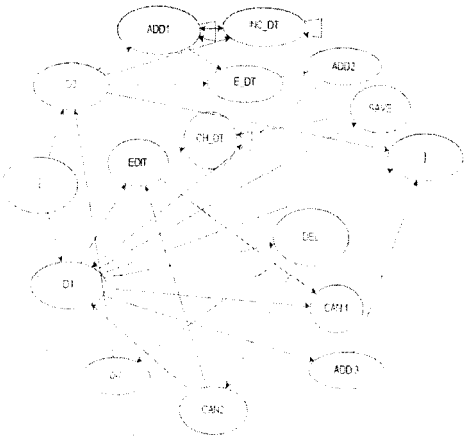

Fig. 3. Example of ESG for additional service

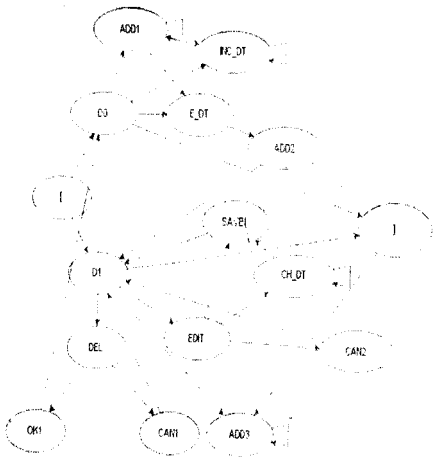

Fig. 4. Example of ESG for special offer

D0 : No Special **D1** : in a Special **INC_DT** :Incomplete Data **EDIT:** Click Edit
CH_DT : Change Information **E_DT** : Enter Complete Data
SAVE : Click Save **CAN** :Click Cancel **OK** : Click OK **ADD** : Click ADD

4.2 Results and Discussion

Table 1, Table 2 and Table 3 summarize the results of mutation analysis. 50 of first-order mutants, 108 of second-order mutants, and 83 of third-order mutants were generated for analysis.

Table 1. Results of First-Order Mutation Analysis

Mutation Operator/ Mutant Type	Killed	Alive	TGMS
All Mutation Op.	**17**	**33**	**0,34**
eO	0	16	0
aI	17	2	0,89
aO	0	15	0

To check the validity of the coupling effect, means of TGMS with respect to the order of the mutants was compared by using One Way ANOVA test. It is a statistical test to determine whether there are differences between different levels of an independent variable or not. In this case study, order of mutants (first-, second-, and third-order) is used as the independent variable.

Coupling Effect Hypothesis (Confidence level is 95%)

H_0: Tests that find first-order mutants also find higher-order mutants
H_1: Higher-order mutation is better in revealing the mutants.

Table 2. Results of Second-Order Mutation Analysis

Mutation Operator / Mutant Type	Killed	Alive	TGMS
All Mutation Op.	**50**	**58**	**0,46**
eO, eO	4	10	0,29
eO, aI	19	3	0,86
aO, aI	16	5	0,76
cdA	9	12	0,43
aI, aI	0	2	0
aO, aO	0	11	0
eO, eO	2	15	0,12

Table 3. Results of Third-Order Mutation Analysis

Mutation Operator / Mutant Type	Killed	Alive	TGMS
All Mutation Op.	**67**	**16**	**0,81**
cdA, aO	12	2	0,86
cdA, aI	12	1	0,92
aO, aI, aI	14	4	0,78
cdA, eO	11	9	0,55
aI, aI, eO	4	0	1
aI, aI, eO	14	0	1

Table 4 shows the results of One Way ANOVA. According to Table 4, H_0 hypothesis can not be accepted at 95% confidence level, since sig. value (0.03) is less than 0.05. This states that there is significant difference between first-order mutation analysis and higher-order mutation. In other words, coupling effect does not hold at 95% confidence for this case study.

Table 4. One Way ANOVA

	Sum of Squares	Mean Square	F	Sig.
Between Groups	1,007	0,504	4,647	**0,03**
Within Groups	1,409	0,108		
Total	2,416			

H_0: Tests that find first-order mutants also find second-order mutants
H_0: Tests that find first-order mutants also find third-order mutants
H_0: Tests that find second-order mutants also find third-order mutants
H_1: There is significant difference between test adequacies.

Table 5. Multiple Comparison Test

(I) MO (J) (MO)	Mean Difference (I-J)	Std. Error	Sig.
First-Order - Second Order	-0,055	0,2272	0,813
- Third Order	**-0,555***	0,233	**0,033**
Second-Order -Third-Order	**-0,5002***	0,183	**0,017**

If H_0 hypothesis can not be accepted as a result of one way ANOVA, the question arises "Which means of groups are significantly different from which other means of groups". To answer the question, Multiple Comparison Tests are used. These tests compare all possible pairs of means of groups and produce which pairs significantly different under selected confidence level. According to Table 5, significant difference between first-order mutation analysis and third-order mutation analysis was found since sig. value (0,033) is less than 0.05. Similarly, it can be said that there is significant difference between second-order mutation analysis and third-order mutation analysis with respect to TGMS (Sig = 0.017 <0.05). Lastly, to decide whether basic mutation operators are sufficient for measuring test effectiveness of higher-order mutation analysis or not, mean of TGMS values are compared with 1, since the test set is said to be effective in detecting faults that that were injected into the mutants, if its TGMS value is 1. To carry out the comparison, One-Sample T-Test that is a statistical test technique used to compare the mean of a sample to a known value was performed. Table 6 shows the results of the comparison. As a result of One-Sample T-Test (Table 6), significant difference between TGMS value of higher-order mutation analysis and 1 was not found.

Table 6. One Sample T-Test

	T	Df	Sig. (2-tailed)	Mean Difference
TGMS	-2,134	5	0,086	-0,1483

5 Conclusions, Future Research

This paper introduced higher-order ESG-based mutation analysis by using two basic operators, insertion and omission, and checked the validity of coupling effect. Analysis was carried out on a case study using the web portal ISELTA. Test sets for first-order, second-order, and third-order mutants were generated and executed on SUT to determine the number of killed and live mutants. Statistical "One Way ANOVA" method enabled following:

- Means of TGMS values were compared to check validity of coupling effect.
- There are significant differences between means of first, second, and third-order mutants were found at 95% confidence level.
- Coupling effect could not be confirmed for this case study. However, it cannot be excluded that the results are affected by potential bias between the operators chosen. The coupling effect may be confirmed when using other higher-order mutation operators or all possible combinations of basic operators as higher-order operators.

In a second step, Multiple Comparison test (LSD) was performed to answer the question "which means of groups (order of mutants) are significantly different from which other means of groups". Founding is:

- No significant difference between first-order and second-order mutation analysis.
- Significant difference between first/second-order and third order mutation analysis.

The additionally performed One-Sample T-Test concluded that higher-order mutation analysis carried out by using combinations of basic mutation operators in this study are

adequate in detecting the injected faults. Work planned is to combine and iterate basic operators for creating more sophisticated mutants to be used in further empirical research.

References

[1] Belli, F., Budnik, C.J., White, L.: Event-based Modeling, Analysis and Testing of User Interactions: Approach and Case Study. Journal of Software Testing, Verification and Reliability 16(1), 3–32 (2006)

[2] Belli, F., Budnik, C.J., Wong, W.E.: Basic Operations for Generating Behavioral Mutants. In: Proc. 2nd Workshop on Mutation Analysis (2006)

[3] Belli, F.: Finite-State Testing and Analysis of Graphical User Interfaces. In: Proc. 12th IEEE International Symposium on Software Reliability Engineering, pp. 34–43 (2001)

[4] DeMillo, R.A., Lipton, R.J., Sayward, F.G.: Hints on Test Data Selection: Help for the Practicing Programmer. Computer 11(4) (1978)

[5] Fabbri, S.C.P.F., Maldonado, J.C., Delamaro, M.E., Masiero, P.C.: Mutation Analysis Testing for Finite-State Machines. In: Proc. 5th IEEE International Symposium on Software Reliability Engineering, pp. 220–229 (1994)

[6] Fabbri, S.C.P.F., Maldonado, J.C., Sugeta, T., Masiero, P.C.: Mutation Testing Applied to Validate Specifications Based on Statecharts. In: Proc. 10th IEEE International Symposium on Software Reliability Engineering, pp. 210–219 (1999)

[7] Hamlet, R.G.: Testing Programs with the Aid of a Compiler. IEEE Transactions on Software Engineering 3(4) (1977)

[8] Harman, M., Jia, Y., Langdon, W.B.: A Manifesto for Higher Order Mutation Testing. In: Proc. 5th International Workshop on Mutation Analysis (2010)

[9] ISELTA, `http://www.iselta.de/`

[10] Jia, Y., Harman, M.: Constructing Subtle Faults Using Higher Order Mutation Testing. In: SCAM, pp. 249–258 (2009)

[11] Jia, Y., Harman, M.: Higher Order Mutation Testing. Journal of Information and software Technology 51(10), 1379–1393 (2009)

[12] Mathur, A.P.: Performance, Effectiveness and Reliability Issues in Software Testing. In: Proc. of 5th International Computer Software and Applications Conference, Tokyo, Japan, pp. 604–605 (1991)

[13] Mresa, E.S., Bottaci, L.: Efficiency of Mutation Operators and Selective Mutation Strategies: An Empirical Study. Software Testing, Verification and Reliability 9(4), 205–232 (1999)

[14] Offutt, A.J., Rothermel, G., Zapf, C.: An Experimental Evaluation of Selective Mutation. In: Proc. 15th International Conference on Software Engineering (ICSE 1993), pp. 100–107. IEEE Computer Society Pres., Baltimore (1993)

[15] Offutt, A.J.: Investigations of the Software Testing Coupling Effect. ACM Transactions on Software Engineering and Methodology 1(1), 5–20 (1992)

[16] Offutt, A.J.: The Coupling Effect: Fact or Fiction? In: Proc. 3rd Symposium on Software Testing, Analysis and Verification, Key West, FL, pp. 131–140 (1989)

[17] Schuler, D., Dallmeier, V., Zeller, A.: Efficient Mutation Testing by Checking Invariant Violations. In: Proc. 2009 International Symposium on Software Testing and Analysis, Chicago, pp. 69–80 (2009)

[18] Wah, K.S.H.T.: An Analysis of the Coupling Effect I: Single Test Data. Science of Computer Programming 48(2-3), 119–161 (2003)

[19] Wah, K.S.H.T.: A Theoretical Study of Fault Coupling. Software Testing, Verification & Reliability 10(1), 3–45 (2000)

[20] Zhu, H., Hall, P.A.V., May, J.H.R.: Software Unit Test Coverage and Adequacy. ACM Computing Surveys 29(4), 366–427 (1997)