Information and Software Technology 000 (2016) 1–13

Contents lists available at ScienceDirect

Information and Software Technology

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

A novel use of equivalent mutants for static anomaly detection in software artifacts

Paolo Arcaini[a,*], Angelo Gargantini[b], Elvinia Riccobene[c], Paolo Vavassori[b]

[a] Faculty of Mathematics and Physics, Charles University in Prague, Czech Republic
[b] Department of Economics and Technology Management, Information Technology and Production, Università degli Studi di Bergamo, Italy
[c] Dipartimento di Informatica, Università degli Studi di Milano, Italy

A R T I C L E I N F O

Article history:
Received 12 August 2015
Revised 29 January 2016
Accepted 30 January 2016
Available online xxx

Keywords:
Equivalent mutant
Static anomaly
Quality measure

A B S T R A C T

Context: In mutation analysis, a mutant of a software artifact, either a program or a model, is said equivalent if it leaves the artifact *meaning* unchanged. Equivalent mutants are usually seen as an inconvenience and they reduce the applicability of mutation analysis.

Objective: Instead, we here claim that equivalent mutants can be useful to define, detect, and remove *static anomalies*, i.e., deficiencies of given *qualities*: If an equivalent mutant has a better quality value than the original artifact, then an anomaly has been found and removed.

Method: We present a process for detecting static anomalies based on mutation, equivalence checking, and quality measurement.

Results: Our proposal and the originating technique are applicable to different kinds of software artifacts. We present anomalies and conduct several experiments in different contexts, at specification, design, and implementation level.

Conclusion: We claim that in mutation analysis a new research direction should be followed, in which equivalent mutants and operators generating them are welcome.

1. Introduction

Mutation has a long history and has been applied to several areas of software engineering [1] and to different kinds of software artifacts, as code and formal specifications [2–4]. The main application of mutation analysis is *mutation testing* [5], in which faults are artificially introduced in the code under test and test cases are used to detect (or *kill*) these faults (*mutants*). Good tests can kill all the injected faults in the program or, at least, most of them: a test suite has a *mutation score* equal to the portion of mutants it can kill. However, some mutants are impossible to kill since only *behavioral* faults can be detected by a test: a mutant is said *equivalent* if it leaves the behavior of the program unchanged. Equivalent mutants cannot be detected by a test and thus they reduce the mutation score of a test suite without a real justification. Also in test generation, equivalent mutants pose a challenge: they consume resources without producing any useful test. Equivalent mutants are therefore usually seen as an inconvenience and the equivalent mutant problem is considered as one of the main causes why mutation testing is seldom used in practice [5,6]. Several attempts have been proposed to eliminate them (e.g., by filtering), or to automatically find and avoid them [7,8].

Due to the aforementioned problems, equivalent mutants earned a bad reputation. Following our early position paper [9], we aim in this paper at *rehabilitating* equivalent mutants reputation, since we claim that they can be useful to discover *non-behavioral faults*, a category of faults that has not been targeted in mutation analysis so far. When looking for non-behavioral faults, equivalent mutants should be seen as an *opportunity*, and the long time experience in finding them should be reused.

We define a certain type of non-behavioral faults, that we call *static anomalies*, in terms of equivalent mutants. We show that if, given an artifact A and a quality of A (like readability, efficiency and, so on), we are able to produce an equivalent mutant with better quality than A, then A contains a static anomaly that should be removed. Differently from classical approaches targeting behavioral faults, we aspire to have a lot of equivalent mutants, since they can be used to detect anomalies.

We present a process for detecting static anomalies based on mutation, equivalence checking, and quality checking. We show

* Corresponding author. Tel.: +420 221914285.
E-mail address: arcaini@d3s.mff.cuni.cz (P. Arcaini).

http://dx.doi.org/10.1016/j.infsof.2016.01.019
0950-5849/© 2016 Elsevier B.V. All rights reserved.

that this process is applicable to several types of artifacts produced at different phases of the software life cycle (at specification, design, and implementation levels), for several anomalies, and using several mutation operators.

The paper is organized as follows. Section 2 introduces some background on the classical definition of software anomaly, mutation, and the problem of equivalent mutants. Section 3 presents our definition of static anomalies in terms of equivalent mutants and a technique for discovering them. Sections 4 and 5 show the application of the technique at the specification level (on feature models and NuSMV models), Sections 6 and 7 at the implementation level (on Boolean expressions and source code), and Section 8 on package dependency graphs (either at the design or implementation level). Section 9 discusses some threats to the validity of our proposal, while Section 10 presents some related work. Finally, Section 11 concludes the paper.

2. Background

We here briefly review some basic concepts on software anomalies, mutation, and equivalent mutants.

2.1. Software anomalies

Software anomalies are defined in the IEEE standard [10] as:

Any condition that deviates from the expected based on requirements specifications, design documents, user documents, standards, etc. or from someone' s perceptions or experiences. Anomalies may be found during, but not limited to, the review, test, analysis, compilation, or use of software products or applicable documentation.

In this paper, we refer to *software artifact* as any product that is developed along the software life cycle at several levels: for example, source code at implementation level or models at specification level.

According to the IEEE standard, each software artifact should have some *quality* attributes (like *readability, compactness, efficiency, correctness*, etc.) and an anomaly is any deviation in terms of the expected (quality) attributes. For example, faults represent deviations w.r.t. the expected behavior, dead code is a deviation w.r.t. compactness.

We here focus on *static anomalies*, i.e., anomalies that can be removed without changing the “meaning” of the artifact. Static anomalies regard the artifacts' structure and they relate to qualities that may be statically measured.

2.2. Mutation

Mutation is a well known technique in the context of software artifacts as program code and formal specifications. It consists in introducing small modifications into the artifact such that these simple syntactic changes, called *mutations*, represent typical mistakes that programmers or designers often make. These faults are deliberately seeded into the original artifact in order to obtain a set of faulty variations called *mutants*. A transformation rule generating a mutant from the original artifact is known as *mutation operator*.

Mutation is very often used in combination with program testing, and its use is twofold. Mutants are classically used to assess the quality of test suites. High quality test suites should be able to distinguish the original program from its mutants, i.e., to detect the seeded faults. Given a test suite T, if the result of running a mutant is different from the result of running the original program for at least a test case in T, then the mutant is said to be *killed*; otherwise, it is said to have *survived*. A test suite has a *mutation score* equal to the portion of mutants it can kill. After all test cases have been executed, there may still be a few surviving mutants. To improve the test suite T, the program tester can provide additional test inputs to kill these surviving mutants. In this case, mutation is used for *test generation* purposes.

The history of mutation can be traced back to the 70s [1]. Mutation has been mainly applied to programming languages, but also at the design level to formal specifications [2–4,11–13].

2.3. Equivalent mutants

When a mutant has the same meaning (e.g., the same behavior for programs or the same logical models for Boolean expressions) as the original artifact, it is said to be *equivalent*. These mutants are syntactically different but semantically equivalent to the original artifact.

Equivalent mutants are considered as one of the main causes why mutation testing is seldom used in practice [5,6]. In software testing, equivalent mutants do not represent actual faults and cannot be detected (killed) by a test. They thus reduce the quality index (mutation score) of a test suite without a real justification. In test generation, equivalent mutants consume resources without producing any useful test.

In code mutation, automatically detecting all equivalent mutants is impossible [14] because program equivalence is undecidable [15]. Several attempts try to eliminate (e.g., by filtering) or to avoid them [7,8].

In the context of other software artifacts with a higher abstraction level than code, and with a concept of equivalence and a technique for checking it, some approaches for detecting equivalent mutants have been developed [2,3]. However, also in these contexts, equivalent mutants are seen as an inconvenience [16,17] and efforts to detect them are only finalized to skip them.

In the approach presented here, we rehabilitate equivalent mutants, in the sense that the goal of detecting them is finalized to use them to improve artifacts' qualities, and not to eliminate or avoid them.

3. Using mutation to detect static anomalies

In this section, we introduce our definition of static anomalies in terms of equivalent mutants, and we propose a technique for anomaly detection.

3.1. Static anomalies

We define the concept of static anomaly in terms of equivalence and quality of artifacts. We assume that one can define a quality q over artifacts and that q induces a partial order (of better quality) $>_q$ among all the artifacts, i.e., an artifact may be *better* than another one in terms of a certain quality q. Whenever possible, we will define q as a real-valued function over the considered artifacts, such that q induces a total order. Moreover, we assume that it is possible to check equivalence among artifacts.

Given a certain quality q, an artifact may contain a static anomaly in terms of q if the following condition holds:

Definition 1 (Static anomaly detection)**.** Given an artifact A and its mutation A', if A' is equivalent to A (i.e., $A \equiv A'$) and $A' >_q A$, then A contains a static anomaly. The static anomaly is the *difference* between A' and A.

Thesis 1. Each classic static anomaly introduced in the literature can be redefined in terms of Definition 1.

3.2. Detecting static anomalies

Definition 1 gives the foundation of a methodology for defining, finding, and possibly removing static anomalies. Normally, the