Targeted Mutation: Efficient Mutation Analysis for Testing Non-Functional Properties

Björn Lisper*, Birgitta Lindström†, Pasqualina Potena‡, Mehrdad Saadatmand‡ and Markus Bohlin‡
*Mälardalen University, Sweden
†University of Skövde, Sweden
‡SICS Swedish ICT Västerås AB, Sweden
Email: *bjorn.lisper@mdh.se, †birgitta.lindstrom@his.se, ‡{pasqualina.potena, mehrdad.saadatmand, markus.bohlin}@sics.se

***Abstract*—Mutation analysis has proven to be a strong technique for software testing. Unfortunately, it is also computationally expensive and researchers have therefore proposed several different approaches to reduce the effort. None of these reduction techniques however, focuses on non-functional properties. Given that our goal is to create a strong test suite for testing a certain non-functional property, which mutants should be used? In this paper, we introduce the concept of *targeted mutation*, which focuses mutation effort to those parts of the code where a change can make a difference with respect to the targeted non-functional property. We show how targeted mutation can be applied to derive efficient test suites for estimating the Worst-Case Execution Time (WCET). We use program slicing to direct the mutations to the parts of the code that are likely to have the strongest influence on execution time. Finally, we outline an experimental procedure for how to evaluate the technique.**

I. Introduction

Embedded systems permeate our daily lives. As they are part of our environment, where they often perform critical functions, the requirements on them are stronger than on desktop systems. In particular *non-functional properties*, such as performance, efficiency, robustness, safety, and reliability, are becoming increasingly important. It is thus important to have good methods for verifying these properties. In particular, it is important to have good testing practices for this purpose.

The efficient testing for non-functional properties relies on the existence of effective test suites that target these properties. What is "efficient" depends on the property in question: for non-functional resource properties, such as energy consumption or execution time, it is often of interest to find the extremes. An effective test suite should thus aim at finding the "corner cases" with a limited set of test cases.

One way to create effective test suites is *mutation testing*. Mutation testing injects changes ("mutations") into the code of the system under test. The changed programs ("mutants") can then be used to check how effective the different test cases are at detecting the changes ("killing the mutants"). This information can be used to increase the fault detection reflectiveness of the test suites.

However, to be useful, mutants should be distinguishable from the original program with respect to the non-functional property under investigation. For instance, if testing for execution time, then a mutant with the same execution time as the original program will be useless for this purpose, since no test case will be able to distinguish the mutant from the original program. Mutations that do not alter the tested property should thus be avoided.

Resource properties like energy or memory consumption, or execution time, are often more or less strongly associated with different parts of the code (e.g., memory allocation statements for memory consumption). We therefore introduce the concept of *targeted mutation*, which will mutate such parts of the code where a change in the code will likely change the property under study. The aim is to reduce the number of indistinguishable (or "equivalent") mutants, thereby creating a set of mutants that is more adequate for creating effective test suites.

In this paper we specifically target the Worst-Case Execution Time (WCET), which is an important property for real-time tasks. We propose a form of targeted mutation analysis where the mutations are applied to the parts of the code that are most likely to significantly affect the execution time. We also outline a method for experimental evaluation of the technique, where the targeted mutations are used to optimize test suites for WCET estimation.

A. Mutation analysis

Mutation analysis was originally introduced by DeMillo et al. [1]. In mutation analysis, multiple (often faulty) versions of the software, *mutants* are created by systematically applying rules for changing syntactic elements, *mutation operators* (or mutators) [2]. Test suites are run against the mutants to determine the percentage of mutants the tests will detect, called the *mutation adequacy score*. The mutation adequacy score is a coverage criterion, like statement and data flow coverage, but has been found to be stronger than other known criteria and is thus often referred to as a "gold standard" [2]. Mutation is unique among coverage criteria in that it not only requires a test to reach a location in the program (the mutated statement), but it also requires the test to create an error in the program state and in case of strong mutation, propagate that error to an output of the program. Mutators have been created for many different languages, including Fortran, Java, and C [3], [4], [5]. Mutation analysis has also been used for testing extra-functional properties [6], [7], [8], [9].

Mutation analysis is a strong technique for testing [2] but it can also be computationally expensive. The major reason

978-1-5090-6676-6/17 $31.00 © 2017 IEEE
DOI 10.1109/ICSTW.2017.18

is the high number of generated mutants and the fact that most of these mutants do not contribute to the quality of the analysis [10]. Equivalent mutants are especially difficult since they cannot be detected by any test [10]. Therefore, strategies to avoid generating equivalent mutants are of great interest.

B. WCET Analysis for Real-Time Systems

Many embedded systems are real-time systems, where it has to be verified that timing constraints are met. The verification is often divided into two levels: *system-level analysis*, where the timing of systems of recurring tasks is analyzed, and *code-level analysis* where the codes of individual tasks are analyzed w.r.t. their timing properties. In particular, code-level analysis seeks to bound the WCET, which in essence is the longest time to execute a piece of code if it runs uninterrupted on some given target hardware.

The execution time of an uninterrupted task depends on two factors: the executed path through the code, and the execution times for the instructions that are executed on this path. *Static WCET analysis* [11] tries to find tight bounds for both these factors in order to compute a safe and tight WCET bound. Industrial practice, however, is to rather estimate the WCET from measured execution times. As it is important not to underestimate the WCET, effective test suites are needed for this. Mutation analysis can be used to construct such test suites by favoring test cases that detect the changes in execution time caused by the mutations.

II. Methodology

The goal of the work initiated in this paper is to show how targeted mutation analysis can be used to improve test suites for WCET estimation from timing measurements. First we need to identify the parts of the code that are likely to have a strong influence on the execution time. Then we create mutations in those parts, and compare the execution times of the mutants with the execution time of the original program for the different test cases. The results can then be used to prioritize the test cases. This paper presents the initial steps in this approach.

The key assumption underlying the approach is that the control flow (e.g., number of loop iterations) will have the strongest influence on the variability of the execution time of a program. Thus, mutations that can affect the control flow are more likely to yield mutants that are not timing-equivalent to the original program. Other mutations can of course still affect the execution time due to hardware effects, like altering the cache state, but these can be expected to affect the execution time less than, say, a change in the number of loop iterations. Thus we restrict the mutations to appear only in parts of the code that may affect the program flow. As the program flow is determined by the conditions in the program, we apply *static backwards program slicing* [12] with respect to these and perform the mutations only in this slice. Mutation operators can be defined by empirically investigating the coupling effect between real faults - leading to temporal failures - and mutants generated with commonly used mutation operators, as inspired by the study in [13].

We can improve the selection of targeted code further by observing that some control flow is input-independent, and thus will be the same for all test cases. The archetypal example is a for-loop that always iterates the same number of times. The effect on execution time from mutations of such code will be more or less the same for all test cases, and will thus not be very helpful for prioritization. Therefore, the parts of the slice that are not dependent on any inputs can also be removed. Standard methods for program slicing, like PDG-based slicing [14], will create a dependence graph for the program that can be used to quickly identify these parts.

The above holds for intra-procedural programs. For programs with function calls, mutating a call may also affect the program flow significantly so these mutations must then also be considered even if not in the slice.

Currently, we are implementing a prototype of our framework for *targeted mutation*. The framework is mainly comprised of (i) a Static Analysis Tool (e.g., SWEET [15]) – identifying the parts of the code with a strong influence on the execution time, and (ii) a Mutation Tool (e.g., Proteum [16]) – injecting changes in the identified parts of the code. The framework basically explores the relationship between mutants and control flow; therefore, it is based on a pattern matching between class of control flow (e.g., loops where updates over program variables are linear expressions) and mutators (e.g., conditionals boundary mutators).

III. Case Study: Generation of mutants for testing execution time (like WCET)

In the following, we describe an experimental procedure that we devised in order to (i) illustrate how *targeted mutation* may influence effectiveness and efficiency of temporal testing, and (ii) provide recommendations and guidelines, which stem from our experience in the field of static WCET analysis and mutation testing. Specifically, the experimental procedure aims to confirm or refute the hypothesis that mutation analysis encourages design of effective test suites for WCET estimation and that targeted mutation improves efficiency of this process significantly while maintaining effectiveness. We focus on the following two research questions.

- RQ1: How effective is targeted mutation for WCET estimation?
- RQ2: How efficient is targeted mutation compared to traditional mutation?

To answer RQ1, we propose to compare the WCET estimations given by three different test suites designed by; (i) random, (ii) traditional mutation analysis, and (iii) targeted mutation analysis. Our hypothesis is that mutation analysis gives better WCET estimation than random and that the traditional and targeted approaches are equally effective.

RQ2 focuses on how much we gain by targeted mutation. We propose to compare traditional mutation analysis to targeted mutation analysis with respect to; (i) size of test suite, (ii) number of mutants (generated, stillborn and equivalent),

(iii) time to generate the mutants, and (iv) time to run the test suites against the mutants. Our hypothesis is that targeted mutation is significantly more efficient than traditional even when taking the time to perform the static analysis and slicing into account. The size of test suites and the number of mutants are indicators of the cost for the approach. A comparison of the time spent in executing the test suite against the generated mutants can be very interesting if the two sets of mutants (targeted and traditional) are expected to be different in terms of how difficult it is to detect them.

To perform WCET estimates, there is the need to make the right trade-off between the number of executed test cases and the accuracy of the WCET estimates, i.e., the expected quality (confidence levels/accuracy) of the WCET depends on the number of executed test cases. The experimental procedure consists of the following high-level steps:

1) Generation of mutants for original program.
2) Execution of the test suite against the generated mutants. To answer our research questions, different metrics can be use to compare the results with and without slicing; therefore, examples of comparison steps are (i) the annotation of the time spent in the test case execution and the number of executions; (ii) the annotation of the number of killed mutants by time-out and time spent to kill them; and (ii) the WCET estimation.
3) If the mutants were executed less than three times, go to Step 2.
4) Identification of program parts relevant for execution time by using program slicing.
5) Execution of the test suite against only the mutants (*targeted mutations*) related to relevant parts. Similarly as Step 2, different measures can be collected. Tests that no longer contribute to mutation score are removed.
6) If the mutants were executed less than three times, go to Step 5.

Discussion. The ideas presented in this paper are, to the best of our knowledge, a first attempt to investigate the combination of static and mutation analysis for testing non-functional properties. We believe *targeted mutation* to be a promising approach, which may outperform existing approaches and resolve some of their major drawbacks. Even though interest in the exploration of static analysis techniques as a means to estimate non-functional properties (e.g., WCET) has grown rapidly, there are still big research challenges to be addressed. Existing approaches often make simplifying assumptions (e.g., to estimate the number of loop iterations and/or recursion depth) that may affect the WCET estimation. We believe that the cost of mutation testing can be effectively addressed by using static analysis techniques to reduce the search space. Program slicing has already been proven to be an efficient strategy for search space reduction in the context of test data generation [17].

IV. Related Work

WCET analysis by testing is usually referred to as "measurement-based WCET analysis". Various methods for test case generation have been attempted, like path-based testing [18], and search-based testing, using a combination of evolutionary algorithms and model-checking [19] or using multi-objective optimization [20]. *Measurement-Based Probabilistic Timing Analysis* (MBPTA) applies extreme value theory to estimate the probability of the WCET estimate being overrun [21]. Hybrid methods combine measurements with elements of static WCET analysis [22], [23].

There are a few mutation-based approaches to test time properties reported [7], [24], [25] but these are model-based approaches to test at a (sub-)system level. In such models, assumptions about e.g., the inter-arrival time and execution time of individual tasks are used. Mutators are then applied to the modeled constraints on time and order in the task set. In contrast, our work focuses on mutating the source code for individual tasks and thus, does not rely on execution time assumptions.

There are many papers suggesting different approaches to reduce the number of mutants to speed up the analysis. Several approaches select a subset of available mutators [26], [27], [28]. In contrast to to these approaches, we propose to apply the mutators only to certain slices of the software, which are relevant to the non-functional property of interest that we aim to test.

A recent and promising approach to reduce the number of mutants focuses on identifying subsumption relations between the mutants [29], [30], [31], [32]. A mutant m_i that is subsumed by another mutant m_j can be removed since any test that detects m_j is guaranteed to detect m_i. Our proposed approach is clearly related to this research since we avoid generating irrelevant mutants. However, the mutants we avoid in our analysis are irrelevant from a temporal perspective and not necessarily redundant in terms of their functional behavior.

V. Conclusions and Future Work

We introduced *targeted mutation* as a paradigm for mutation testing of non-functional properties, where the mutations are focused to the parts of the code that are likely to have a significant impact on the property in question. We applied this methodology to WCET analysis, arguing that the parts of the code that can affect the program flow provide the best mutation targets. These parts are easily identified by program slicing. We also outline an experimental evaluation of the approach.

This paper presents the main idea of the method of targeted mutation for testing of non-functional properties, and can as such be considered as a position paper. We plan two follow-up studies, outlined below.

1) We will evaluate and refine the targeted mutation approach. The first step is to run the experiment outlined in Section III. This experiment will evaluate our proposed technique in terms of effectiveness and efficiency for estimating WCET. During the experiment, we collect data on the mutants (number of generated, stillborn, equivalent, useful). The second step in our work will be to analyze this data to identify differences that can be used to reduce the mutation effort further, e.g., certain changes that never create useful mutants. This information

will help us propose reduced mutation with focus on WCET estimation. Finally, we will identify mutations that current mutators do not cover and which can be useful for WCET estimation. Each step will be evaluated empirically.

2) We also plan to apply targeted mutation to testing of other resource consumption properties. Just as we can identify parts of the code that are associated with execution time, it should be possible to identify parts of the code that are associated with e.g., memory or energy consumption. Conducting an experiment on targeted mutation that focuses on energy or memory would give an indication of the value of our approach in a more general context than just execution time.

VI. Acknowledgments

This work was partly funded by The Knowledge Foundation (KKS) through the project 20130085: Testing of Critical System Characteristics (TOCSYC).

References

[1] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on test data selection: Help for the practicing programmer," *Computer*, vol. 11, no. 4, pp. 34–41, 1978.

[2] P. Ammann and J. Offutt, *Introduction to software testing*. Cambridge University Press, 2008.

[3] K. N. King and A. J. Offutt, "A Fortran language system for mutation-based software testing," *Software Practice & Experience*, vol. 21, no. 7, pp. 685–718, jun 1991.

[4] Y. S. Ma, Y. R. Kwon, and J. Offutt, "Inter-class mutation operators for Java," in *Proceedings of the 13th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE Computer Society, 2002, pp. 352–363.

[5] H. Agrawal, R. A. Demillo, B. Hathaway, W. Hsu, W. Hsu, E. W. Krauser, R. J. Martin, A. P. Mathur, and E. H. Spafford, "Design of mutant operators for the C programming language," Software Engineering Research Center, Purdue University, Tech. Rep. SERC-TR-41-P, March 1989.

[6] Y. Le Traon, T. Mouelhi, and B. Baudry, "Testing security policies: going beyond functional testing," in *The 18th IEEE International Symposium on Software Reliability (ISSRE'07)*. IEEE, 2007, pp. 93–102.

[7] R. Nilsson, J. Offutt, and J. Mellin, "Test case generation for mutation-based testing of timeliness," *Electronic Notes in Theoretical Computer Science*, vol. 164, no. 4, pp. 97–114, 2006.

[8] R. M. Hierons and M. G. Merayo, "Mutation testing from probabilistic and stochastic finite state machine," *Journal of Systems and Software*, vol. 82, no. 11, pp. 1804–1818, 2009.

[9] B. Lindström, S. F. Andler, J. Offutt, P. Pettersson, and D. Sundmark, "Mutating aspect-oriented models to test cross-cutting concerns," in *IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2015.

[10] B. Kurtz, P. Ammann, J. Offutt, and M. Kurz, "Are we there yet? how redundant and equivalent mutants affect determination of test completeness," in *IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2016.

[11] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution time problem — overview of methods and survey of tools," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 3, pp. 1–53, 2008.

[12] M. Weiser, "Program Slicing," *IEEE Transactions on Software Engineering*, vol. SE-10, no. 4, pp. 352–357, Jul. 1984.

[13] R. Just, D. Jalali, L. Inozemtseva, M. D. Ernst, R. Holmes, and G. Fraser, "Are mutants a valid substitute for real faults in software testing?" in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22)*. ACM, 2014, pp. 654–665.

[14] J. Ferrante, K. J. Ottenstein, and J. D. Warren, "The program dependence graph and its use in optimization," *ACM Transactions on Programming Languages and Systems*, vol. 9, no. 3, pp. 319–349, Jul. 1987.

[15] B. Lisper, "Sweet a tool for wcet flow analysis," in *6th Intern. Symp. On Leveraging Applications of Formal Methods, Verification and Validation*. Springer-Verlag, 2014, pp. 482–485.

[16] J. C. Maldonado, M. E. Delamaro, S. C. P. F. Fabbri, A. d. S. Simão, T. Sugeta, A. M. R. Vincenzi, and P. C. Masiero, "Mutation testing for the new century," 2001, ch. Proteum: A Family of Tools to Support Specification and Program Testing Based on Mutation, pp. 113–116.

[17] P. McMinn, M. Harman, K. Lakhotia, Y. Hassoun, and J. Wegener, "Input Domain Reduction through Irrelevant Variable Removal and Its Effect on Local, Global, and Hybrid Search-Based Structural Test Data Generation," *IEEE Trans. Software Eng.*, vol. 38, no. 2, pp. 453–477, 2012.

[18] N. Williams and M. Roger, "Test generation strategies to measure worst-case execution time," in *Proc. 2009 ICSE Workshop on Automation of Software Test*, May 2009, pp. 88–96.

[19] I. Wenzel, R. Kirner, B. Rieder, and P. Puschner, "Measurement-based worst-case execution time analysis," in *Proc. Third IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS'05)*, May 2005, pp. 7–10.

[20] I. Bate and U. Khan, "WCET analysis of modern processors using multi-criteria optimisation," *Empirical Software Engineering*, vol. 16, no. 1, pp. 5–28, 2011. [Online]. Available: http://dx.doi.org/10.1007/s10664-010-9133-9

[21] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Q. ones, and F. J. Cazorla, "Measurement-based probabilistic timing analysis for multi-path programs," in *Proc. 24th Euromicro Conference on Real-Time Systems (ECRTS'12)*, Jul. 2012, pp. 91–101.

[22] G. Bernat, A. Colin, and S. M. Petters, "WCET analysis of probabilistic hard real-time systems," in *Proc. 23rd IEEE Real-Time Systems Symposium (RTSS'02)*. Austin, TX: IEEE Computer Society, Dec. 2002, pp. 279–288.

[23] B. Lisper and M. Santos, "Model identification for WCET analysis," in *Proc. 15th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'09)*. San Francisco, CA: IEEE Computer Society, Apr. 2009, pp. 55–64.

[24] B. K. Aichernig, F. Lorber, and D. Ničković, "Time for mutants — model-based mutation testing with timed automata," in *International Conference on Tests and Proofs*. Springer, 2013, pp. 20–38.

[25] M. S. AbouTrab, S. Counsell, and R. M. Hierons, "Specification mutation analysis for validating timed testing approaches based on timed automata," in *2012 IEEE 36th Annual Computer Software and Applications Conference*. IEEE, 2012, pp. 660–669.

[26] R. Untch, "On reduced neighborhood mutation analysis using a single mutagenic operator," in *ACM Southeast Regional Conference*, Clemson SC, March 2009, pp. 19–21.

[27] L. Deng, J. Offutt, and N. Li, "Empirical evaluation of the statement deletion mutation operator," in *6th IEEE International Conference on Software Testing, Verification and Validation (ICST 2013)*, Luxembourg, March 2013.

[28] M. E. Delamaro, L. Deng, V. H. S. Durelli, N. Li, and J. Offutt, "Experimental evaluation of SDL and one-op mutation for C," in *7th IEEE International Conference on Software Testing, Verification and Validation (ICST 2014)*, Cleveland, Ohio USA, March 2014.

[29] R. Just, G. M. Kapfhammer, and F. Schweiggert, "Do redundant mutants affect the effectiveness and efficiency of mutation analysis?" in *Eighth IEEE Workshop on Mutation Analysis (Mutation 2012)*, Montreal, Canada, April 2012.

[30] G. Kaminski, P. Ammann, and J. Offutt, "Improving logic-based testing," *Journal of Systems and Software, Elsevier*, vol. 86, pp. 2002–2012, August 2013.

[31] P. Ammann, M. E. Delamaro, and J. Offutt, "Establishing theoretical minimal sets of mutants," in *7th IEEE International Conference on Software Testing, Verification and Validation (ICST 2014)*, Cleveland, Ohio, March 2014.

[32] B. Kurtz, P. Ammann, M. E. Delamaro, J. Offutt, and L. Deng, "Mutant subsumption graphs," in *Tenth IEE Workshop on Mutation Analysis (Mutation 2014)*, Cleveland Ohio, USA, March 2014.