Test Suite Quality Assessment Using Model Inference Techniques

Hermann Felbinger
VIRTUAL VEHICLE Research Center
Austria, 8010 Graz
Email: hermann.felbinger@v2c2.at

***Abstract*—To state whether a System Under Test is sufficiently tested requires an assessment of the test suite quality. Existing methods to assess the quality of a test suite either are based on the structure of an implementation or determine the quality using mutation score. In this paper we introduce a method, which is based on inductive inference to assess the quality of a test suite and propose a method to augment a test suite depending on the quality assessment result. In this paper we provide a short glimpse on our objectives and show preliminary results of model inference of a test suite.**

I. Introduction

The primary goals of testing are to provide information to determine the current quality of the System Under Test (SUT) and to find defects in the SUT prior to its release. But testing causes high effort and therefore we work on automating tasks in software testing. In this work the focus is on the task of test case design. A test case is an input/output pair of a known input to the SUT and an expected output respectively. In this work test cases are used to ensure functional suitability of a software product and the test cases are automatically designed (referred to as test case generation in the remainder of this work).

The main objective is to design test cases that are effective at finding deficiencies in the SUT without requiring an excessive number of tests to be carried out, because exhaustive testing is not possible due to exponential growth of the number of possible input values. Test case design depends on a criterion which states when an SUT is sufficiently tested. An overview of these criteria is provided in `ISO/IEC/IEEE 29119-4` where the criteria are labeled *test design techniques*. These *test design techniques* are divided in the three main groups *specification-based*, *structure-based*, and *experience-based*.
Due to the emergence of model driven development approaches test case generation, which is based on models is becoming more and more important. Hence the generation of large test suites, which contain a vast number of test cases becomes possible. But a large test suite does not automatically imply that all test cases are useful (e.g. redundant test cases can occur where selecting one of them yields the same contribution to achieve a criterion which states when an SUT is sufficiently tested as selecting all of them).

Test suite efficacy is a measure of its fault detection ability what is an expression of test suite quality. Efficacy can be measured using mutation score [1], which requires to execute the test suite on created mutants.
Here we suggest an approach, which does not require to execute the test suite and is therefore, a very cost efficient quality assessment method for a test suite. In this approach we infer a model from a test suite and assess the quality of the test suite determined by a new metric, which is based on the differences between the original model (e.g. implementation) and the inferred model from the test suite. Intuitively, if the inferred model coincides with the original model then we conclude that the test suite has the highest quality, as any program that passes such an exhaustive test suite must have implemented the required model. The test suite quality metric depends on the determination of (program-) model-equivalence, which is, in general, undecidable. Therefore we must be willing to consider practically attainable approximations in order to make this quality assessment method usable. If a difference is determined we assess the existing test suite according to the difference and create additional test cases to reduce or at best eliminate the difference.

II. Positioning in the State of the Art

The quality of a test suite can be expressed currently in two different ways. First the quality is expressed in coverage as calculated in equation $C = \frac{N}{T} * 100$ where C is the coverage in % satisfied of a test coverage criterion, N is the number of test coverage items (e.g. conditions, decisions, branches, ...) for which a test case exists in the test suite, and T is the total number of test coverage items identified in the SUT. Second way to express the quality is by measuring the fault detection capability of the test suite by determining the mutation score. An example, where the fact that a test suite satisfying purely a coverage criterion is a poor indication of test suite quality by measuring the mutation score is verified, is provided in [2]. In [2] the authors show industrial use cases where a randomly generated test suite is clearly more efficient in fault detection than a test suite satisfying the Modified Condition/Decision Coverage (MCDC) criterion (randomly generated test suite has equal size).
The mutation score is determined by mutation based testing. Mutation based testing techniques where different faulty implementations are created and test cases are generated, which reveal these faults [3] are not well established yet. The reasons that mutation based testing is not well established originates in the high costs of execution time, hardly available tool support, and the fact that mutation based testing is biased by the type of mutants.
Elaine Jessica Weyuker introduced in [4] a similar approach to ours where she infers a program Q from executing a test suite T and determines the inference adequacy by approximations of the equivalence relation $P \equiv Q$ with the original program

978-1-4799-7125-1/15/$31.00 ©2015 IEEE

P, meaning that $P(x) = Q(x)$ for every input x. Satisfying the inference adequacy guarantees that a test suite must truly test each portion of the program code. In our work we use a passive approach to infer a model where executing the test suite is not required. A passive method to infer a model from a test suite to assess the quality of the test suite is shown in [5]. In [5] the authors provide metrics, which can be measured on the inferred model (e.g. missclassification, unused category, etc.). In our work we refine these metrics to assess the inferred model but focus on the equivalence of the inferred model and the original model similar to the ideas in [4].

III. Scientific and Technical Objectives

We assess the quality of a test suite by determining the equivalence of an inferred model and the original model. In [6] the authors prove that if there is a computable procedure for checking if two programs are equivalent, then there is also a computable procedure for generating inference adequate test cases for a program and vice versa. They also show that, in general, neither of these computable procedures exists. Thus there cannot be a complete algorithmic solution to the equivalence problem. That is detecting equivalence between two arbitrary models (programs) is an undecidable problem.

Because equivalence is undecidable we have to find methods to calculate approximations of equivalence. In this work we investigate equivalence checking based on structural similarities [7], logical equivalence, and we will demonstrate non-equivalence by generating distinguishing test cases similar to [8] which yield different results when executing them on the inferred model and on the original model. Applying these methods we obtain results, which assess the current quality of a test suite. A test suite is of highest quality, if the inferred model coincides with the original model.

IV. Preliminary Results

To show initial results of our approach we use an example test suite which satisfies one of the strongest *structure-based* criteria, MCDC, infer a model and assess the quality of the test suite by determining the difference of the inferred model and the original model.
Figure 1 presents the original model of a decision with an initial state q_a and three outcome states q_b, q_c, q_d (outcomes). For each outcome a guard in form of propositional logic exists. The given decision is complete and deterministic because for each valuation of the variables in the decision exactly one guard evaluates to true. We created a set of five test cases which satisfy MCDC for the given decision:
$t_1 : x = 1,\ y = 1,\ z = 2 \mid t_2 : x = 0,\ y = 1,\ z = 1$
$t_3 : x = 1,\ y = 0,\ z = 1 \mid t_4 : x = 1,\ y = 1,\ z = 0$
$t_5 : x = 0,\ y = 1,\ z = 0$

We use these test cases as input for the *Weka 3.6*[1]*: Data Mining Software*, which is a collection of machine learning algorithms for data mining tasks. With Weka we created a decision tree using the C4.5 algorithm. From the decision tree we can create a decision which is comparable to the original decision as shown in Figure 2. The inferred decision can be compared to the original decision, by comparing the guards for each and every outcome with the guards in the original decision. We compare the guards by checking if they are logically equivalent applying an SMT-Solver. Two decisions are logically equivalent if the two have the same sets of outcomes, and for each outcome the guards are logically equivalent which is clearly not the case in the given example.

[1] http://www.cs.waikato.ac.nz/ml/weka/

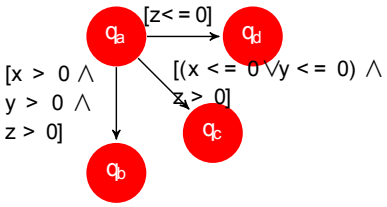

Fig. 1. Original decision with multiple outcomes

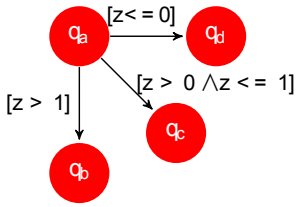

Fig. 2. Inferred decision with multiple outcomes

V. Work Plan

The equivalence check is a crucial part of this work. To determine the equivalence of the inferred model and the original model, we implement different approaches as mentioned in Section III to evaluate which is the best applicable approach. Additionally we will investigate different algorithms to infer a model and the language space which constrains the program constructs which we can use in this quality assessment approach. Finally we implement a method to augment a test suite depending on the quality assessment result.

VI. Acknowledgment

I would like to thank my supervisor Franz Wotawa for his help and support. I also thank my advisor Alexandre Petrenko for his inspiration and help.

References

[1] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *Software Engineering, IEEE Transactions on*, vol. 37, no. 5, pp. 649–678, Sept 2011.

[2] M. Staats, G. Gay, M. Whalen, and M. Heimdahl, "On the danger of coverage directed test case generation," in *Proceedings of the 15th International Conference on Fundamental Approaches to Software Engineering*, ser. FASE'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 409–424. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-28872-2_28

[3] B. Aichernig, "Model-based mutation testing of reactive systems," in *Theories of Programming and Formal Methods*, ser. Lecture Notes in Computer Science, Z. Liu, J. Woodcock, and H. Zhu, Eds. Springer Berlin Heidelberg, 2013, vol. 8051, pp. 23–36. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-39698-4_2

[4] E. J. Weyuker, "Assessing test data adequacy through program inference," *ACM Transactions on Programming Languages and Systems*, vol. 5, pp. 641–655, 1983.

[5] L. C. Briand, Y. Labiche, and Z. Bawar, "Using machine learning to refine black-box test specifications and test suites," *Quality Software, International Conference on*, vol. 0, pp. 135–144, 2008.

[6] T. Budd and D. Angluin, "Two notions of correctness and their relation to testing," *Acta Informatica*, vol. 18, no. 1, pp. 31–45, 1982. [Online]. Available: http://dx.doi.org/10.1007/BF00625279

[7] C. A. J. V. Eijk, "Sequential equivalence checking based on structural similarities," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, pp. 200–0, 2002.

[8] F. Wotawa, M. Nica, and B. Aichernig, "Generating distinguishing tests using the minion constraint solver," in *Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on*, April 2010, pp. 325–330.