Application of Search-Based Software Engineering Methodologies for Test Suite Optimization and Evolution in Mission Critical Mobile Application Development

Andreas Schuler*
University of Applied Sciences Upper Austria
Softwarepark 13
Hagenberg, Austria 4232
andreas.schuler@fh-hagenberg.at

ABSTRACT

The demand for high quality mobile applications is constantly rising, especially in mission critical settings. Thus, new software engineering methodologies are needed in order to ensure the desired quality of an application. The research presented proposes a quality assurance methodology for mobile applications through test automation by optimizing test suites. The desired goal is to find a minimal test suite while maintaining efficiency and reducing execution cost. Furthermore to avoid invalidating an optimized test suite as the system under test evolves, the approach further proposes to extract patterns from the applied changes to an application. The evaluation plan comprises a combination of an empirical and an industrial case study based on open source projects and an industrial project in the healthcare domain. It is expected that the presented approach supports the testing process on mobile application platforms.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**; *Software evolution*; *Search-based software engineering*;

KEYWORDS

test automation, test suite optimization, mobile application development, multi-objective optimization

ACM Reference format:
Andreas Schuler. 2017. Application of Search-Based Software Engineering Methodologies for Test Suite Optimization and Evolution in Mission Critical Mobile Application Development. In *Proceedings of 2017 11th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Paderborn, Germany, September 4–8, 2017 (ESEC/FSE'17),* 4 pages.
https://doi.org/10.1145/3106237.3119876

*Advisor: Univ.-Prof. Mag. Dr. Gabriele Anderst-Kotsis (gabriele.kotsis@jku.at)

ESEC/FSE'17, September 4–8, 2017, Paderborn, Germany

ACM ISBN 978-1-4503-5105-8/17/09…$15.00
https://doi.org/10.1145/3106237.3119876

1 INTRODUCTION

The development of applications for modern mobile platforms is a challenging task due to the heterogeneous nature of said platforms. Different device vendors, as well as different operating systems and versions, contribute to this problem. Given these premises together with the fact that mobile applications are nowadays considered to be used in critical domains -like healthcare, government or in payment systems- mobile application development and associated testing becomes more complex, time consuming and costly to ensure the quality of a released product [24]. Still there is a raising demand for high quality mobile applications, especially when used in mission critical settings [16, 18]. Thus, Muccini et al. [18] conclude, that new software engineering approaches are required in order to provide thorough testing of mission critical mobile applications.

According to Muccini et al. [18] the aforementioned challenges arise from the fact that mobile computing differs from traditional computing in the following areas: *limited resources, security and vulnerability, performance and reliability* as well as *finite energy power*. It can be further concluded that test automation in mobile computing is a promising field of research to identify new approaches and methodologies applicable for a mobile computing environment [16, 24]. Although as described by Zein et al. [24] and Muccini et al. [18] there are a lot of different test automation methodologies available, e.g. model-based testing and search-based testing, the outlined differences in mobile computing require further research.

The proposed thesis aims to research improving the test suite quality in mission critical mobile application development by leveraging multi-objective optimization techniques from the field of *Search-based Software Engineering (SBSE)* in order to minimize testing time while reaching an optimal coverage. Consequently, the thesis aims to provide answers to following research questions:

RQ-1: How can search-based techniques be applied to the problem of test suite optimization in mobile application development as part of the software development life cycle, in order to maximize test coverage for a given test criteria (e.g. memory-usage, energy-consumption, etc.) while minimizing execution time and cost?

RQ-2: What effects result from changes, additions or modifications to a system under test for an existing test suite and what effects does it have on the optimization?

RQ-3: Is it possible to derive features/patterns specific to a given test criteria for a software increment and can such an approach be used to predict validity of an optimised test suite given the advancement of the system under test?

The remainder of this work outlines previous work on test suite optimization in mobile computing in section 2. In sections 4 to 6 the proposed approach, the expected contribution together with the results achieved so far is presented. Section 7 further describes how the proposed work will be evaluated. Finally, section 8 will conclude with some final remarks.

2 BACKGROUND

Test suite optimization is a field of software testing that deals with optimising the test cases contained in a test suite, in order to achieve some coverage criteria of a *System Under Test (SUT)*. Given a test suite that was either created manually or generated through a model-based, search-based or by any other means of automatic test generation approach (e.g. [2, 7, 13, 14, 17]), said test suite has to evolve accordingly to the SUT. Although this is often not the case, as mentioned by Farooq and Lam [6], test suites often contain redundant test cases and they remain undetected until test execution (see Fig. 1). This leads to the fact that a test suite with redundant test cases takes longer to complete and consumes more resources [1, 6, 8, 15]. Test suite optimization deals with this aspect, by removing, changing or adding new test cases to a test suite in order to cope with the evolution of a SUT. In their survey Yoo and Harman [23] further describe that test suite optimization can be categorised in different techniques that are applied to a given test suite:

- ***Test Suite Minimisation:*** In test suite minimisation the goal is to reduce the size of a given test suite by removing obsolete or redundant test case, in order to find a minimal hitting set, which is described as the minimal set of test cases able to satisfy a desired test criteria.
- ***Test Case Selection:*** While test suite minimisation only strives to reduce the size of a given test suite, test suite selection is by some means aware of changes in the underlying SUT. Thus it requires a static white-box analysis of the program code [23].
- ***Test Case Prioritisation:*** The idea behind test case prioritisation is to find a optimal permutation of the sequence of test cases in order to identify possible faults at an early stage in the sequence of executed test cases.

In order to solve a test suite optimization problem for one of the different techniques listed above, several approaches have been proposed in literature consisting of *linear programming* [9], *greedy algorithms* [9] as well as *evolutionary computation* based on metaheuristic optimization algorithms [4, 6, 21].

Independent of the applied algorithm in any case the first step in test suite optimization is to find a proper problem representation. For example Farooq and Lam [6] propose a metaheuristic approach for test suite minimisation using a binary coding scheme, where an individual is encoded by assigning each test case in a test suite either a 1 or a 0 value. Hence the number of 1's in a test suite represent its size. The second step in test suite optimization consists of the definition of a fitness function or in more general cases an objective function, which allows to determine if a specific individual test suite is superior to another one in terms of a desired criteria. Common used criteria are *code coverage*, *execution time*, *execution cost*, *fault detection rate* [23].

In what follows, a selection of current research approaches regarding test suite optimization is presented. The list is far from complete, instead it is a brief overview of available work in this field of research.

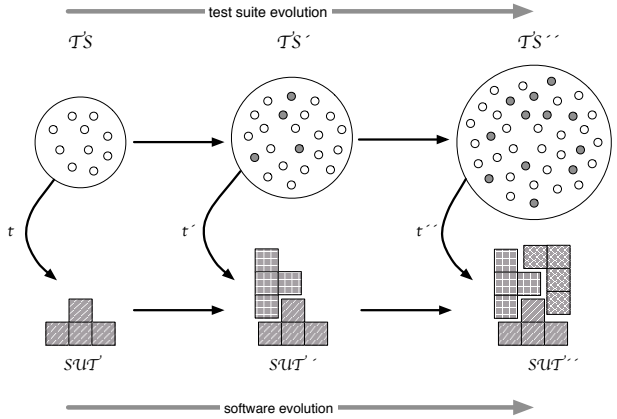

Figure 1: The evolution of a test suite TS alongside the evolution of system under test SUT and the tests t being executed. As the application matures, the test suite changes accordingly, however some of the test case become redundant, denoted by the dark coloured test cases as part of a test suite.

3 RELATED WORK

The approach described by Hemmati et al. [8] strives to reduce the cost of model-based testing trough what is called test case diversity. To achieve the described goal, they developed a similarity measure, that aims to maximize diversity in test cases. Anwar and Ahsan [1] present a regression test suite optimization technique using Fuzzy Logic. Mayo and Spacey [15] propose a novel framework that aims to predict regression test failure. The basic idea behind the approach is to capture runtime behaviour based on a performance analysis of each test case executed in a regression test suite. De Souza et al. [4] formulate a test case selection problem as a constrained search-based optimization task. The fitness function utilised, strives to maximise requirements coverage in combination with the execution time for a specific test case. Prasad et al. [21] discuss the process of test suite minimization in regression testing using a combination of evolutionary computation and greedy algorithms. Jeyaprakash and Alagarsamy [10] apply a variant of a genetic algorithm called *Non-dominated Sorting Genetic Algorithm (NSGAII)* to the problem of test suite minimisation. Jabbarvand et al. [9] introduce an energy-aware test suite minimisation technique specifically addressed to Android applications applying both *Integer Linear Programming* and *Greedy Algorithms* to solve the optimization problem at hand.

4 APPROACH

The first step in achieving the research objective comprises a systematic literature review following the guidelines from Kitchenham and Charters [11]. The review serves as an initial step to conceive the current state of the art of multi-objective test suite optimization in mobile application development with special focus on its application in mission critical applications. Furthermore relevant

test criteria to be used as part of the multi-objective optimization problem have to be identified, two of which already stated are *energy-consumption* and *memory-usage* which are relevant for this thesis.

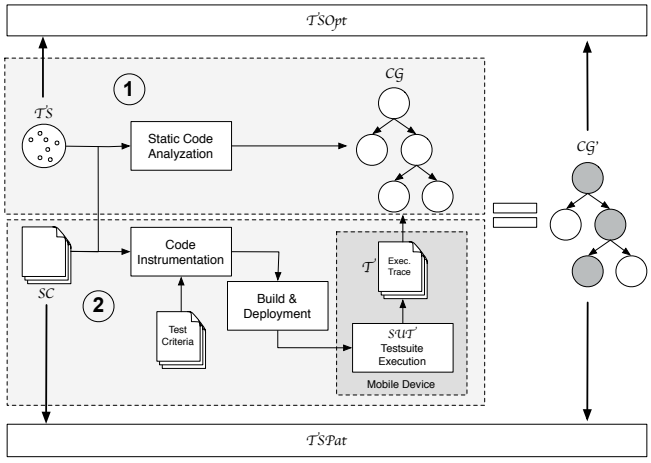

Figure 2: (1) A call graph *CG* is build using static analysation based on a test suite *TS* together with the associated source code *SC*. (2) The *SC* is instrumented with respect to a defined test criteria, deployed and executed. The resulting execution trace is further used to enrich the call graph *CG′*. (3) Finally the *CG′* is used to optimize *TSOpt* the test suite and derive features *TSPat* relevant to the test criteria.

Based on the problem definition and the initially stated research questions in section 1, a first conceptual architecture is presented in Fig. 2. As depicted the approach requires source code access as well as access to a mobile device for test execution. The latter is needed as specific test criteria, e.g. *energy consumption*, can only be addresses on a physical device [9]. Thus the presented approach is to be categorised as a form of white-box testing and corresponds to the test case selection technique defined in section 2 [23]. Given the proposed architecture the approach is further described in the following sections.

4.1 Extraction and Instrumentation

Starting from a test suite together with the source code representation of a SUT, a call graph *CG* (see Fig. 2 step (1)) is to be created using static code inspection. The call graph is further used to trace method call stacks from each test case in the given test suite *TS*. Yoo and Harman [23] describe this form of test case selection as a control-flow analysis approach. To enrich the generated call graph with information relevant to a specific test criteria, the available source code is instrumented accordingly. This step involves different instrumentation approaches, as they are closely connected to the test criteria one is interested in. This means for example in case of the test criteria energy consumption, the code instrumentation is somewhat different than in case of memory usage, where for the latter, the current memory levels before and after a specific method execution is traced. Whereas for energy consumption the approach involves sampling of test suites during program execution and collecting respective energy levels. The collected energy traces are correlated with the extracted call graph and thus result in the enriched call graph *CG′* [12]. Challenges that have to be addressed in course of the thesis involve the implications of *garbage collection*, *thread context switching* and *hardware tail energy* on measured energy levels. Current research has already described possible approaches on how to deal with these aspects when mapping energy levels to individual source code lines [12].

4.2 Test Suite Optimization

In order to provide answer to RQ-1, the enriched *CG′* is used as input for a multi-objective optimization algorithm, striving to seek a minimal selection of test cases, that allow for optimal coverage of the enriched *CG′* with respect to the desired test criteria. For example, in case of energy consumption, the optimization problem to be solved by the algorithm, is to find a minimal selection of test cases in a given test suite that cause the highest energy foot print. Preliminary work on applying energy-aware optimization on test suites are described by Jabbarvand et al. [9]. As for the algorithm, the presented approach is to rely on population based algorithms, e.g. Genetic Algorithms or Evolution Strategy [3].

4.3 Test Criteria Feature Extraction

Since the enriched call graph *CG′* is tied to the state of the source code it was created from, manipulations in form of additions result in an invalid call graph. This leads to the fact, that with every software increment, the enriched call-graph *CG′* would have to be recreated. This means that all test cases in the affected test suite need to be executed on a physical device again.

Thus, the presented approach proposes an additional step in the optimization process. A feature extraction process is to be applied on the enriched call graph and the associated source code. The feature extraction process is responsible to determine patterns and features relevant to the selected test criteria. E.g. Vásquez et al. [22] describe code patterns that are subject to high energy footprint on android devices. As the patterns are closely tied to the test criteria, part of answering RQ-3 will consist of researching available patterns with respect to the test criteria.

As a result, when the SUT is changed, either by adding or manipulating the code areas relevant to the given test criteria, these features, based on the changes, are expected to be able to predict a quality value for the respective test case and thus give insight to RQ-3. Furthermore, this will allow to determine if code and or a test case that have been added will yield better results compared to a current available solution.

5 RESULTS ACHIEVED SO FAR

First experiments with multi-objective test suite optimization have started recently by applying a genetic algorithm on an existing test suite generated using the randoop framework in order to reduce the size and execution time of a generated test suite while maintaining coverage [20]. Furthermore, static analysis techniques were utilised to retrieve call graphs from sample source code in Java.

A review protocol for an upcoming literature review is currently prepared, identifying evidence in the area of multi-objective test suite optimization in mobile application development. Additionally 3 open source projects have been selected from the F-Droid Android

OSS [5] repository It is planned to use the selected open source projects for a case study in which the aforementioned approach is utilised in order to empirically evaluate the approach.

6 EXPECTED CONTRIBUTIONS

Given the aforementioned research approach the expected contributions of the thesis are:

CO-1: A case study illustrating the approach being able to reduce test suite size given a defined test criteria, while preserving coverage. A first set of relevant criteria for this thesis consist of (*A*) *memory-usage* and (*B*) *energy-consumption*.

CO-2: A new multi-objective test suite optimisation methodology for mobile applications.

CO-3: A library of features and patterns associated with the investigated test criteria (*A*) and (*B*) to support the test case selection process as the SUT evolves.

CO-4: The foundation for a framework, able to cope with test suite optimization problems in mobile computing, offering a set of different algorithms and optimization techniques as a basis for subsequent research.

7 EVALUATION PLAN

The evaluation plan of the thesis comprises both, an evaluation based on open source projects as well as in industrial project. To answer RQ-1 and RQ-2 open source applications from the F-Droid [5] repository will be used. The first goal is to evaluate the effectiveness of the approach by optimizing available test suites and compare the results with the current state of testing for respective projects. The expected result is, that the presented approach is able to reduce time and cost for testing, as existing test suites are optimised along a defined test criteria. Subsequently the test criteria feature extraction will be evaluated by comparing specific software increments from respective open source applications. This allows to determine, if the proposed feature extraction technique is able to outperform standalone test suite optimization, thus answering RQ-3.

As an addition a comparative study state of the art test suite optimization approaches based on an existing benchmark suite is planned. However as described by Nagappan and Shihab [19] currently most of the available test automation approaches geared towards at mobile applications work on app binaries due to the fact the researchers often don't have access to source code.

In case of the industrial case study first discussions on how to apply the presented approach have started. While the goal of the industrial case study is to determine the effectiveness accompanying the development progress and utilizing the optimization of a mission critical addressed to healthcare.

8 CONCLUSION AND OUTLOOK

This paper proposes a thesis in the field of test suite optimization. Backed by a selection of previous work in this field the problem statement as well as the research questions are presented. Furthermore an approach to apply multi-objective test optimization algorithms in connection with test criteria driven feature extraction mechanism is proposed, enabling to estimate quality levels of test cases as a system under test evolves. Subsequently the planned work is described on the way to fulfill the stipulated goals set for the doctoral thesis. An overview of the expected outcome is presented together with a first evaluation plan, describing how the research questions are to be answered.

REFERENCES

[1] Zeeshan Anwar and Ali Ahsan. 2013. Multi-objective regression test suite optimization with Fuzzy logic. In *2013 16th International Multi Topic Conference (INMIC)*. IEEE, 95–100.

[2] P N Boghdady, N Badr, M Hashem, and M F Tolba. 2011. Test case generation and test data extraction techniques. *Inter J Electr Comput Sci* (2011).

[3] M Cavazzuti. 2012. Optimization methods: from theory to design scientific and technological aspects in mechanics. AIP.

[4] Luciano S De Souza, Ricardo B C PrudêNcio, Flavia De A Barros, and Eduardo H Da S Aranha. 2013. Search based constrained test case selection using execution effort. *Expert Systems with Applications: An International Journal* 40, 12 (Sept. 2013), 4887–4896.

[5] F-Droid Limited. 2017. F-Droid Android Open Source Repository. (2017). https://f-droid.org

[6] Usman Farooq and C P Lam. 2015. Evolving the Quality of a Model Based Test Suite. In *2009 International Conference on Software Testing, Verification, and Validation Workshops*. IEEE, 141–149.

[7] Gordon Fraser, Gordon Fraser, and Andrea Arcuri. 2013. Whole Test Suite Generation. *IEEE Transactions on Software Engineering* 39, 2 (2013), 276–291.

[8] Hadi Hemmati, Lionel Briand, Andrea Arcuri, and Shaukat Ali. 2010. *An enhanced test case selection approach for model-based testing: an industrial case study*. ACM, New York, New York, USA.

[9] Reyhaneh Jabbarvand, Alireza Sadeghi, Hamid Bagheri, and Sam Malek. 2016. Energy-aware test-suite minimization for Android apps. ACM Press, 425–436.

[10] Srividhya Jeyaprakash and K Alagarsamy. 2015. A Distinctive Genetic Approach for Test-Suite Optimization. *Procedia Computer Science* 62 (2015), 427–434.

[11] Barbara Kitchenham and Stuart Charters. 2007. *Guidelines for performing systematic literature reviews in software engineering*. Technical Report. Technical report, EBSE Technical Report EBSE-2007-01.

[12] Ding Li, Shuai Hao, William G. J. Halfond, and Ramesh Govindan. 2013. Calculating Source Line Level Energy Information for Android Applications. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis (ISSTA 2013)*. ACM, New York, NY, USA, 78–89. https://doi.org/10.1145/2483760.2483780

[13] Riyadh Mahmood, Nariman Mirzaei, and Sam Malek. 2014. EvoDroid - segmented evolutionary testing of Android apps. *SIGSOFT FSE* (2014), 599–609.

[14] Ke Mao, Mark Harman, and Yue Jia. 2016. Sapienz: multi-objective automated testing for Android applications. ACM Press, 94–105.

[15] Michael Mayo and Simon Spacey. 2013. Predicting Regression Test Failures Using Genetic Algorithm-Selected Dynamic Performance Analysis Metrics. In *Search Based Software Engineering*. Springer Berlin Heidelberg, Berlin, Heidelberg, 158–171.

[16] A Méndez. 2015. Automated testing of mobile applications: a systematic map and review. *Proceedings of the 10th International Conference on Web Information Systems and Technologies* (2015).

[17] S M Mohi-Aldeen and S Deris. 2013. Comparative Evaluation of Automatic Test Case Generation Methods. *… of Technology* (2013).

[18] Henry Muccini, Antonio Di Francesco, and Patrizio Esposito. 2012. Software testing of mobile applications: Challenges and future research directions. In *2012 7th International Workshop on Automation of Software Test (AST)*. IEEE, 29–35.

[19] Meiyappan Nagappan and Emad Shihab. 2016. Future Trends in Software Engineering Research for Mobile Apps. *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)* (2016).

[20] Carlos Pacheco and Michael D. Ernst. 2007. Randoop: Feedback-directed Random Testing for Java. In *OOPSLA 2007 Companion, Montreal, Canada*. ACM.

[21] Dhanyamraju S U M Prasad, Simy Chacko, Satya Sai Prakash Kanakadandi, and Gopi Krishna Durbhaka. 2014. Automated Regression Test Suite Optimization Based on Heuristics. In *ICAIET '14: Proceedings of the 2014 4th International Conference on Artificial Intelligence with Applications in Engineering and Technology*. IEEE Computer Society, 48–53.

[22] Mario Linares Vásquez, Gabriele Bavota, Carlos Bernal-Cárdenas, Rocco Oliveto, Massimiliano Di Penta, and Denys Poshyvanyk. 2014. Mining energy-greedy API usage patterns in Android apps - an empirical study. *MSR* (2014).

[23] Shin Yoo and Mark Harman. 2012. Regression testing minimization, selection and prioritization - a survey. *Softw. Test., Verif. Reliab.* 22, 2 (2012), 67–120.

[24] S Zein, N Salleh, and J Grundy. 2016. A systematic mapping study of mobile application testing techniques. *Journal of Systems and Software* 117 (2016), 334–356.