TESTABLE CODE DETECTION TOOL FOR OBJECT ORIENTED PROGRAMMING. A TOOL FOR NOVICE PROGRAMMER

SAIFUL BAHRI BIN HISAMUDIN

FSKTM 2019 20

TESTABLE CODE DETECTION TOOL FOR OBJECT ORIENTED PROGRAMMING. A TOOL FOR NOVICE PROGRAMMER

By

SAIFUL BAHRI BIN HISAMUDIN

Thesis submitted to School of Graduate Studies

Universiti Putra Malaysia

in Fulfillment of the Requirement for the

Master of Software Engineering

June 2019

DEDICATION

To:

This dissertation is dedicated to my parents, Supervisor, Dr. Salmi Binti Baharom, family and friends.

Abstract of dissertation presented to the Senate of Universiti Putra Malaysia in fulfillment of the requirement for the degree of Master of Software Engineering

TESTABLE CODE DETECTION TOOL FOR OBJECT ORIENTED PROGRAMMING. A TOOL FOR NOVICE PROGRAMMER

ABSTRACT

Automated software testing has gained huge attention in the last past decades due to the rapid software development cycles. Ever increasing inherent complexity, dynamic behaviors in system as well as new development paradigm required tedious work in software testing. Object oriented design and programming has become the dominant development paradigm for software projects and widely used in software development industry. The concept of object oriented ease the software development process but it make the testing process difficult. Subsequently, this will increase the testability effort in software program. Prior researches analyze the complexity and testability using object oriented design metrics where these research notably state that different attributes may add directly to the complexity of design that require more testing efforts. This research will focus on detecting the non-deterministic characteristics in software program. A tool developed which able to detect pattern in method in object oriented programming to determine the testability of the software product. The testing tool is aim at to guide novice programmer to write testable code during software development process.

Abstrak tesis yang dikemukakan kepada Universiti Putra Malaysia sebagai memenuhi keperluan untuk ijazah Sarjana Kejuruteraan Perisian

TOOL UNTUK MENGESAN KEBOLEHUJIAN ATURCARA BAGI PENGATURCARAAN BERASASKAN OBJEK. TOOL UNTUK PENGATURCARA NOVIS.

ABSTRAK

Pengujian perisian secara automatik semakin mendapat perhatian dalam beberapa dekad yang lepas. Sistem yang bertambah rumit dan dinamik serta kewujudan paradigma pembangunan baru memerlukan usaha yang lebih semasa pengujian perisian. Pengatucaraan dan reka bentuk berasaskan objek merupakan paradigma pembangunan yang dominan dan digunakan secara meluas dalam industri pembangunan perisian. Konsep pengaturcaraan berasaskan objek memudahkan pembangunan perisian tetapi menyukarkan proses pengujian perisian. Oleh itu, ini akan meningkatkan usaha terhadap kebolehujian bagi aturcara perisian. Kajian-kajian terdahulu menganalisa kerumitan dan kebolehujian dengan menggunakan metrik rekabentuk berasaskan objek di mana kajian tersebut menyatakan attribut berbeza mungkin akan menambahkan kerumitan rekabentuk yang memerlukan lebih banyak usaha semasa pengujian. Kajian ini memfokuskan kepada mengesan karekter tidak deterministik dalam atur cara perisian. Satu alat dibangunkan di mana dapat mengesan corak metod dalam pengaturcaraan berasaskan objek untuk menentukan kebolehujian terhadap produk perisian. Alat pengujian ini bertujuan untuk membantu pengaturcara novis menulis aturcara yang boleh di uji semasa proses pembangunan perisian.

ACKNOWLEDGEMENT

In the name of ALLAH, the Beneficent, the Compassionate, thanks and praise to God for giving me strength and patience to complete my duties successfully.

I would like to express my gratitude to parents and family for supporting my education. They have been a great support throughout my life and studies. Motivations and encouragements from parents were never lacking in any situation.

Next, I would like to sincerely thank and express my deep thanks and gratitude to my supervisor Dr. Salmi Binti Baharom for her endless support, guidance, correction, encouragement, advice and valuable observations in my Master's degree.

Finally, to all UPM staff, thanks you for your facilitation. I would like to acknowledge to any individual who are not mentioned here for his/her irreplaceable helps and cooperation.

.

APPROVAL

Thesis submitted to the Senate of University Putra Malaysia and has been accepted as fulfillment of the requirement for the degree of Master of Software Engineering.

_____

Supervisor,

Salmi Binti Baharom, PhD.

Department of Software Engineering and Information Systems

Faculty of Computer Science and Information Technology

Universiti Putra Malaysia

June, 2019

DECLARATION

I hereby confirm that:

- this thesis is my original work;
- quotations, illustrations and citations have been duly referenced;
- this thesis has not been submitted previously or concurrently for any other degree at any other institutions;
- intellectual property from the thesis and copyright of thesis are fully-owned by Universiti Putra Malaysia, as according to the Universiti Putra Malaysia (Research) Rules 2012;
- written permission must be obtained from supervisor and the office of Deputy Vice-Chancellor (Research and Innovation) before thesis is published (in the form of written, printed or in electronic form) including books, journals, modules, proceedings, popular writings, seminar papers, manuscripts, posters, reports, lecture notes, learning modules or any other materials as stated in the Universiti Putra Malaysia (Research) Rules 2012;
- there is no plagiarism or data falsification/fabrication in the thesis, and scholarly integrity is upheld as according to the Universiti Putra Malaysia (Graduate Studies) Rules 2003 (Revision 2012-2013) and the Universiti Putra Malaysia (Research) Rules 2012. The thesis has undergone plagiarism detection software.

Signature: _____ Date: _____

Name : Saiful Bahri Bin Hisamudin

Matric No.: GS49373

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

OOP	OBJECT ORIENTED PROGRAMMING
ISO	INTERNATIONAL ORGANIZATION FOR STANDARDIZATION
IEEE	INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS
CPU	CENTRAL PROCESSING UNIT
UPM	UNIVERSITI PUTRA MALAYSIA
API	APPLICATION PROGRAMMING INTERFACE
OSS	OPEN SOURCE SOFTWARE
JSL	JAVA STANDARD LIBRARY

CHAPTER 1

INTRODUCTION

1.1 Background

Software testing always perceived as being time consuming, increasing the cost and less significant but in reality it is important to produce high quality software product. It is mainly consist of validation and verification process that determine whether developed system meets user's requirement [1]. Meanwhile, implementation of object oriented programming during software development process has improved the quality of software product while increasing its complexity. As the complexity of software increases, testability effort increased in line with it. Testability effort can be reduce by automating the test which are more effective and efficient especially in large scale software project as it may need to be executed repetitively and time consuming. Automating the testing reduced human intervention that prone to error.

The most commonly used types of software testing are Black Box Testing, White Box Testing and Grey-box Testing. In White Box Testing which is also known as clear-box testing, the detailed analysis of internal structure and source code are required where software engineer has full access to internal structure and source code of the software. However, testing the internal structure of source code requires profound knowledge and skill for the programming language of the source code [2]. Internal structure of source code will determine testability effort needed by the software engineer.

Testability in [12] is defined is the ease of performing testing. Software testability is an external software quality attribute that evaluates the complexity and the effort required for software testing where it is a key aspect in detecting difficult error to uncover defects in software.

Testability in object oriented technology affected by encapsulation, inheritance and coupling software design properties among others. Encapsulation is the main feature of object oriented technology [6] where it involves the concept of information hiding. Encapsulated parameters and methods declared as public or private will affect the complexity and testability of software. The non-deterministic characteristic make it impossible to test internal logic of method in the perspective of unit testing.

Therefore, there is a need to detect non-deterministic characteristics in software program to evaluate the code testability. This research developed a testable code detection tool that able to detect non-deterministic characteristics in software program to determine testability of the software product. In [10] stated that testing tools help to reduces time, cost and effort in by automating the testing process compared to manual process which is tedious and time consuming.

1.2 Problem statements

Unit testing usually considered as an approach for white box testing which is one of the strategy in software testing. Other software testing strategy are integration testing, system testing and acceptance test. Unit testing is a way of running a module in isolation away is a usage of the test cases prepared and hence comparing the predicted result from module designing and the actual result. This testing is done by the developer and the proper knowledge of core program designing is required. It is the first level of testing which helps to make whole picture of software testing as a complete system [5].

To enhance testing effectiveness, the developed artifacts (requirements, code) must be designed to be testable [4]. By examining the internal logic of the code, white box testing is basically a process of providing input values and monitoring how the system processes in order to provide desires output. An isolated module without input values make it impossible to test. However, module with input values may contain untestable code's characteristic such as poisoning the code base with non-deterministic factors and side effect. To improve testability, bad practices in writing code must be avoid to ensure the generated code are less troublesome, more robust and easy to maintain.

Therefore, it would be useful to identify whether developed code are too complex and poorly design. To determine that the code is testable, code's characteristics need to be assess by the software developer in the early stage of software development.

In this paper we proposed a tool to detect testable code in object oriented programming for novice programmer. The proposed tool is based on the object oriented programming language because it is widely applicable in software development industry.

1.3 Research objectives

The main objective of this research is to propose a testing tool that can detect the non-deterministic characteristics in software program in which be able to increase the software testability during the testing phase. Other objectives as below:

i) To study the criteria of non-deterministic characteristics in object oriented programming.
ii) To develop a testing tool that able to detect the non-deterministic characteristics in object oriented programming.
iii) To compare and evaluate the effectiveness of testing tool on the software program.

1.4 Scope of the study

The scope of this study is specifically to help the novice programmer in writing a testable code in object oriented programming language during software development process by detecting non-deterministic characteristics in software program.

1.5 Dissertation organization

The remaining chapters of this research are ordered as follows:

CHAPTER 2 discusses general overview about software testing and its activities. It also discusses on external software quality metrics that affecting testability such controllability, and observability. In addition to that, it also explanation on how controllability relate to the non-deterministic characteristic of software program. This chapter also covered the sources of non deterministic characteristic in software program based on the related work of the study.

CHAPTER 3 presents the overall methodology used to conduct this research in order to achieve the main goal of the research. There are two parts: first part provides the general overview of the methodology and divided into five phases by providing short explanations of each phase. On the second parts, it discusses in detail the components and how covered each phase.

CHAPTER 4 contains two main sections: the first section explain the framework of the proposed tool in this study while second section explain implementation section of the framework of the proposed tool.

CHAPTER 5 explains the evaluation of the study or research. An evaluation framework provided to give an overview followed by detail explanation of evaluation process. Then, result from manual detection compared to the result generated by the developed tool.

CHAPTER 6 gives explanation about the conclusion. It discussed the achievements of the study based on the objectives and research questions. Then, it discussed contributions, limitation as well as future works.

References

[1] Muhammad Abid Jamil, Muhammad Arif, Normi Sham Awang Abubakar, Akhlaq Ahmad (2016). Software Testing Techniques: A Literature Review. 2016 6th International Conference on Information and Communication Technology for The Muslim World. IEEE.

[2] Rashad Khalid (2017). Towards an Automated Tool for Software Testing and Analysis. Proceedings of 2017 14th International Bhurban Conference on Applied Sciences & Technology (IBCAST), Islamabad, Pakistan, 10th – 14th January, 2017. IEEE.

[3] Er. Mohit kumar, Mr. Parteek Sharma and Dr. Harsh Sadawarti (2010). Measuring Testability of Aspect Oriented Programs. 2010 International Conference on Computer Applications and Industrial Electronics (ICCAIE 2010), December 5-7, 2010, Kuala Lumpur, Malaysia. IEEE.

[4] Jane Huffman Hayes, Wenbin Li,Tingting Yu, Xue Han, Mark Hays, Clinton Woodson (2015) Measuring Requirement Quality to Predict Testability. Advancing Innovation in Residency Education(AIRE) 2015,Ottawa,ON,Canada. IEEE.

[5] Jai Gaur, Akshita Goyal, Tanupriya Choudhury, Sai Sabitha (2016). A walk through of software testing techniques. Proceedings of the SMART -2016, IEEE Conference ID: 39669. IEEE.

[6] Sadaf Khalid,Saima Zehra, Fahim Arif (2010). Analysis of Object Oriented Complexity and Testability Using Object Oriented Design Metrics. NSEC'10, 04-OCT-2010, Rawalpindi, Pakistan. ACM 978-1-4503-0026-1/10/10.

[7] Khaled M. Mustafa, Rafa E. Al-Qutaish, Mohammad I. Muhairat (2009). Classification of Software Testing Tools Based on the Software Testing Methods. 2009 Second International Conference on Computer and Electrical Engineering. IEEE.

[8] International Standard ISO/IEC/IEEE 29119-1, First edition 2013-09-01

[9] Pradeep Kumar Singh, Om Prakash Sangwan, Amar Pal Singh, Amrendra Pratap (2015). An Assessment of Software Testability using Fuzzy Logic Technique for Aspect-Oriented Software. I.J. Information Technology and Computer Science, 2015, 03, 18-26.

[10] Tao Xie, Nikolai Tillmann, Pratap Lakshman (2016). Advances in Unit Testing: Theory and Practice. ICSE '16 May 14-22, 2016, Austin, TX, USA. ACM ISBN 978-1-4503-4205-6/16/05.

[11] IEEE Standard Glossary (1990) Page 80. IEEE.

[12] Mahfuzul Huda, Dr.Y.D.S.Arya, Dr. M. H. Khan (2014). Measuring Testability of Object Oriented Design: A Systematic Review. International Journal of Scientific Engineering and Technology. Volume No.3 Issue No.10, pp : 1313-1319

[13] Pushpa R. Suri andHarsha Singhani (2015). Object Oriented Software Testability (OOST) Metrics Analysis. International Journal of Computer Applications Technology and Research. Volume 4– Issue 5, 359 - 367, 2015, ISSN:- 2319–8656.

[14] Harsha Singhani Ratnani and Dr. Pushpa R. Suri (2015). Testability Assessment Model for Object Oriented Software based on Internal and External Quality Factors. Global Journals Inc. (US). Volume 15 Issue 5 Version 1.0 Year 2015

[15] Vahid Garousi, Michael Felderer and Feyza Nur Kılıçaslan (2018). What we know about software testability: a survey. Page 17.

[16] R.V. Binder (1994), Design for Testability in Object-Oriented Systems, Communications of the ACM, 37(9): p. 87-101. ACM.

[17] Vesa Vainio andJorma Sajaniemi (2007). Factors in Novice Programmers' Poor Tracing Skills. ITiCSE'07, June 23–27, 2007, Dundee, Scotland, United Kingdom. ACM 978-1-59593-610-3/07/0006.

[18] Yizhou Qian and James Lehman (2017). Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. ACM Trans. Comput. Educ. 18, 1, Article 1 (October 2017), 24 pages.

[19] Joseph Devietti, Brandon Lucia, Luis Ceze, Mark Oskin (2009). DMP: Deterministic Shared Memory Multiprocessing. ASPLOS'09, March 7–11, 2009, Washington, DC, USA. ACM978-1-60558-406-5/09/03

[20] Marek Olszewski Jason Ansel Saman Amarasinghe (2009). Kendo: Efficient Deterministic Multithreading in Software. ASPLOS '09 March 7–11, 2009, Washington, DC, USA. ACM978-1-60558-406-5/09/03.

[21] Robert L. Bocchino Jr., Vikram S. Adve, Sarita V. Adve and Marc Snir (2009). Parallel Programming Must Be Deterministic by Default. HotPar'09 Proceedings of the First USENIX conference on Hot topics in parallelism Pages 4-4. ACM.

[22] Harish Patil, Cristiano Pereira, Mack Stallcup, Gregory Lueck, James Cownie (2010). PinPlay: A Framework for Deterministic Replay and Reproducible Analysis of Parallel Programs. CGO'10, April 24–28, 2010, Toronto, Ontario, Canada. ACM 978-1-60558-635-9/10/04.

[23] Jacob Burnim and Koushik Sen (2009). Asserting and Checking Determinism for Multithreaded Programs. ESEC-FSE'09, August 24–28, 2009, Amsterdam, The Netherlands. ACM 978-1-60558-001-2/09/08.

[24] Cedomir Segulja Tarek S. Abdelrahman (2014). What is the Cost of Determinism?. PACT '14 Proceedings of the 23rd international conference on Parallel architectures and compilation Pages 99-112. ACM.

[25] Mohammad Mahdi Hassan, Wasif Afzal, Martin Blom, Birgitta Lindstrom, Sten F. Andler and Sigrid Eldh (2015). Testability and Software Robustness: A Systematic Literature Review. 2015 41st Euromicro Conference on Software Engineering and Advanced Applications. IEEE.

[26] Alex Gyori, Ben Lambert, Sarfraz Khurshid and Darko Marinov (2016).Exploring Underdetermined Specifications using Java PathFinders. ACM. ISBN 978-1-4503-2138-9.

[27] Amittai Aviram, Shu-Chun Weng,Sen Hu, Bryan Ford (2012). Efficient system-enforced deterministic parallelism. Communications of the ACM. Volume 55 Issue 5, May 2012. Pages 111-119. ACM.

[28] Sandra Loosemore, Richard M. Stallman, Roland McGrath, Andrew Oram, and Ulrich Drepper. The GNU C Library Reference Manual Version 2.29.

[29] August Shi, Alex Gyori, Owolabi Legunsen and Darko Marinov (2016). Detecting Assumptions on Deterministic Implementation of Non-deterministic Specifications. 2016 IEEE International Conference on Software Testing, Verification and Validation.

[30] Hirohiko Suwa, Akinori Ihara, Raula Gaikovina Kula, Daiki Fujibayashi, Kenichi Matsumoto (2017). An Analysis of Library Rollbacks: A Case Study of Java Libraries. 2017 24th Asia-Pacific Software Engineering Conference Workshops. IEEE.

[31] JavaDoc “https://docs.oracle.com/javase/8/docs/api/index.html”

[32] Qingzhou Luo, Farah Hariri, Lamyaa Eloussi and Darko Marinov (2014). An Empirical Analysis of Flaky Tests. FSE’14, November 16–21, 2014, Hong Kong, China. ACM 978-1-4503-3056-5/14/11.

[33] Sina Shamshiri, Ren´e Just, Jos´e Miguel Rojas , Gordon Fraser , Phil McMinn and Andrea Arcuri (2015). Do Automatically Generated Unit Tests Find Real Faults? An Empirical Study of Effectiveness and Challenges. 2015 30th IEEE/ACM International Conference on Automated Software Engineering. IEEE.

[34] Bell, Owolabi Legunsen, Michael Hilton, Lamyaa Eloussi, Tifany Yung and Darko Marinov (2018). DeFlaker: Automatically Detecting Flaky Tests Jonathan. ICSE ’18, May 27-June 3, 2018, Gothenburg, Sweden. ACM ISBN 978-1-4503-5638-1/18/05

[35] Xu Zhou, Kai Lu, Xiaoping Wang, Wenxhe Zhang, Khai Zhang, Xu Li, Gen Li (2013). Deterministic Message Passing for Distributed Parallel Computing. IEICE TRANS. INF. & SYST., VOL.E96-D, NO.5 MAY 2013.

[36] Doug Lea (2005). The java.util.concurrent synchronizer framework. Science of Computer Programming 58 (2005) 293–309. ScienceDirect.