Science of Computer Programming 120 (2016) 25–48

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Model-based mutation testing—Approach and case studies ☆

Fevzi Belli [a], Christof J. Budnik [b], Axel Hollmann [c], Tugkan Tuglular [d,*], W. Eric Wong [e]

[a] *Department of Computer Science, Electrical Engineering and Mathematics, University of Paderborn, Germany*
[b] *Siemens Corporation, Corporate Technology, USA*
[c] *andagon GmbH, Cologne, Germany*
[d] *Department of Computer Engineering, Izmir Institute of Technology, Turkey*
[e] *Department of Computer Science, University of Texas at Dallas, USA*

A R T I C L E   I N F O

Article history:
Received 19 July 2014
Received in revised form 7 December 2015
Accepted 14 January 2016
Available online 21 January 2016

Keywords:
Mutation testing
Model-based testing
Model-based mutation testing
Mutation operator
Fault detection capability

A B S T R A C T

This paper rigorously introduces the concept of model-based mutation testing (MBMT) and positions it in the landscape of mutation testing. Two elementary mutation operators, *insertion* and *omission*, are exemplarily applied to a hierarchy of graph-based models of increasing expressive power including directed graphs, event sequence graphs, finite-state machines and statecharts. Test cases generated based on the mutated models (*mutants*) are used to determine not only whether each mutant can be killed but also whether there are any faults in the corresponding system under consideration (SUC) developed based on the original model. Novelties of our approach are: (1) evaluation of the fault detection capability (in terms of revealing faults in the SUC) of test sets generated based on the mutated models, and (2) superseding of the great variety of existing mutation operators by iterations and combinations of the two proposed elementary operators. Three case studies were conducted on industrial and commercial real-life systems to demonstrate the feasibility of using the proposed MBMT approach in detecting faults in SUC, and to analyze its characteristic features. Our experimental data suggest that test sets generated based on the mutated models created by *insertion* operators are more effective in revealing faults in SUC than those generated by *omission* operators. Worth noting is that test sets following the MBMT approach were able to detect faults in the systems that were tested by manufacturers and independent testing organizations before they were released.

1. Introduction

Testing is one of the most commonly used techniques for assuring quality in the software industry. It entails the execution of the software[1] in its real environment, under real-life conditions. The initial step of software development is usually the requirements elicitation, and its outcome is the specification of the system's behavior. It is important to generate test cases and define appropriate testing processes at this early stage, long before the implementation begins, in compliance

☆ Authors are listed in the alphabetical order of their last names.
* Corresponding author.
E-mail addresses: belli@adt.upb.de (F. Belli), christof.budnik@siemens.com (C.J. Budnik), a.hollmann@andagon.com (A. Hollmann), tugkantuglular@iyte.edu.tr (T. Tuglular), ewong@utdallas.edu (W.E. Wong).
[1] The terms "program," "software," "application," and "implementation" are used interchangeably, whenever appropriate.

http://dx.doi.org/10.1016/j.scico.2016.01.003
0167-6423/© 2016 Elsevier B.V. All rights reserved.

with the user's expectancy of how the system should behave. An advantage of this is to avoid faults being detected at late stages, when they could have been detected much earlier and fixed less expensively. There are techniques to visualize and represent the relevant features of the system under consideration (SUC) in its environmental context, leading to a *model* of the SUC. These models are increasingly being used as a basis (in the context of model-based testing (MBT)) to generate test cases, which satisfy certain criteria, for example, coverage of every node in the model. These tests are then used to validate the SUC. The assumption is that the model is correct, and if the behavior of SUC is consistent with that of the model during conformance testing against test cases generated above, then the SUC must also be correct.

A broad variety of formal or informal models exist for modeling software as recommended in standards such as UML [1] or TTCN-3 [2]. These models describe the SUC at different levels of granularity and preciseness. Graph-based models consist of nodes and arcs. The semantic meaning associated with these nodes and arcs determines the level of precision of the SUC description. The issue of which model to use is not determined solely by identifying which is most suitable for modeling the SUC, but also by taking into account other factors (especially when considering the proposed MBMT approach in Section 3) such as how easily the model can be mutated with respect to certain mutation operators and how test cases can be generated for the model. In this paper, we choose the following formal graph-based models in order of increasing expressive power: (i) Directed graphs (DG) include no semantics; that is, nodes and arcs have no interpretation. The paper uses them to introduce notions, especially mutation operators, syntactically. (ii) Event sequence graphs (ESG) interpret nodes as events and arcs as sequences of events [3]. (iii) Finite-state machines (FSM) interpret nodes as states, whereas arcs labeled by events symbolizing state transitions. (iv) Statecharts (SC) consider concurrency and hierarchy aspects [4,1].

A fundamental problem of testing stems from the huge variety of possible software faults to be considered. For this purpose, *mutation testing* techniques generate faulty versions of a software system (namely, *mutants*) by introducing simple but representative faults using mutation operators (which are also known as mutant operators in the literature). Different strategies have been proposed for mutation testing. Some are applied to source code of the SUC while others are applied to models derived from specifications. Mutants generated can be used at the unit testing [5,6], as well as at the integration testing level [7] (further details on related work are given in Section 6). A common problem of these studies is that they use a large number of mutation operators to generate mutants because they *empirically* determine the mutation operators based on selected fault models. This makes it arguable as to the exact number of operators that should be developed for a given fault model. A different approach is used in this paper by introducing two *elementary* mutation operators: *insertion* and *omission*. Starting with a DG as the underlying model, the elementary operators are applied to other models, such as ESG, FSM, and SC in the same manner. This is also a major difference between this paper and other studies, such as [7–16]. Many mutation operators known from literature can be reduced to these two elementary operators and their combinations and iterations. For example, the "sdl" operator in the Mothra tool set [17] deletes a statement that is equivalent to an "omission" operator, and the "svr" operator does a scalar variable replacement that is an "omission" followed by an "insertion." In fact, mutants generated by the mutation operators in the aforementioned literature could be viewed as special cases of the fault model developed in this paper. However, mutants generated by the two elementary operators (with multiple iterations) and their combinations presented in this paper might not be generated by the operators introduced by other authors. Refer to Section 4.4 for more details.

The primary objective of this paper is to propose a precise *model-based mutation testing* (MBMT) approach. We view the SUC as a black box and assume that the source code is not available. Hence, it cannot be mutated. Instead, a model (say $\mathcal{M}$) is available and will be mutated. We further assume that the conformance between $\mathcal{M}$ and the corresponding SUC has also been validated by a test set $\mathcal{T}$ created by applying an appropriate test generation algorithm (say Φ) to $\mathcal{M}$. The proposed elementary mutation operators, *insertion* and *omission*, generate a set of mutated models (mutants) when applied to $\mathcal{M}$. The same test generation algorithm Φ which has been applied to $\mathcal{M}$ will be applied to these mutants to generate a test set for each mutant. While using test cases from these sets to determine whether a mutant can be killed, the same test cases can help us detect faults in SUC which have not been revealed during the conformance testing described above using $\mathcal{T}$. In other words, we attempt to assess the fault detection capability of $\mathcal{T}$. This is definitely a different view than the ones found in the literature, e.g., [18]. Note also in contrast to the code-based mutation testing, mutants in MBMT are classified into more categories as described in other approaches to MBMT. The later sections, especially Section 3, will precisely explain this novel interpretation of MBMT.

Three case studies on industrial and commercial applications covering interactive, reactive, and proactive systems were conducted to demonstrate the feasibility of using the proposed MBMT approach and to discuss its characteristic features. The results of our experiments suggest that tests generated by the proposed approach can be effective in detecting faults in the corresponding SUC. For example, in the third case study (the control terminal of a marginal strip mower), the test cases generated by the approach described here could detect faults in the released version that were not found by the previous (and independent) testers, such as a technical control board in Germany.

The remainder of the paper is organized as follows. Section 2 summarizes MBT and reviews the models and test generation processes selected for our study. Section 3 discusses MBMT and explains the difference between this new approach and the code-based mutation testing. Section 4 defines the elementary mutation operators and compares them with other mutation operators reported in the literature. Section 5 presents case studies including results, analysis and discussion. Section 6 gives related work. Our conclusion and future work appear in Section 7.