TEST SUITE REDUCTION IN ANDROID APPLICATION USING REFACTORING

MARYAM TEMITAYO AHMED

A thesis submitted in
fulfillment of the requirement for the award of the
Doctor of Philosophy in Information Technology

Faculty of Computer Science and Information Technology
Universiti Tun Hussein Onn Malaysia

JULY 2016

DEDICATION

I dedicate this thesis to Almighty Allah, the cherisher and the sustainer who has made my journey all worth it. And to my lovely parent, thanks for always being there for me. To my husband, I appreciate your love and support. And for the two precious gifts Allah blessed me with during the course of study, I love you so much.

ACKNOWLEDGEMENT

All praises, thanks and adorations are due to Almighty Allah. I am more than grateful to Him for seeing me through this course. I am deeply indebted to my supervisor Prof. Dr. Rosziati Ibrahim whose help, stimulating suggestions, useful critiques and encouragement helped me in all the times of study and analysis of the pre and post research period. My heart felt appreciation to my co-supervisor, Dr. Noraini Ibrahim, your support, patience and assistance is well appreciated. I am indeed grateful. My appreciation to the members of staff of the Faculty of Computer Science and Information Technology (FSKTM) and Centre of Graduate Studies (CGS) of Universiti Tun Hussein Onn Malaysia (UTHM) for providing a comfortable environment that enhanced my study. I am deeply indebted to the Office of Research, Innovation, Commercialization and Consultancy Management (ORICC) for sponsoring this research under the vot. 1256. I would like to express my gratitude to Dr. Rasheeda Olanrewaju, Mrs. Remi Ahmed, Mr Abiola Balogun, Aisha Ishaq, Maryam Salihu, Hauwa Atiku, Saima Anwar Lashari, my neighbours in Malewar and my colleagues. You all have made the journey worthwhile. Special thanks goes to my friends both home and abroad. Thanks for being part of my story. To my family; uncles, aunties, cousins and in-laws, thank you all for your moral support and words of encouragement. You are well appreciated. I am without any doubt indebted to my siblings. Thanks for touching my life in a special way. My utmost gratitude goes to my parent and mother-in-law. You are indeed a blessing to me. Thank you for believing in me. And to all my lovely nieces; thanks for being my source of inspiration. My heartfelt appreciation goes to my darling husband and lovely sons for their support and encouragement during the course of this research. Your support and co-operation can never be overestimated.

ABSTRACT

The vast increase in demand for android applications has made android application testing inevitable. The android open source feature has led to developers of unknown level of expertise developing the application, thus raising concerns on quality issues. Currently, android applications are found lagging in the area of testing. Test case generation is the most important and challenging area of software testing. Test cases tend to be large in number as redundant test cases are generated due to the presence of code smells in the software. Code smells are unnecessary codes, as a result of poor design or implementation. Several approaches have been proposed in the past by both academy and industrial researchers to tackle the high number of generated test cases in android applications, including test case minimization and prioritization technique. Nonetheless, these approaches are reactive rather than proactive. The technique used in this study is to apply code refactoring before test case generation to avoid redundant test cases from being generated. To achieve this, the detection of smells was done, followed by refactoring of detected smells. Test cases were then generated from the refactored code. More explicitly, this research presents three rules for detection of three code smells; lazy class, small method and duplicate, and three rules for their refactoring. The rules were implemented in a tool named DART (Detection and Refactoring Tool) to refactor the android source code for the reduction of test cases. The resultant source code is compared with the original source code by generating a number of branches, branch coverage and complexity using Clover. The results of this research show a reduction of about 7.7% in the cyclomatic complexity of the source code, while increasing the branch coverage with up to 9.2% increment. Also, there is a 28% reduction in the number of test cases generated. These show that refactoring can be used to reduce redundant test cases.

ABSTRAK

Permintaan yang meningkat hebat terhadap aplikasi android telah menyebabkan pengujian android sesuatu yang tidak dapat dielakkan. Ini kerana, ciri sumber terbuka android mengundang pembangun sistem yang tidak diketahui kepakarannya turut membangunkan aplikasi ini dan seterusnya menimbulkan kebimbangan terhadap isu kualiti. Pada masa ini, aplikasi android masih dilihat ketinggalan dalam bidang pengujian. Penghasilan kes ujian ialah bidang pengujian perisian yang paling penting dan mencabar. Penghasilan bilangan kes ujian yang besar berpunca daripada penghasilan kes ujian lewah yang terhasil daripada kewujudan hiduan kod atau *code smells* dalam perisian. Hiduan kod ialah kod yang tidak perlu yang berpunca daripada reka bentuk dan pelaksanaan yang tidak baik. Beberapa pendekatan telah dicadangkan oleh para penyelidik akademik dan industri bagi menangani kebanjiran kes ujian yang dihasilkan dalam aplikasi android, termasuk teknik peminimuman dan pengutamaan kes ujian. Walau bagaimanapun, pendekatan ini lebih bersifat reaktif dan bukan proaktif, lantas menyebabkan pembaziran masa dan usaha. Teknik yang dicadangkan adalah dengan menggunakan pemfaktoran semula kod sebelum kes ujian dihasilkan untuk mengelakkan kes ujian dihasilkan secara lewah. Untuk mencapai perkara ini, pengesanan hiduan dilakukan, diikuti dengan pemfaktoran semula kod hiduan yang dikesan. Kes ujian kemudiannya dijana daripada kod terfaktor semula tersebut. Secara lebih jelasnya, kajian ini membentangkan tiga peraturan untuk mengenal pasti tiga jenis hiduan kod, iaitu *lazy class, small method* dan *duplicate* dan tiga peraturan untuk pemfaktorannya semula. Peraturan- peraturan ini dilaksanakan dalam alat yang dinamakan *DART (Detection and Refactoring Tool)* untuk memfaktorkan semula kod sumber android bagi mengurangkan kes ujian. Hasil kod sumber yang telah difaktorkan dibandingkan dengan kod sumber asal dan

bilangan, liputan cabang serta kekompleksannya dijana menggunakan *Clover*. Hasil kajian menunjukkan pengurangan kekompleksan siklomatik kod sumber sebanyak 7.7% serta meningkatkan kualiti kes ujian melalui liputan cabang sebanyak 9.2%. Selain itu, terdapat pengurangan sebanyak 28% dalam jumlah kes ujian yang dijana. Ini menunjukkan pemfaktoran kod semula boleh menjadi pendekatan alternatif untuk mengurangkan kes ujian lewah.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF SYMBOLS AND ABBREVIATIONS

SQA	–	Software Quality Assurance
iOS	–	iPhone Operating System
IPC	–	Inter-Process Communication
SUT	–	Software under Test
DART	–	Detection and Refactoring Tool
WHLM	–	Wong, Horgan, London and Marthur
OS	–	Operating System
API	–	Application Program Interface
DVM	–	Dalvik Virtual Machine
JVM	–	Java Virtual Machine
SDK	–	Software Development Kit
XML	–	Extensible Markup Language
ILP	–	integer linear programming
WHLP	–	Wong, Horgan, London and Pasquini
HGS	–	Harrold Gupta Soffa
SPL	–	Software Product Line
XVCL	–	XML-based Variant Configuration Language
SLR	–	Systematic Literature Review
BOG	–	BiObjective Greedy
GUI	–	Graphical User Interface
JDK	–	Java Development Kit
APK	–	Android Package
FSM	–	Finite State Machine
LOC	–	Lines of Code
IDE	–	Integrated Development Environment
DR	–	Detection Rules
RR	–	Refactoring Rules
app	–	Application
OO	–	Object Oriented
OOP	–	Object Oriented Programming
TS	–	Test Suite
TC	–	Test Case
app	–	Application
APFD	–	Average Percentage of Fault Detection
FEP	–	Fault Exposing Potential

ADL	–	Array Description Language
FDRTS	–	Number of Faults Detected by the Reduced Test Suite
FDOTS	–	Number of Faults Detected by the Original Test Suite
NTCR	–	Number of Test Cases in the Reduced Test Suite
NTCO	–	Number of Test Cases in the Original Test Suite
BC	–	Branch Coverage
CC	–	Cyclomatic Complexity
AST	–	Abstract Syntax Tree
JDT	–	Java Development Tool
PDE	–	Plugin Development Environment
UAT	–	User Acceptance Testing
	–	

LIST OF APPENDICES

LIST OF PUBLICATIONS

JOURNAL:

I **Maryam Ahmed**, Rosziati Ibrahim, Noraini Ibrahim. (2015) "An Adaptation model for Android Application Testing with Refactoring" In International Journal of Software Engineering and Its Application (IJSEIA) vol. 9, no. 10, pp. 65-74.

CONFERENCE PROCEEDINGS:

I **Maryam Ahmed**, Rosziati Ibrahim, Noraini Ibrahim (2015). "Adaptation model for testing Android Application" Second International Conference for Computing Technology and Information Management (ICCTIM) pp. 130-133. IEEE indexed.

II **Maryam Ahmed**, Rosziati Ibrahim. (2015). "A Comparative Study of Web Application Testing and Mobile Application Testing" In Advanced Computer and Communication Engineering Technology, pp. 491-500 (Springer).

III **Maryam Ahmed**, Rosziati Ibrahim (2014). "Improving effectiveness of testing using reusability factor" International Conference on Computer and Information Sciences (ICCOINS) pp 1-5. IEEE indexed.

IV **Maryam Ahmed**, Rosziati Ibrahim, Noraini Ibrahim (2015). Mobile Analyzer: An Analysis Tool for Android Apps Malaysian Technical Universities Conference on Engineering and Technology (MUCET) 2015 SCOPUS indexed.

CHAPTER 1

INTRODUCTION

The idea of quality assurance has been in existence even before the advent of software [3], hence quality of products including software has been a subject of keen interest from past till present. Quality has been an issue as long as human has been producing products [3]. To endorse the quality of software, software testing becomes a requisite. Software testing is an important and major area of Software Quality Assurance (SQA). Software testing is the process of executing a program with the purpose of finding faults. Testing include effort to find defects but does not include getting solution to fixing the defects. This is the difference between testing and quality assurance as quality assurance is not limited to developing the test plan but also include testing, preventing and fixing the faults found during the process of testing. However, testing can cover different areas such as specification, design and implementation testing. Implementation testing which deals with the working system of the software is most time referred to as software testing [3].

Software testing as an aspect of SQA differs in mobile application testing. The uniqueness of mobile application testing as discussed in [4][5][6] as related to the structure of the mobile application itself include the limited resources, rapid growth in advancement, frequent updates, screen size and processing power. Of the distinctive

properties of mobile applications, frequent update has led to minimal testing, hence, increasing the chance of low quality applications. Mobile applications work on different platforms which include windows, iOS and Android. Android specifically has been going through a rapid growth and frequent updates.

Android applications, often referred to as Android apps or apps, are application software developed to run on smartphones, tablet and other devices running on Android OS [7]. The demand and market of this application continue to rise as the capabilities of the mobile phone is limitless. Initially, the mobile apps are known to perform basic tasks such as receiving and sending emails and access to the internet. Over the years, the scope of activities being done on the mobile is overwhelming and this is dependent on the available apps in the market. The number of Android applications available today is alarming. There are more than 1.6 million Android applications available for download in Google [8]. Over a million Android new devices are activated each day with over a billion apps downloaded each month [9].

According to the statistics on Appbrain website [10], 37% of the Android apps are categorized as low quality with less than 3 ratings. Though Google takes out applications from the market on a quarterly bases if found to be of bad quality. Nonetheless, these low quality applications are in the market for a while and users are able to download and use them within that period they were in market [7]. Some developers go further to reuse these bad apps to develop a new one hence, increasing the impact of the low quality apps. With about 85% of free Android apps [10], Android remains the most downloaded mobile apps [8]

The poor quality of the Android apps can be attributed to the lack of sufficient testing process due to its rapid development practice [11]. Android developers tend to ignore good testing practices as it is considered time consuming, expensive and with lots of repetitive tasks. To improve the quality of the apps, more attention needs to be given to effective approaches and tools for testing Android apps [11]. Test case generation is a core aspect of software testing. Improving the quality of test cases generated improves the quality of testing. Android test case libraries can

be unnecessary filled with redundant test cases and clones as a result of clones and redundant codes present in the source code [12]. These clones and redundant codes can be referred to as code smells.

Code smells are source code related patterns as a result of bad design and programming actions [13]. Code smells do not affect external attributes of the program as in the case of programming error. However, they remain a threat in maintainability of the software. Detection of code smells indicate areas in a source code that refactoring could be done to improve the software quality and maintainability [13]. Refactoring is defined as a practice aim at restructuring an existing code by changing the internal structure of the code without affecting the external functionality [14]. Not refactoring the code only increases difficulty in its maintainability including the testability [13], although, it does not stop the code from working properly. Thus, it can be said that refactoring helps to improve the maintainability and quality of the software. Given that testing is considered the most expensive process of software development and that the ratio of testing cost over total software development cost is increasing as time passes [15], refactoring could save money in the end. Since code smells are defined in terms of their programming styles, they can be detected using static analysis (code analysis), unlike behavioral style where dynamic information is needed. This implies that a tool for detecting code smells does not need to run the application. It only needs access to the source code and possibly libraries that are referenced by the source code. A good technique to find code smells and refactor the code will help improve the maintainability and quality of the systems.

1.1 Problem Statement

As important as software testing is, many android developers avoid it due to time and effort involved as a result of redundancies. Two different commercial crash reporting services presented the results of their big data analyses of millions of crash reports in consecutive years at DroidCon in 2012 and 2013. The top root cause of this can be associated to lack of sufficient testing [16]. The consistency of these

failures were further corroborated by a robustness evaluation of Android inter-process communication (IPC) study done by [17] in late 2012. Hence, there is need to improve testing process in android application.

Test case generation is the first and most challenging aspect of software testing. Test cases are assumed to mirror the original software under test (SUT). Hence, the effectiveness of test case generated can be associated to the quality of the source code of system under test [12]. Improving the quality and reliability of test cases generated improves the quality of testing [18], and so is an improvement in source code can enhance the quality of test case generated. Presence of code smells in android application source code has led to many redundant and avoidable test cases being generated. This eventually increases the cost and effort in testing the application [19]. Researchers in both academy and industry are making effort in minimizing the effect of the redundant test case generated [20].

The two well-known approaches presently being looked into are test case prioritization [21][22][23] and test case minimization [20][24][25]. In these two approaches, the test cases are generated which includes the redundant test cases. Most of these approaches are well established in the area of regression testing. Theres less attention given to redundancy avoidance in a newly developed application. Hence, there is need for a technique that averts redundant test cases from being generated in a new system. Asaithambi and Jarzabek [12] suggested refactoring as a possible solution that can be explored in the future to reduce generating redundant test cases. Refactoring the source code can eliminate code smells that could generate redundant test cases. On the other hand, most refactoring approaches lack a clear definition of the technique or the evaluation [26].

This research nonetheless, applies the refactoring technique to refactor the android source code to eliminate duplicates, lazy classes and small methods code smells that could possibly lead to generating redundant test cases. This is achieved by formalizing the detection and refactoring rules for the three code smells in focus. The formalization is done using set notation. Each code smell with its detection and

refactoring technique is specified. This formalization is done to clearly state and define the detection and refactoring rules as most refactoring approaches still lack a clear definition [26]. Each of the detected smell has its own refactoring rule. Manually refactoring is time consuming and takes lots of effort [26]. Automating the process is indispensable. Hence, a tool named DART (Detection and Refactoring Tool) is developed to implement the detection and refactoring rules. DART is evaluated using an android application source code by generating branch coverage and cyclomatic complexity from both the original and the refactored source code using Clover for android and the results are compared. Test cases are also generated before and after refactoring to see the effect of the refactored source code.

1.2 Research Question

Having discussed the problem statement, the research questions that guided this research are as follows:

i How is test case redundancy presently handled ?

ii How can refactoring be used to eliminate redundant test cases ?

iii Has refactoring reduced redundancy in test case ?

1.3 Research Objectives

The major objectives of the study are as follows:

i to propose the detection rules for lazy class, small method and duplicate code smells,

ii to propose the refactoring rules to refactor the detected code smells in (i),

iii to implement the detection and refactoring rules in (i) and (ii) respectively into a tool environment named Detection and Refactoring Tool (DART).

iv to evaluate DART by comparing generated cyclomatic complexity, branch coverage and number of test cases generated from original and refactored codes of an android application.

REFERENCES

1. A. Core, "The expected growth rate of adoptions of mobile apps,"2013. [Online]. Available: http://www.aumcore.com/blog/2013/05/14/the-expected-growth-rate-of-adoptions-of-mobile-apps/
2. S. Singh, "Smartphone market share by country - q2 2013: Android leads, iphone slips, windows phone inconsistent," 2013. [Online]. Available: http://www.tech-thoughts.net/2013/08/smartphone-market-share-trends-by-country-q2-2013.html# .V2NLzih96VM
3. K. Naik and P. Tripathy, SOFTWARE TESTING AND QUALITY ASSURANCE - Theory and Practice - KSHIRASAGAR NAIK, PRIYADARSHI TRIPATHY. John Wiley & Sons, 2011.
4. D. Franke and C. Weise, "Providing a software quality framework for testing of mobile applications," 2011/03 2011. [Online]. Available: http://dx.doi.org/10.1109/icst.2011.18
5. H. Muccini, A. Di Francesco, and P. Esposito, "Software testing of mobile applications: Challenges and future research directions," 2012/06 2012. [Online]. Available: http://dx.doi.org/10.1109/iwast.2012.6228987

6. C. Tao and J. Gao, "Modeling mobile application test platform and environment: testing criteria and complexity analysis," pp. 28–33, 2014.

7. F. Taibi, "On measuring the reusability proneness of mobile applications," *International Journal of Computer, Control, Quantum and Information Engineering*, vol. 8, no. 7, p. 9, 2014.

8. IDC, "Smartphone os market share in 2015," 2015. [Online]. Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp

9. Satista, "Number of apps available in leading app stores as of july 2015," 2015. [Online]. Available: http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

10. AppBrain, "New: Android market statistics overload on appbrain!" 2015. [Online]. Available: http://blog.appbrain.com/2011/03/new-android-market-statistics-overload.html

11. D. Amalfitano, A. R. Fasolino, and P. Tramontana, "A gui crawling-based technique for android mobile application testing," 2011/03 2011. [Online]. Available: http://dx.doi.org/10.1109/icstw.2011.77

12. S. Asaithambi and S. Jarzabek, *Towards Test Case Reuse: A Study of Redundancies in Android Platform Test Libraries*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, vol. 7925, book section 4, pp. 49–64. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-38977-1_4

13. D. Verloop, "Code smells in the mobile applications domain," 2013.

14. E. L. G. Alves, P. D. L. Machado, T. Massoni, and S. T. C. Santos, "A refactoring-based approach for test case selection and prioritization," in *8th International Workshop on Automation of Software Test (AST), 2013*, 2013, Conference Proceedings, pp. 93–99.

15. A. K. Jena, S. K. Swain, and D. P. Mohapatra, "A novel approach for test case generation from uml activity diagram," in *International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT), 2014*,

2014, Conference Proceedings, pp. 621–629.

16. Levy, "Crash reporting trends for mobile app developers," 2012. [Online]. Available: http://uk.droidcon.com/sessions/crash-reporting-trends-formobile-app-developers/

17. A. K. Maji, F. A. Arshad, S. Bagchi, and J. S. Rellermeyer, "An empirical study of the robustness of inter-component communication in android," in *Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on*. IEEE, 2012, pp. 1–12.

18. S. e. a. Anand, "An orchestrated survey of methodologies for automated software test case generation," *Journal of Systems and Software*, vol. 86, no. 8, pp. 1978–2001, 2013.

19. L. Chen and Q. Li, "Automated test case generation from use case: A model based approach," in *3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT), 2010*, vol. 1. IEEE, 2010, Conference Proceedings, pp. 372–377.

20. S. P. R. Asaithambi and S. Jarzabek, *Pragmatic Approach to Test Case Reuse-A Case Study in Android OS BiDiTests Library*. Springer, 2014, pp. 122–138.

21. Z. Ke, J. Bo, and W. K. Chan, "Prioritizing test cases for regression testing of location-based services: Metrics, techniques, and case study," *IEEE Transactions on Services Computing,*, vol. 7, no. 1, pp. 54–67, 2014.

22. M. Papadakis and N. Malevris, "Mutation based test case generation via a path selection strategy," *Information and Software Technology*, vol. 54, no. 9, pp. 915–932, 2012.

23. C. Zhang, A. Groce, and M. A. Alipour, "Using test case reduction and prioritization to improve symbolic execution," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*. ACM, 2014, Conference Proceedings, pp. 160–170.

24. S. K. Mondal and H. Tahbildar, "Regression test cases minimization for object oriented programming using new optimal page replacement algorithm," *International Journal of Software Engineering & Its Applications*, vol. 8, no. 6, 2014.

25. W. Zhang and D. Zhao, "Reuse-oriented test case management framework," in *International Conference on Computer Sciences and Applications (CSA), 2013*. IEEE, 2013, Conference Proceedings, pp. 512–515.

26. J. Al Dallal, "Identifying refactoring opportunities in object-oriented code: A systematic literature review," *Information and Software Technology*, vol. 58, pp. 231–249, 2015.

27. N. P. Singh, R. Mishra, and R. R. Yadav, "Analytical review of test redundancy detection techniques," *Int J Comput Appl*, pp. 0975–8887, 2011.

28. E. Raymond, "The cathedral and the bazaar," *Knowledge, Technology & Policy*, vol. 12, no. 3, pp. 23–49, 1999.

29. G. Rothermel, M. J. Harrold, J. Ostrin, and C. Hong, "An empirical study of the effects of minimization on the fault detection capabilities of test suites," in *Proceedings of International Conference on Software Maintenance, 1998.* IEEE, 1998, Conference Proceedings, pp. 34–43.

30. B. Suri and I. Mangal, "Analyzing test case selection using proposed hybrid technique based on bco and genetic algorithm and a comparison with aco," *International Journal of Advanced Research in Computer Science and Software Engineering, ISSN*, vol. 2277, 2012.

31. M. Catelani, L. Ciani, V. L. Scarano, and A. Bacioccola, "Software automated testing: A solution to maximize the test plan coverage and to increase software reliability and quality in use," *Computer Standards & Interfaces*, vol. 33, no. 2, pp. 152–158, 2011.

32. K. Minhyuk, S. Yong-Jin, M. Bup-Ki, K. Seunghak, and K. Hyeon Soo, "Extending uml meta-model for android application," 2012/05 2012. [Online]. Available: http://dx.doi.org/10.1109/icis.2012.48

33. P. Pocatilu, C. Boja, and C. Ciurea, "Syncing mobile applications with cloud storage services," *Informatica Economica*, vol. 17, no. 2/2013, pp. 96–108, 2013. [Online]. Available: http://dx.doi.org/10.12948/issn14531305/17.2.2013.08

34. M. Sharma and A. Thakur, "Review paper on android operating system," *International Journal of Emerging Trends in Science and Technology*, vol. 2, no. 05, 2015.

35. M. A. Kabir, A. Rahman, and M. I. Jabiullah, "Life cycle implementation of an android application for self-communication with increasing efficiency, storage space and high performance," 2012.

36. A. Wee, "Google announce the birth of the nexus 5x, 6p, android mashmallow and more!" *Screen*, 2016.

37. N. Mirzaei, S. Malek, C. S. Psreanu, N. Esfahani, and R. Mahmood, "Testing android apps through symbolic execution," *ACM SIGSOFT Software Engineering Notes*, vol. 37, no. 6, pp. 1–5, 2012.

38. D. Octeau, S. Jha, and P. McDaniel, "Retargeting android applications to java bytecode," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, 2012, p. 6.

39. T.-H. Su, H.-J. Tsai, K.-H. Yang, P.-C. Chang, T.-F. Chen, and Y.-T. Zhao, "Reconfigurable vertical profiling framework for the android runtime system," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 2s, p. 59, 2014.

40. C. Nimodia and H. Deshmukh, "Android operating system," *Software Engineering*, vol. 3, no. 1, p. 10, 2012.

41. M. Qu, N. Cui, B. Zou, and X. Wu, "An embedded software testing requirements modeling tool describing static and dynamic characteristics," in *2015 International Symposium on Computers & Informatics*. Atlantis Press, 2015.

42. R. Tiwari and N. Goel, "Reuse: reducing test effort," *ACM SIGSOFT*

Software Engineering Notes, vol. 38, no. 2, pp. 1–11, 2013.

43. S. Baride and K. Dutta, "A cloud based software testing paradigm for mobile applications," *SIGSOFT Softw. Eng. Notes*, vol. 36, no. 3, p. 1, 2011. [Online]. Available: http://dx.doi.org/10.1145/1968587.1968601

44. S. She, S. Sivapalan, and I. Warren, "Hermes: A tool for testing mobile device applications," 2009. [Online]. Available: http://dx.doi.org/10.1109/aswec.2009.17

45. W. Yang, M. R. Prasad, and T. Xie, "A grey-box approach for automated gui-model generation of mobile applications," pp. 250–265, 2013. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-37057-1_19

46. J. Bo, L. Xiang, and G. Xiaopeng, "Mobiletest: A tool supporting automatic black box test for software on smart mobile devices," 2007. [Online]. Available: http://dx.doi.org/10.1109/ast.2007.9

47. C. Hu and I. Neamtiu, "Automating gui testing for android applications," 2011. [Online]. Available: http://dx.doi.org/10.1145/1982595.1982612

48. L. Zhifang, L. Bin, and G. Xiaopeng, "Test automation on mobile device," 2010. [Online]. Available: http://dx.doi.org/10.1145/1808266.1808267

49. R. Black, *Pragmatic software testing: Becoming an effective and efficient test professional*. John Wiley Sons, 2013.

50. P. C. Jorgensen, *Software testing: a craftsmans approach*. CRC press, 2013.

51. A. Bertolino, "Software testing research: Achievements, challenges, dreams," 2007. [Online]. Available: http://dx.doi.org/10.1109/fose.2007.25

52. S. Kansomkeat and W. Rivepiboon, "Automated-generating test case using uml statechart diagrams," in *Proceedings of the 2003 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology*. South African Institute for Computer Scientists and Information Technologists, 2003, pp. 296–300.

53. C. Chantrapornchai, K. Kinputtan, and A. Santibowanwing, "Test case

reduction case study for white box testing and black box testing using data mining," *International Journal of Software Engineering & Its Applications*, vol. 8, no. 6, 2014.

54. M. Shahid and S. Ibrahim, "A new code based test case prioritization technique," *International Journal of Software Engineering & Its Applications*, vol. 8, no. 6, 2014.

55. H. K. Leung and L. White, "Insights into regression testing [software testing]," in *Conference on Software Maintenance, 1989., Proceedings.* IEEE, 1989, Conference Proceedings, pp. 60–69.

56. D. Jeffrey and N. Gupta, "Improving fault detection capability by selectively retaining test cases during test suite reduction," *IEEE Transactions on Software Engineering, 2007*, vol. 33, no. 2, pp. 108–123, 2007.

57. S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 67–120, 2012. [Online]. Available: http://dx.doi.org/10.1002/stvr.430

58. H.-Y. Hsu and A. Orso, "Mints: A general framework and tool for supporting test-suite minimization," in *IEEE 31st International Conference on Software Engineering, 2009. ICSE 2009.* IEEE, 2009, Conference Proceedings, pp. 419–429.

59. S. Yoo and M. Harman, "Pareto efficient multi-objective test case selection," in *Proceedings of the 2007 international symposium on Software testing and analysis.* ACM, 2007, Conference Proceedings, pp. 140–150.

60. R. C. Bryce and A. M. Memon, "Test suite prioritization by interaction coverage," in *Workshop on Domain specific approaches to software test automation: in conjunction with the 6th ESEC/FSE joint meeting.* ACM, 2007, Conference Proceedings, pp. 1–7.

61. A. M. Smith, J. Geiger, G. M. Kapfhammer, and M. L. Soffa, "Test suite reduction and prioritization with call trees," in *Proceedings of the*

twenty-second IEEE/ACM international conference on Automated software engineering. ACM, 2007, Conference Proceedings, pp. 539–540.

62. A. Leitner, M. Oriol, A. Zeller, I. Ciupa, and B. Meyer, "Efficient unit test case minimization," in *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. ACM, 2007, Conference Proceedings, pp. 417–420.

63. W. E. Wong, J. R. Horgan, S. London, and A. P. Mathur, "Effect of test set minimization on fault detection effectiveness," pp. 347–369, 1998.

64. H. Do, S. Mirarab, L. Tahvildari, and G. Rothermel, "An empirical study of the effect of time constraints on the cost-benefits of regression testing," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. ACM, 2008, Conference Proceedings, pp. 71–82.

65. X. Qu, M. B. Cohen, and G. Rothermel, "Configuration-aware regression testing: an empirical study of sampling and prioritization," in *Proceedings of the 2008 international symposium on Software testing and analysis*. ACM, 2008, Conference Proceedings, pp. 75–86.

66. S. Yoo, M. Harman, P. Tonella, and A. Susi, "Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge," in *Proceedings of the eighteenth international symposium on Software testing and analysis*. ACM, 2009, Conference Proceedings, pp. 201–212.

67. S. Elbaum, G. Rothermel, S. Kanduri, and A. G. Malishevsky, "Selecting a cost-effective test case prioritization technique," *Software Quality Journal*, vol. 12, no. 3, pp. 185–210, 2004.

68. D. P. M. Ajay Kumar Jena, Santosh Kumar Swain, "A novel approach for test case generation from uml activity diagram," 2014.

69. R. Ibrahim, M. Z. Saringat, N. Ibrahim, and N. Ismail, "An automatic tool for generating test cases from the system's requirements," pp. 861–866, 2007.

70. C. D. Nguyen, A. Marchetto, and P. Tonella, "Combining model-based and

combinatorial testing for effective test case generation," in *Proceedings of the 2012 International Symposium on Software Testing and Analysis*. ACM, 2012, Conference Proceedings, pp. 100–110.

71. R. Swain, V. Panthi, P. K. Behera, and D. P. Mohapatra, "Automatic test case generation from uml state chart diagram," *International Journal of Computer Applications*, vol. 42, no. 7, pp. 26–36, 2012.

72. Z. Wenning and Z. Dong, "Reuse-oriented test case management framework," in *International Conference on Computer Sciences and Applications (CSA), 2013*, 2013, Conference Proceedings, pp. 512–515.

73. D. Flemstrm, D. Sundmark, and W. Afzal, "Vertical test reuse for embedded systems: A systematic mapping study," 2005.

74. J. Crussell, C. Gibler, and H. Chen, "Scalable semantics-based detection of similar android applications," in *Proc. of Esorics*, vol. 13. Citeseer, 2013, Conference Proceedings.

75. E. Engström, P. Runeson, and M. Skoglund, "A systematic review on regression test selection techniques," *Information and Software Technology*, vol. 52, no. 1, pp. 14–30, 2010.

76. Y. Singh, A. Kaur, B. Suri, and S. Singhal, "Systematic literature review on regression test prioritization techniques," *Informatica*, vol. 36, no. 4, 2012.

77. J. S. Rajal and S. Sharma, "A review on various techniques for regression testing and test case prioritization," *International Journal of Computer Applications*, vol. 116, no. 16, 2015.

78. M. Rava and W. Wan-Kadir, "A review on prioritization techniques in regression testing," *International Journal of Software Engineering and its Applications*, vol. 10, no. 1, pp. 221–232, 2016, cited By 0. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84957622928&partnerID=40&md5=af4ea8f2144f4eb1548c030cbe466b96

79. M. Usaola and P. Mateo, "Mutation testing cost reduction

techniques: A survey," *IEEE Software*, vol. 27, no. 3, pp. 80–86, 2010, cited By 22. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-77951679235&partnerID=40&md5=10668429e8bdda908ad2e4f594092ffd

80. B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering–a systematic literature review," *Information and software technology*, vol. 51, no. 1, pp. 7–15, 2009.

81. S. Larguech, F. Azais, S. Bernard, M. Comte, V. Kerzerho, and M. Renovell, "A framework for efficient implementation of analog/rf alternate test with model redundancy," vol. 07-10-July-2015, 2015, pp. 621–626, cited By 0. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84957023273&partnerID=40&md5=d02c035f979e3eb21b23f48c82421e9c

82. P. Harris and N. Raju, "A greedy approach for coverage-based test suite reduction," *International Arab Journal of Information Technology*, vol. 12, no. 1, pp. 17–23, 2015, cited By 0. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84920449209&partnerID=40&md5=cb7dd627221471cb3304a518a08072e8

83. Y. Liu, C. Xu, and S.-C. Cheung, "Characterizing and detecting performance bugs for smartphone applications," pp. 1013–1024, 2014.

84. A. Petrenko and N. Yevtushenko, "Adaptive testing of nondeterministic systems with fsm," 2014, pp. 224–228, cited By 2. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84898646612&partnerID=40&md5=5736e08cc5e7dee1580bc3f87f904766

85. S. Jeyaprakash and K. Alagarsamy, "A distinctive genetic approach for test-suite optimization," vol. 62, 2015, pp. 427–434, cited By 0. [Online]. Available: https://www.scopus.

com/inward/record.uri?eid=2-s2.0-84962589372&partnerID=40&md5=5c183ff0910978644aa3a886f8d7e7af

86. A. Moraes, W. Andrade, and P. Machado, "A family of test selection criteria for timed input-output symbolic transition system models," *Science of Computer Programming*, 2015, cited By 0; Article in Press. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84967185851&partnerID=40&md5=6cd0a3d2e09f2daa708484ad5ab2610f

87. A. Hamid, M. Ilyas, M. Hummayun, and A. Nawaz, "A comparative study on code smell detection tools," *International Journal of Advanced Science and Technology*, vol. 60, pp. 25–32, 2013.

88. K. Beck, "Embracing change with extreme programming," *Computer*, vol. 32, no. 10, pp. 70–77, 1999.

89. M. Fowler, "Refactoring: Improving the design of existing code," 1999, Conference Proceedings.

90. W. F. Opdyke, "Refactoring object-oriented frameworks," Thesis, 1992.

91. M. Rieger, B. Van Rompaey, B. Du Bois, K. Meijfroidt, and P. Olievier, "Refactoring for performance: an experience report," *Proc Softw Evol*, vol. 2, no. 9, 2007.

92. R. Arcoverde, A. Garcia, and E. Figueiredo, "Understanding the longevity of code smells: preliminary results of an explanatory survey," in *Proceedings of the 4th Workshop on Refactoring Tools*. ACM, 2011, Conference Proceedings, pp. 33–36.

93. N. Moha, Y.-G. Gueheneuc, L. Duchien, and A.-F. Le Meur, "Decor: A method for the specification and detection of code and design smells," *IEEE Transactions on Software Engineering*, vol. 36, no. 1, pp. 20–36, 2010.

94. Z. Ujhelyi, G. Szke, k. Horvth, N. I. Csiszr, L. Vidcs, D. Varr, and R. Ferenc, "Performance comparison of query-based techniques for anti-

pattern detection," *Information and Software Technology*, vol. 65, pp. 147–165, 2015.

95. F. A. Fontana, M. Mangiacavalli, D. Pochiero, and M. Zanoni, "On experimenting refactoring tools to remove code smells," pp. 1–8, 2015.

96. G. Bavota, A. De Lucia, A. Marcus, and R. Oliveto, "Automating extract class refactoring: an improved method and its evaluation," *Empirical Software Engineering*, vol. 19, no. 6, pp. 1617–1664, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10664-013-9256-x

97. M. Fowler, *Refactoring: improving the design of existing code.* Pearson Education India, 1999.

98. S. Li, "Juxtapp and dstruct: detection of similarity among android applications," Thesis, 2012.

99. F. Khomh, M. Di Penta, Y.-G. Guhneuc, and G. Antoniol, "An exploratory study of the impact of antipatterns on class change-and fault-proneness," *Empirical Software Engineering*, vol. 17, no. 3, pp. 243–275, 2012.

100. W. Li and R. Shatnawi, "An empirical study of the bad smells and class error probability in the post-release object-oriented system evolution," *Journal of systems and software*, vol. 80, no. 7, pp. 1120–1128, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121206002780

101. M. Abbes, F. Khomh, Y.-G. Gueheneuc, and G. Antoniol, "An empirical study of the impact of two antipatterns, blob and spaghetti code, on program comprehension," in *15th European Conference on Software Maintenance and Reengineering (CSMR), 2011.* IEEE, 2011, Conference Proceedings, pp. 181–190.

102. F. Palomba, G. BAVOTA, R. OLIVETO, and A. DE LUCIA, "Anti-pattern detection: Methods, challenges, and open issues," *ADVANCES IN COMPUTERS, VOL 95*, vol. 95, pp. 201–238, 2014.

103. G. Hecht, R. Rouvoy, N. Moha, and L. Duchien, "Detecting antipatterns in

android apps," Thesis, 2015.

104. M. Mantyla, J. Vanhanen, and C. Lassenius, "A taxonomy and an initial empirical study of bad smells in code," in International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings. IEEE, 2003, Conference Proceedings, pp. 381–384.

105. D. Dig, "A refactoring approach to parallelism," Software, IEEE, vol. 28, no. 1, pp. 17–22, 2011. [Online]. Available: http://ieeexplore.ieee.org/ielx5/52/5672507/05672516.pdf?tp=&arnumber=5672516&isnumber=5672507

106. N. Yoshioka, H. Washizaki, and K. Maruyama, "A survey on security patterns," Progress in informatics, vol. 5, no. 5, pp. 35–47, 2008.

107. A. Garrido, S. Firmenich, G. Rossi, J. Grigera, N. Medina-Medina, and I. Harari, "Personalized web accessibility using client-side refactoring," Internet Computing, IEEE, vol. 17, no. 4, pp. 58–66, 2013. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp? arnumber=6362136

108. E. Murphy-Hill and A. P. Black, "An interactive ambient visualization for code smells," in Proceedings of the 5th international symposium on Software visualization. ACM, 2010, pp. 5–14.

109. N. Moha, Y.-G. Gueh´eneuc,´ A.-F. Le Meur, L. Duchien, and A. Tiberghien, "From a domain analysis to the specification and detection of code and design smells," Formal Aspects of Computing, vol. 22, no. 3-4, pp. 345–361, 2010.

110. Moha, Naouel, et al. "DECOR: A Method for the Specification and Detection of Code and Design Smells" *TRANSACTIONS ON SOFTWARE ENGINEERING*.

111. M. Lanza and R. Marinescu, Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems. Springer Science & Business Media, 2007.

112. N. Tsantalis and A. Chatzigeorgiou, "Identification of extract method refactoring opportunities for the decomposition of methods," Journal of Systems and Software, vol. 84, no. 10, pp. 1757–1782, 2011.

113. N. Zazworka and C. Ackermann, "Codevizard: a tool to aid the analysis of

of software evolution," in *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 2010, p. 63.

114. E. Van Emden and L. Moonen, "Java quality assurance by detecting code smells," in *Reverse Engineering, 2002. Proceedings. Ninth Working Conference on*. IEEE, 2002, pp. 97–106.

115. Y. K. Malaiya, M. N. Li, J. M. Bieman, and R. Karcich, "Software reliability growth with test coverage," *IEEE Transactions on Reliability,*, vol. 51, no. 4, pp. 420–426, 2002.

116. U. Tiwari and S. Kumar, "Cyclomatic complexity metric for component based software," *SIGSOFT Softw. Eng. Notes*, vol. 39, no. 1, pp. 1–6, 2014.

117. T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, no. 4, pp. 308–320, 1976.

118. K. Kevrekidis, S. Albers, P. J. M. Sonnemans, and G. M. Stollman, "Software complexity and testing effectiveness: An empirical study," 2009/01 2009. [Online]. Available: http://dx.doi.org/10.1109/rams.2009.4914733

119. D. Silva, R. Terra, and M. T. Valente, "Jextract: An eclipse plug-in for recommending automated extract method refactorings," 2015.

120. B. Dennis, "Repatterning: Improving the reliability of android applications with an adaptation of refactoring," 2014.

121. J. Lee, S. Kang, and D. Lee, "A survey on software product line testing," 2012. [Online]. Available: http://dx.doi.org/10.1145/2362536.2362545

122. M. Fokaefs, N. Tsantalis, E. Stroulia, and A. Chatzigeorgiou, "Jdeodorant: identification and application of extract class refactorings," pp. 1037–1039, 2011.