UNIVERSITI PUTRA MALAYSIA

FAULT DEPENDENCY AND LOCATION ANALYSIS TO IMPROVE MULTIPLE FAULT LOCALIZATION

MUHAMMAD LUQMAN BIN MAHAMAD ZAKARIA

FSKTM 2020 28

FAULT DEPENDENCY AND LOCATION ANALYSIS TO IMPROVE MULTIPLE FAULT LOCALIZATION

By

MUHAMMAD LUQMAN BIN MAHAMAD ZAKARIA

Thesis Submitted to the School of Graduate Studies, Universiti Putra Malaysia, in Fulfilment of the Requirements for the Degree of Doctor of Philosophy

December 2019

DEDICATION

This thesis is dedicated to my beloved parent, family, supervisory committee who always support me through PhD Journey

Abstract of thesis presented to the Senate of Universiti Putra Malaysia in fulfilment of the requirement for the degree of Doctor of Philosophy

FAULT DEPENDENCY AND LOCATION ANALYSIS TO IMPROVE MULTIPLE FAULT LOCALIZATION

By

MUHAMMAD LUQMAN BIN MAHAMAD ZAKARIA

December 2019

Chairman : Khaironi Yatim Sharif, PhD
Faculty : Computer Science and Information Technology

In a software development life cycle, two phases which are considered as critical are software testing and software maintenance. The cost involved in both phases is high, ranging from 40% to 67% of the total cost of software development. Due to this issue, various studies have been done in both phases, especially in fault localization. Finding the root cause of the faults in a program is one of the crucial parts in software testing and maintenance.

Many techniques have been proposed, such as program slicing, code coverage, program state, and mutation analysis. Although all these techniques give a good insight into fault localization, it appears that these techniques are made based on the assumption that a single fault causes the faults. In reality, one fault could also possibly caused by multiple faults.

In Coverage-Based Fault Localization, several techniques have been proposed to address the above problem by using suspiciousness formula to locate the location of the faults. However, this formula only indicate the line location containing the fault. It is not suitable for cases where a line contains more than one fault. Suspiciousness formula is unable to identify which operators are faulty as the technique can only highlight line numbers. This requires a search technique which enable suspiciousness formula to perform analysis on all faulty operators either on the same line or different line number.

In the same light, multiple faults problem might be more complicated in object-oriented setting. Object-oriented is made up of multiple classes and methods where they can interact through class objects. Eventhough object-oriented is made up of various components and better features compared to structured approach, it still has problems related with logical errors, operators, and programmatic styles. Considering the three

problems related to object-oriented which are logical errors, operators, and programmatic styles, it seems that all of these are underpinned by operators. This can be explained by the wide usage of operators in logical operations (comparison) and program flow directive (condition, counter, looping, etc). Likewise, programmers' programmatic syles also relate to operators such as function, algorithm, checking, and interface. Hence, operators are suggested in the investigation of multiple fault localization for both structured and object-oriented programming.

Fault dependency Identification (FDI) and Fault Dependency and Location analysis had been proposed to handle to solve problem related to multiple faults. FDI was designed to capture the class and method structures as well as the dependencies between classes and methods. This is to ensure that all the related object class is examined during analysis. FDLA was designed with the aims to find the location of the faults by doing some modifications on a part of the code. To achieve this goal, a technique based on mutation was used called Hybrid Genetic Algorithm.

Genetic algorithm (GA) is well known for finding an optimal solution to a problem while a local search is capable of removing duplication. Since both have their advantages, both were combined into one technique called Hybrid Genetic Algorithm for Multiple Fault Localization (HGAMFL). An experiment was executed on five Java programs against O^p. Results of the experiment and statistical tests showed strong evidence that HGAMFL is able to localize multiple faults more effectively and accurately compared to O^p for a situation where multiple faults appear at the same line number or different line numbers.

As a conclusion, the results of the study show that the combination of Genetic Algorithm and local search had improved the effectiveness in localizing multiple faults in Java programs. This technique can identify dependency between faults and return the accurate coordinate location of the faults.

Abstrak tesis yang dikemukakan kepada Senat Universiti Putra Malaysia sebagai memenuhi keperluan untuk ijazah Doktor Falsafah

ANALISA KEBERGANTUNGAN KESALAHAN DAN LOKASI BAGI MENINGKATKAN PENYETEMPATAN PELBAGAI KESALAHAN

Oleh

MUHAMMAD LUQMAN BIN MAHAMAD ZAKARIA

Disember 2019

Pengerusi : Khaironi Yatim Sharif, PhD
Fakulti : Sains Komputer dan Teknologi Maklumat

Di dalam kitaran hayat pembangunan perisian, terdapat dua fasa yang dianggap sebagai kritikal iaitu ujian perisian dan penyelenggaraan perisian. Kedua-dua fasa ini memakan kos yang sangat tinggi, yang meliputi sekitar 40% hingga 67% daripada jumlah kos pembangunan perisian. Disebabkan isu ini, pelbagai kajian telah dilakukan oleh penyelidik di dalam kedua-dua fasa ini, terutamanya dalam lapangan berkaitan penyetempatan kesalahan. Hal demikian kerana mencari punca kesalahan adalah bahagian penting dalam ujian dan penyelenggaraan perisian.

Pada masa ini, terdapat banyak teknik yang telah dicadangkan seperti *program slicing*, *code coverage*, *program state*, dan *mutation analysis*. Walaupun semua teknik ini menunjukkan peningkatan yang baik dalam penyetempatan kesalahan, namun ternyata teknik-teknik ini dibuat berdasarkan anggapan bahawa kesalahan disebabkan oleh kesalahan tunggal sahaja. Secara realitinya, satu kesalahan juga boleh disebabkan oleh pelbagai kesalahan.

Dalam Penyetempatan Kesalahan Berasaskan Liputan, beberapa teknik telah diusulkan untuk mengatasi masalah di atas dengan menggunakan formula kecurigaan untuk mencari lokasi kesalahan. Walau bagaimanapun, formula ini hanya menunjukkan baris yang mengandungi kesalahan. Ia tidak sesuai untuk kes di mana sesuatu baris yang mengandungi lebih daripada satu kesalahan. Formula kecurigaan tidak dapat mengenal pasti operator mana yang salah kerana teknik ini hanya dapat menunjukkan nombor baris sahaja. Ini memerlukan teknik carian yang membolehkan formula kecurigaan untuk melakukan analisis terhadap semua operator yang salah sama ada pada baris yang sama atau nombor baris yang berbeza.

Pada masa yang sama, isu berkaitan pelbagai kesalahan adalah lebih rumit di dalam persekitaran berorientasikan objek.

Objek orientasi mengandungi pelbagai kelas dan fungsi dan ia berkomunikasi melalui objek. Walaupun objek orientasi terdiri dari berbagai komponen dan fungsi yang lebih baik dibandingkan dengan pendekatan struktur, ia masih mempunyai masalah yang berkaitan dengan kesalahan logik, operator, dan gaya program. Mengambil kira tiga masalah yang berkaitan dengan objek orientasi iaitu kesalahan logik, pengendali, dan gaya program, jelas kelihatan kesemua masalah ini adalah disebabkan oleh penggunaan operator. Ini dapat dijelaskan oleh penggunaan operator yang meluas dalam operasi logik (perbandingan) dan arahan aliran program (keadaan, kaunter, perulangan, dll). Tidak terkecuali juga gaya pengaturcaraan program di mana ianya berkaitan dengan operator seperti fungsi, algoritma, pemeriksaan, dan antara muka. Oleh itu, penggunaan operator dicadangkan dalam penyiasatan penyetempatan pelbagai kesalahan untuk pengaturcaraan berstruktur dan berorientasikan objek.

Pengenalpastian kebergantungan kesalahan (FDI) dan Analisis Kebergantungan kesalahan dan lokasi telah dicadangkan untuk menyelesaikan masalah berkaitan penyetempatan pelbagai kesalahan. FDI direka untuk mendapatkan struktur kelas dan fungsi serta kebergantungan diantara pelbagai kelas dan fungsi. Ia untuk memastikan semua objek kelas yang berkaitan diperiksa semasa analisis dijalankan. FDLA pula direka dengan tujuan untuk mencari lokasi kesalahan dengan melakukan pengubahsuaian pada sebahagian kod aturcara. Untuk tujuan ini, satu teknik berasaskan mutase telah dgunakan iaitu Hybrid Genetic Algorithm.

Algoritma Genetik (GA) adalah satu teknik yang popular dalam mencari penyelesaian yang optimum bagi sesuatu masalah manakala pencarian tempatan pula mampu menghapus duplikasi. Oleh kerana kedua-duanya mempunyai kelebihan mereka sendiri, kedua-dua teknik telah digabungkan dalam satu teknik yang dikenali sebagai Hibrid Algoritma Genetik untuk Penyetempatan Pelbagai Kesalahan (HGAMFL). Eksperimen telah dilaksanakan pada HGAMFL dan O^p dengan menggunakan lima Program Java. Hasil eksperimen dan ujian statistik menunjukkan bahawa terdapat bukti yang kuat bahawa HGAMFL dapat melaksanakan penyetempatan pelbagai kesalahan lebih berkesan dan tepat berbanding dengan O^p untuk keadaan di mana banyak kesalahan muncul pada baris yang sama atau berbeza.

Sebagai kesimpulan, hasil kajian menunjukkan bahawa gabungan Algoritma Genetik dan pencarian tempatan telah meningkakan tahap keberkesanan dalam melaksanakan proses penyetempatan pelbagai kesalahan di dalam program Java. Teknik ini dapat mengenalpasti kebergantungan antara kesalahan dan menunjukkan koordinat lokasi yang betul bagi sesuatu kesalahan.

ACKNOWLEDGEMENTS

Firstly, I would like to express my sincere gratitude to my supervisor, Dr. Khaironi Yatim Sharif for the continuous support of my Ph.D study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not imagine having a better advisor and mentor for my Ph.D study.

Apart from my advisor, I would like to thank the rest of my thesis committee, Prof. Dr Abdul Azim Abd Aziz, Prof. Madya Dr. Hazura Zulzalil, and Dr. Koh Tieng Wei for their insightful comments and encouragement, as well as the hard questions which led me to widen my research from various perspectives.

I thank my fellow labmates for the stimulating discussions, for the sleepless nights of working before deadlines, and for all the fun we have had in the last five years.

Last but not least, I would like to thank my family, my parents, Mahamad Zakaria M Kabir Sehib and Normadiah Jamaluddin, and my brothers for supporting me spiritually throughout writing this thesis and my life in general.

This thesis was submitted to the Senate of the Universiti Putra Malaysia and has been accepted as fulfilment of the requirement for the degree of Doctor of Philosophy. The members of the Supervisory Committee were as follows:

Khaironi Yatim Sharif, PhD
Senior Lecturer
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
(Chairman)

Abdul Azim Abd Ghani, PhD
Professor
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
(Member)

Koh Tieng Wei, PhD
Associate Professor
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
(Member)

Hazura Zulzalil, PhD
Associate Professor
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
(Member)

ZALILAH MOHD SHARIFF, PhD
Professor and Dean
School of Graduate Studies
Universiti Putra Malaysia

Date: 08 October 2020

Declaration by graduate student

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 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 and Matric No: Muhammad Luqman bin Mahamad Zakaria GS40294

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

SDLC	Software Development Life Cycle
GA	Genetic Algorithm
HGA	Hybrid Genetic Algorithm
HGAMFL	Hybrid Genetic Algorithm for Multiple Fault Localization
FDI	Fault Dependency Identification
FDLA	Fault Dependency and Location Analysis

CHAPTER 1

INTRODUCTION

1.1 Research Background and Motivation

Technology is rapidly changing. To cope with these changes, software development setting is also drastically changed. For example, the software itself was transformed from just a stand-alone application or software to a cloud-based application. As shown in Figure 1.1, software evolution has changed the environment of software development. Each paradigm experienced different environmental changes such as organization, middleware, requirements, and functionalities. Software should be maintained to ensure the functionalities are aligned with the environmental changes.

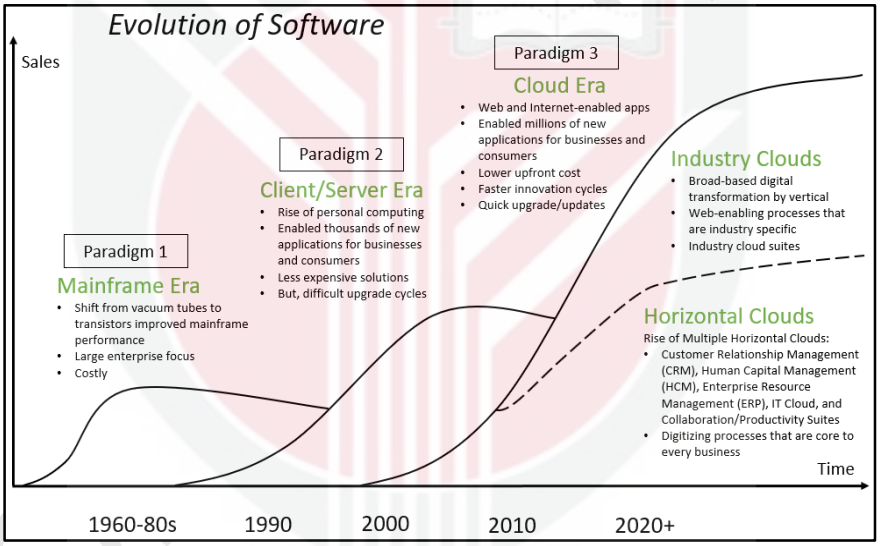

Figure 1.1 : Evolution of software (Jonathon, 2018)

The new technology requires an effective adaptive maintenance effort to ensure the particular system works correctly and is adaptable to the latest technology. The technological changes in a software development setting will directly affect developers (Mens et al., 2005). Figure 1.2 shows various challenges faced by a software developer during software development. These challenges are likely to harm the software development process (Stack Overflow, 2016), even more so if the software developer lacks the skills, knowledge, and experience. It may cause the software developer to develop fragile and buggy codes which can affect the produced software quality.

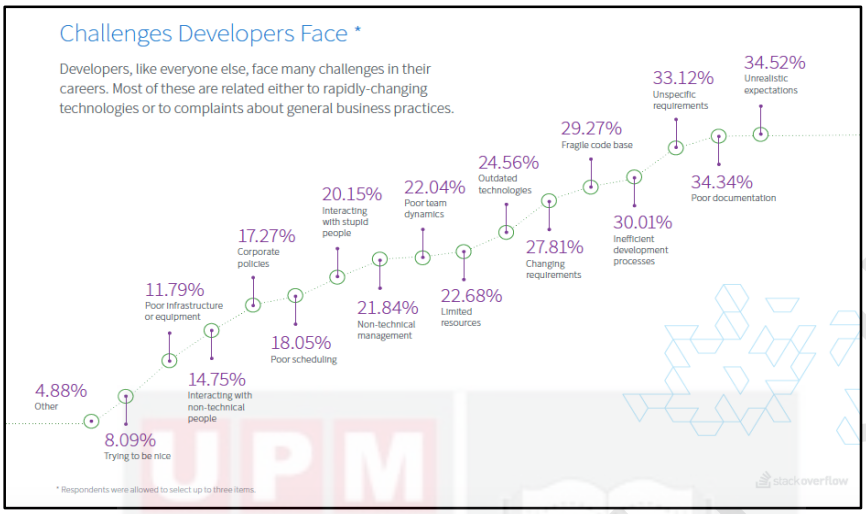

Figure 1.2 : Challenges faced by a software developer (Stack Overflow, 2016)

Producing and maintaining high-quality software requires specific skills and adoption of good and controlled development and operation practices (Laporte & April, 2017). According to ISO/IEC software quality standards, several characteristics need to be considered during software development such as functionality, reliability, usability, efficiency, maintainability, and portability (Nuseibeh & Easterbrook, 2000). However, maintaining software quality is difficult due to software evolution. Evolution of software could result in higher complexity and more bugs (Basili & Perricone, 1983; Erdweg et al., 2014).

In this context, McConnel reported that ten defects are found in every 1000 lines of code (McConnell, 2004). If a new functionality is added into the software, lines of code as well as bugs and program complexity would also be increased. Due to these facts, more time is spent by the software developer and tester in the software testing and maintenance phase. In software maintenance, slow speed of implementation leads to higher cost during maintenance (Sharif, 2012). It is shown that the cost ranges from 60% to 70% of the total software development cost (Malhotra & Chug, 2016). Software maintenance can be categorized into four classes: adaptive, perfective, corrective, and preventive (Bennett & Rajlich, 2000; Lientz et al., 1978). Among these four classes, fixing faults requires at least 21% of maintenance effort (Bennett & Rajlich, 2000). The reason is that software correction can only be performed if the fault location is found. Therefore, fault identification can be achieved through software testing.

Software testing is one of the critical phases that falls under software engineering. This phase aims to observe the execution of a software system and validate the behavior of the system, whether it behaves as intended (Bertolino, 2007). As a result, any part of the system that contains faults can be detected. Various types of software testing activities can be executed by the software developer or tester. They involve testing activities on a

piece of a small code developed by the developer (unit testing), testing during source code integration (integration testing), testing on the acceptance level of the user against the system (acceptance testing), and monitoring service applications during run time (M. Ali & Fairouz, 2015).

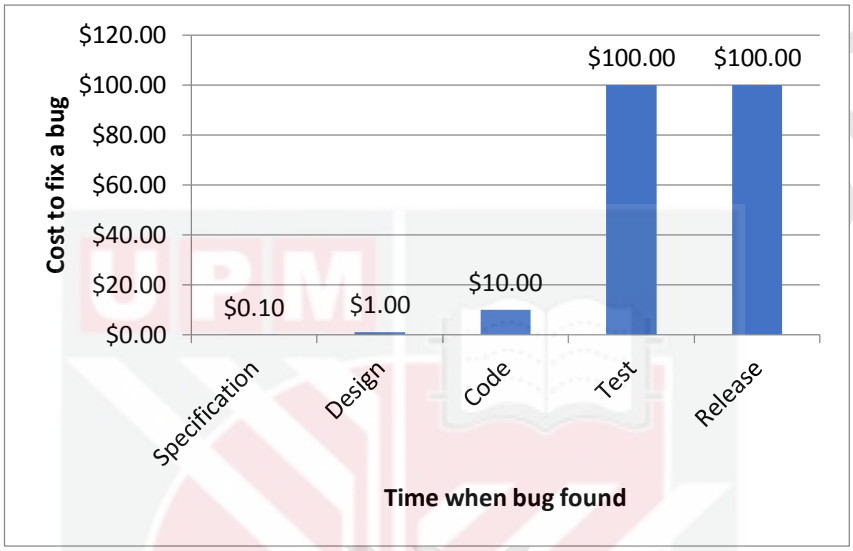

Figure 1.3 : Total bugs found in the software development phase
(E. Burton Swanson, 1976)

According to E. Burton Swanson, most software needs to go through software maintenance due to software failure during execution (Swanson, 1976). As shown in Figure 1.3, the cost to fix the bugs increases for each stage in software development (Patton, 2001). The later the bugs are detected during SDLC stages, the higher the cost incurred as the bugs get more severe. Hence, it is essential to detect bugs as early as possible. Most of the bugs can be detected during testing and after the software is released to the user. This demonstrates the importance of conducting a more in-depth study in finding bugs in computer software. Several techniques had been introduced for improving the process of finding faults such as fault prediction (B. Li et al., 2014; Rawat & Dubey, 2012), fault detection (Dunwei Gong & Zhang, 2013; Ko-Li Cheng et al., 2011), and fault localization (Abreu et al., 2009b; Moon et al., 2014; Sun et al., 2013).

Although fault prediction and detection can help software developers to predict and detect the location of the faults, both approaches are not able to identify the root cause of the faults. As suggested by Wotawa, fault localization is more significant as it can reduce the time taken during debugging and is capable in finding the root cause of the fault in a program with less effort (Wong et al., 2016). Work on fault localization has been on-going for a number of years. Various techniques have been proposed to tackle issues on fault localization as well as reducing its time consumption. With the

improvement of single fault localization technique in recent years, it is now possible to extend the study to localize multiple faults. However, dealing with multiple faults requires a different strategy compared to the single fault localization technique due to the nature of multiple faults that can appear anywhere in a program. Hence, it is important to ensure that the correct fault is pointed out, and no new faults are generated during fault localization process (Demott, 2012; DiGiuseppe & Jones, 2015; Sahoo, 2012). This is one of the known challenges in this approach.

Object-oriented framework is designed to support modularity, extensibility, and reuseability (Fayad & Schmidt, 1997). Eventhough its features provide a better structure and ease the development process, several authors have shown that these features are often the limiting factor in object-oriented approach which lead to more bugs in the program. Some of the faults found during debugging will also generate more bugs in the program (C. T. Chen et al., 2009; Nguyen et al., 2010). Until 2017, object-oriented programming such as C++, Java, and Python had been widely used in software development. As reported by Putano, about 13.27% of software developers had used Java language in developing a software or application (Putano, 2017). It is believed that the number of Java language applications will grow in the future. As this can have serious consequences, it is imperative to better understand how fault localization techniques could leverage their advantages in dealing with object-oriented.

1.2 Problem Statement

Various works on finding the correct fault on computer programs have been proposed. Although results indicate that current fault localization techniques are capable of localizing fault correctly and effectively, they are only tested on a single type of fault (Wong et al., 2016). In real software, one program might contain more than one type of fault located at different line numbers (Abreu et al., 2011). Indeed, some of the faults might be caused by another fault or a combination of faults. Thus, it is essential to enhance fault localization research from a single type of fault to multiple types of fault. Zoltar-M (Abreu et al., 2011) and Barinel (Abreu et al., 2009b) are two techniques proposed to handle multiple faults. Both of the techniques have been executed on programs containing multiple faults. However, the output for both techniques only indicates the location of the row containing the error. In a cases where more than one fault appears on the same row, it is necessary to point out the location more accurately and show the type of operator identified as a fault since one line can contain more than one operators.

To deal with multiple faults, a developer cannot simply select all the highest suspiciousness statements as the root cause of the fault when they appear at different line numbers. The other fault might trigger some of the faults at different lines. Until the day the study was conducted, identifying statements in a buggy code that should be considered as defective among the highest suspiciousness statement (Pearson et al., 2017) still needs in-depth study. In some cases, it is crucial to inform software developers which fault they need to look first among the other faults. Therefore, they will not waste their time fixing the wrong fault. In code coverage, suspiciousness formula is used to

identify fault locations inside a program (Abreu et al., 2006; S. Ali et al., 2009). However, determining the fault location alone in a case of multiple faults is not enough as the formula can only highlight line numbers considered faulty. In a case where a line contains more than one operator, this formula is not suitable. Suspiciousness formula is unable to identify which operators are faulty as the technique can only highlight line numbers. Therefore, a search technique is needed to enable suspiciousness formula to perform analysis on all faulty operators either on the same line or different line number.

In the same light, multiple faults problem might be more complicated in object-oriented setting. As described by Mark Stefik, object-oriented is made up of multiple classes and methods (Stefik & Bobrow, 1985). Each of the class can interact through class objects. If we observe the interaction between classes, the object will create relationship and dependency between classes and methods. Indeed, a nested dependency is created if more than one class is involved. Eventhough object-oriented is made up of various components and better features compared to structured approach (Booch et al., 2008), it still has problems related with logical errors, operators, and programmatic styles (Basso et al., 2009). This paper also suggests that the programmer tends to commit the same mistake either on structured or object-oriented approach. Considering the three problems related to object-oriented which are logical errors, operators, and programmatic styles as suggested by Basso et al., it seems that all of these are underpinned by operators. This can be explained by the wide usage of operators in logical operations (comparison) and program flow directive (condition, counter, looping, etc). Likewise, programmers' programmatic syles also relate to operators such as function, algorithm, checking, and interface. Hence, operators are suggested in the investigation of multiple fault localization for both structured and object-oriented programming.

1.3 Research Objectives

The general objective of this study is to improve the existing multiple fault localization technique. This is done by considering both structured and object-oriented programming. To achieve this general objective, three sub-objectives have been outlined:

i. To propose a technique that can analyze fault dependency in multiple fault localization.
ii. To propose a technique that can localize multiple faults.
iii. To empirically evaluate the effectiveness of the proposed technique against an existing technique when dealing with multiple faults at similar and different locations.

1.4 Research Scope

The scope of this study is limited to:

- Object-oriented framework
  This study is focused on the improvement of fault localization technique in the object-oriented approach. For this purpose, this study only covers faults that appear in the method level and fault dependency between classes, methods, statements, and operators. Other faults not stated are not covered and can be included in future work.
- Software testing
  In software testing, several activities can be executed by the software tester to identify a fault location such as unit testing, integration testing, system testing, interface testing, regression testingand others. However, this research only covers two testing activities, which are unit and integration testing. These two testing activities were chosen based on their significance with code compared to another activity, which is closer to design. These two activities are also synonymous with faults that appear in the source code during testing.
- Fault type
  In this study, the focused types of error are faults that a developer commonly does in a method when writing a program. For this purpose, seven categories of faults are covered, which involve logical error, syntax error, and assignment error. They are as follows:
    - Arithmetic Operator Replacement (AOR).
    - Relational Operator Replacement (ROR).
    - Conditional Operator Replacement (COR).
    - Assignment Operator Replacement (AOR).
    - Statement Deletion.
    - Replacement of Boolean.
    - Absolute Value Insertion.

  This study also will cover on the fault dependency between the operators in the categories above. A combination of the operators will be studied in order to identify location of faults inside a program.

- Multiple fault localization
  Generally, fault localization is divided into two categories, which are single and multiple fault localization. In this study, the focus is on multiple fault localization. Three characteristics of a fault considered in this study are as follows:
    - Two or more faults that appear at different line numbers.
    - Two or more faults that appear at the same line but at a different location.
    - Any combination of faults either they appear at the same line or at different line numbers.

1.5 Research Contributions

This thesis has made the following contributions:

- It defined a new technique of localizing multiple faults by analyze the location of the fault and its dependencies with another faults. By implemented fault dependency analysis, the technique is able to locate the fault location for each behavior below:
    - Multiple faults at the same line number.
    - Multiple faults at different line numbers.
- It defined a technique to retrieve location accurately. Instead of providing line number as the output, the technique is able to provide details of the fault such as operators, row, and column of the fault location.
- It provided empirical evidence that the proposed technique can be effective and accurate in localizing multiple faults compared to the current approach.

1.6 Thesis Organization

This thesis contains six chapters, including the introductory chapter. In the introductory chapter, it explains in detail the problem statement, objective of the study, scope of the study, and thesis organization.

The literature review is described in detail in Chapter Two. Background study of software testing and maintenance is detailed out. Based on software testing and maintenance, the study focuses on fault localization. Also, this chapter lists all the suspiciousness formulas and identifies the formulas used in previous studies. Several potential techniques are described in this chapter as well. Finally, this chapter highlights issues for the relevant literature.

Chapter Three describes the methodology used throughout the study. Generally, this chapter describes all the activities involved in data collection, finding a suitable method to be implemented in the proposed technique, and preparation of performing data analysis.

The fourth chapter proposes a new multiple fault localization technique, which combines local search and a genetic algorithm called HGAMFL. In this chapter, the proposed technique is explained, including the flow and the algorithm. The implementation of the proposed technique is also described in this chapter.

The fifth chapter describes the evaluation planning and experiment execution for this research. The results of the analysis and interpretation are also presented in this chapter.

The sixth chapter presents the conclusion and future work of this research. In general, the conclusions are explained and future work for this research are also listed.

REFERENCES

Abreu, R., Zoeteweij, P., & Gemund, A. J. C. van. (2009a). Localizing Software Faults Simultaneously. *2009 Ninth International Conference on Quality Software*, 367–376. https://doi.org/10.1109/QSIC.2009.55

Abreu, R., Zoeteweij, P., & Gemund, A. J. C. van. (2009b). Spectrum-Based Multiple Fault Localization. *2009 IEEE/ACM International Conference on Automated Software Engineering*, 88–99. https://doi.org/10.1109/ASE.2009.25

Abreu, R., Zoeteweij, P., & van Gemund, A. J. ~C. (2008). A Dynamic Modeling Approach to Software Multiple-Fault Localization. *Proceedings of the 19th International Workshop on Principles of Diagnosis (DX'08)*, 7–14.

Abreu, R., Zoeteweij, P., & van Gemund, A. J. C. (2011). Simultaneous debugging of software faults. *Journal of Systems and Software*, *84*(4), 573–586. https://doi.org/10.1016/j.jss.2010.11.915

Abreu, R., Zoeteweij, P., & van Gemund, A. J. C. (2007). On the Accuracy of Spectrum-based Fault Localization. *Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION (TAICPART-MUTATION 2007)*, 89–98. https://doi.org/10.1109/TAICPART.2007.4344104

Abreu, R., Zoeteweij, P., & Van Gemund, A. J. C. (2006). An Evaluation of Similarity Coefficients for Software Fault Localization. *2006 12th Pacific Rim International Symposium on Dependable Computing (PRDC'06)*, 39–46. https://doi.org/10.1109/PRDC.2006.18

Adamopoulos, K., Harman, M., & Hierons, R. M. R. (2004). How to Overcome the Equivalent Mutant Problem and Achieve Tailored Selective Mutation Using Co-evolution. In K. Deb (Ed.), *Genetic and Evolutionary Computation -- GECCO 2004: Genetic and Evolutionary Computation Conference, Seattle, WA, USA, June 26-30, 2004. Proceedings, Part II* (Vol. 5, Issue 3, pp. 1338–1349). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-24855-2_155

Agrawal, H., Horgan, J. R., London, S., & Wong, W. E. (1995). Fault localization using execution slices and dataflow tests. *Proceedings of Sixth International Symposium on Software Reliability Engineering. ISSRE'95*, 143–151. https://doi.org/10.1109/ISSRE.1995.497652

Agrawal, Hiralal, & Horgan, J. R. (1990). Dynamic program slicing. *ACM SIGPLAN Notices*, *25*(6), 246–256. https://doi.org/10.1145/93548.93576

Ali, M., & Fairouz, T. (2015). Software Testing Concepts and Operations. In *John Wiley & Sons: Vol. XXXIII* (Issue 2). https://doi.org/10.1007/s13398-014-0173-7.2

Ali, S., Andrews, J. H., Dhandapani, T., & Wang, W. (2009). Evaluating the Accuracy of Fault Localization Techniques. *2009 IEEE/ACM International Conference on Automated Software Engineering*, 76–87. https://doi.org/10.1109/ASE.2009.89

Amari, S. (1977). Dynamics of pattern formation in lateral-inhibition type neural fields. *Biological Cybernetics*, *27*(2), 77–87. https://doi.org/10.1007/BF00337259

Aranda, J., & Venolia, G. (2009). The secret life of bugs: Going past the errors and omissions in software repositories. *2009 IEEE 31st International Conference on Software Engineering*, 298–308. https://doi.org/10.1109/ICSE.2009.5070530

Askarunisa, A., Manju, T., & Babu, B. G. (2012). Fault Localization for Java Programs using Probabilistic Program Dependence Graph. *International Journal of Computer Science Issues*, *8*(6 6-2), 224–232. http://arxiv.org/abs/1201.3985

Assiri, F. Y., & Bieman, J. M. (2014). Fault Localization for Automated Program Repair: Effectiveness and Performance. *SOFTWARE TESTING, VERIFICATION AND RELIABILITY Softw. Test. Verif. Reliab. 2014; 00:1–17 Fault*, *Volume 21*(Issue 1), 1–17. https://doi.org/10.1002/stvr

Basili, V. R., & Perricone, B. T. (1983). Software errors and complexity: An empirical investigation. *NASA Goddard Space Flight Center Collected Software Engineering Papers*, *2*, 24.

Basso, T., Moraes, R., Sanches, B. P., & Jino, M. (2009). *An Investigation of Java Faults Operators Derived from a Field Data Study on Java Software Faults* (Issue May 2014, pp. 1–13).

Bennett, K. H., & Rajlich, V. T. (2000). Software maintenance and evolution: a Roadmap K. *Proceedings of the Conference on The Future of Software Engineering - ICSE '00*, 73–87. https://doi.org/10.1145/336512.336534

Bertolino, A. (2007). Software Testing Research: Achievements, Challenges, Dreams. *Future of Software Engineering (FOSE '07)*, *September*, 85–103. https://doi.org/10.1109/FOSE.2007.25

Booch, G., Maksimchuk, R. A., Engle, M. W., Young, B. J., Connallen, J., & Houston, K. A. (2008). Object-oriented analysis and design with applications, third edition. *ACM SIGSOFT Software Engineering Notes*, *33*(5), 29. https://doi.org/10.1145/1402521.1413138

Bose, I., & Mahapatra, R. K. (2001). Business data mining - A machine learning perspective. *Information and Management*, *39*(3), 211–225. https://doi.org/10.1016/S0378-7206(01)00091-X

Cellier, P., Ducassé, M., Ferré, S., & Ridoux, O. (2008). Formal Concept Analysis Enhances Fault Localization in Software. In *Formal Concept Analysis: Vol. 4933 LNAI* (pp. 273–288). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-78137-0_20

Chapin, N., Hale, J. E., Khan, K. M., Ramil, J. F., Tan, W.-G., amd Khaled Md. Khan, J. E. H., Ramil, J. F., & Tan, W.-G. (2001). Types of software evolution and software maintenance. *Journal of Software Maintenance and Evolution: Research and Practice*, *13*(1), 3–30. https://doi.org/10.1002/smr.220

Chen, C. T., Cheng, Y. C., Hsieh, C. Y., & Wu, I. L. (2009). Exception handling refactorings: Directed by goals and driven by bug fixing. *Journal of Systems and Software*, *82*(2), 333–345. https://doi.org/10.1016/j.jss.2008.06.035

Chen, G. (2011). *Learning efficient software fault localization via genetic programming* (Issue March) [Chalmers University of Technology]. http://publications.lib.chalmers.se/records/fulltext/156589.pdf

Choi, J.-D., & Ferrante, J. (1994). Static slicing in the presence of goto statements. *ACM Transactions on Programming Languages and Systems*, *16*(4), 1097–1113. https://doi.org/10.1145/183432.183438

Coe, R. (2002). It ' s the Effect Size , Stupid: What effect size is and why it is important. *British Educational Research Association*, 1–18. http://www.cem.org/attachments/ebe/ESguide.pdf

Dallmeier, V., Lindig, C., & Zeller, A. (2005). Lightweight bug localization with AMPLE. *Proceedings of the Sixth Sixth International Symposium on Automated Analysis-Driven Debugging - AADEBUG'05*, 99–104. https://doi.org/10.1145/1085130.1085143

Debroy, V., & Wong, W. E. (2014). Combining mutation and fault localization for automated program debugging. *Journal of Systems and Software*, *90*(1), 45–60. https://doi.org/10.1016/j.jss.2013.10.042

Debroy, V., & Wong, W. E. (2009). Insights on Fault Interference for Programs with Multiple Bugs. *2009 20th International Symposium on Software Reliability Engineering*, 165–174. https://doi.org/10.1109/ISSRE.2009.14

Delahaye, M., Briand, L. C., Gotlieb, A., & Petit, M. (2012). Mutation-based Statistical Test Inputs Generation for Automatic Fault Localization Micka¨. *2012 IEEE Sixth International Conference on Software Security and Reliability*, 197–206. https://doi.org/10.1109/SERE.2012.32

Demott, J. D. (2012). *Enhancing Automated Fault Discovery and Analysis*. Michigan State University.

DiGiuseppe, N., & Jones, J. a. (2015). Fault density, fault types, and spectra-based fault localization. *Empirical Software Engineering*, *20*(4), 928–967. https://doi.org/10.1007/s10664-014-9304-1

DiGiuseppe, N., & Jones, J. A. (2011). On the influence of multiple faults on coverage-based fault localization. *Proceedings of the 2011 International Symposium on Software Testing and Analysis - ISSTA '11*, 210. https://doi.org/10.1145/2001420.2001446

Dinh-Trong, T., & Bieman, J. M. (2004). Open source software development: a case study of FreeBSD. *10th International Symposium on Software Metrics, 2004. Proceedings.*, 96–105. https://doi.org/10.1109/METRIC.2004.1357894

Erdweg, S., Fehrenbach, S., & Ostermann, K. (2014). Evolution of Software Systems with Extensible Languages and DSLs. *IEEE Software*, *31*(5), 68–75. https://doi.org/10.1109/MS.2014.99

Eric Wong, W., Debroy, V., & Choi, B. (2010). A family of code coverage-based heuristics for effective fault localization. *Journal of Systems and Software*, *83*(2), 188–208. https://doi.org/10.1016/j.jss.2009.09.037

Fayad, M., & Schmidt, D. C. (1997). Object-oriented application frameworks. *Communications of the ACM*, *40*(10), 32–38. https://doi.org/10.1145/262793.262798

Foster, K. R., Koprowski, R., & Skufca, J. D. (2014). Machine learning, medical diagnosis, and biomedical engineering research - commentary. *BioMedical Engineering OnLine*, *13*(1), 94. https://doi.org/10.1186/1475-925X-13-94

Fowler, M., Beck, K., Brant, J., & Opdyke, W. (2002). *Refactoring: Improving the Design of Existing Code*.

Fuchs, S., Williams-Jones, A. E., & Przybylowicz, W. J. (2016). The origin of the gold and uranium ores of the Black Reef Formation, Transvaal Supergroup, South Africa. *Ore Geology Reviews*, *72*(P1), 149–164. https://doi.org/10.1016/j.oregeorev.2015.07.010

Gao, M., Li, P., Chen, C., & Jiang, Y. (2018). Research on Software Multiple Fault Localization Method Based on Machine Learning. *MATEC Web of Conferences*, *232*, 01060. https://doi.org/10.1051/matecconf/201823201060

Golafshani, N. (2003). *Understanding Reliability and Validity in Qualitative Research*. *8*(4), 597–606.

Gong, Dandan, Su, X., Wang, T., Ma, P., & Yu, W. (2015). State dependency probabilistic model for fault localization. *Information and Software Technology*, *57*, 430–445. https://doi.org/10.1016/j.infsof.2014.05.022

Gong, Dunwei, & Zhang, Y. (2013). Generating test data for both path coverage and fault detection using genetic algorithms. *Frontiers of Computer Science*, *7*(6), 822–837. https://doi.org/10.1007/s11704-013-3024-3

Hailpern, B., & Santhanam, P. (2002). Software debugging, testing, and verification. *IBM Systems Journal*, *41*(1), 4–12. https://doi.org/10.1147/sj.411.0004

Jonathon, A. (2018). *How Software-as-a-Service (SaaS) Speeds Up Innovation Cycles*.

Jones, J. A., & Harrold, M. J. (2005). Empirical evaluation of the tarantula automatic fault-localization technique. *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering - ASE '05*, 273. https://doi.org/10.1145/1101908.1101949

Jones, K. O., & Boizanté, G. (2011). Comparison of Firefly algorithm optimisation, particle swarm optimisation and differential evolution. *Proceedings of the 12th International Conference on Computer Systems and Technologies - CompSysTech '11*, 191. https://doi.org/10.1145/2023607.2023640

Ju, X., Jiang, S., Chen, X., Wang, X., Zhang, Y., & Cao, H. (2014). HSFal: Effective fault localization using hybrid spectrum of full slices and execution slices. *Journal of Systems and Software*, *90*(1), 3–17. https://doi.org/10.1016/j.jss.2013.11.1109

Juristo, N., & Vegas, S. (2003). Functional Testing, Structural Testing and Code Reading: What Fault Type Do They Each Detect? In *Empirical Methods and Studies in Software Engineering Experiences from ESERNET* (Vol. 2785, Issue 12, pp. 208–232). https://doi.org/10.1007/978-3-540-45143-3_12

Kampstra, P. (2008). Beanplot: A Boxplot Alternative for Visual Comparison of Distributions. *Journal of Statistical Software*, *28*(Code Snippet 1), 1–9. https://doi.org/10.18637/jss.v028.c01

Kessentini, M., Kessentini, W., Sahraoui, H., Boukadoum, M., & Ouni, A. (2011). Design Defects Detection and Correction by Example. *2011 IEEE 19th International Conference on Program Comprehension*, 81–90. https://doi.org/10.1109/ICPC.2011.22

Knapp, G. M., & Wang, H.-P. (Ben). (1992). Machine fault classification: a neural network approach. *International Journal of Production Research*, *30*(4), 811–823. https://doi.org/10.1080/00207543.1992.9728458

Ko-Li Cheng, Ching-Pao Chang, & Chih-Ping Chu. (2011). Software fault detection using program patterns. *2011 IEEE 2nd International Conference on Software Engineering and Service Science*, 278–281. https://doi.org/10.1109/ICSESS.2011.5982308

Korel, B., & Rilling, J. (1998). Dynamic program slicing methods. *Information and Software Technology*, *40*(11–12), 647–659. https://doi.org/10.1016/S0950-5849(98)00089-5

Land, M. (1998). *Evolutionary algorithms with local search for combinatorial optimization*. University of California, San Diego.

Laporte, C. Y., & April, A. (2017). Software Quality Assurance. In *Infection and Immunity* (Vol. 24, Issue 1). John Wiley & Sons, Inc. https://doi.org/10.1002/9781119312451

Le Goues, C., Nguyen, T., Forrest, S., & Weimer, W. (2012). GenProg: A Generic Method for Automatic Software Repair. *IEEE Transactions on Software Engineering*, *38*(1), 54–72. https://doi.org/10.1109/TSE.2011.104

Li, B., Shen, B., Wang, J., Chen, Y., Zhang, T., & Wang, J. (2014). A Scenario-Based Approach to Predicting Software Defects Using Compressed C4.5 Model. *2014 IEEE 38th Annual Computer Software and Applications Conference*, 406–415. https://doi.org/10.1109/COMPSAC.2014.64

Li, Q., & Clifford, G. D. (2012). Dynamic time warping and machine learning for signal quality assessment of pulsatile signals. *Physiological Measurement*, *33*(9), 1491. http://stacks.iop.org/0967-3334/33/i=9/a=1491

Lientz, B. P., Swanson, E. B., & Tompkins, G. E. (1978). Characteristics of application software maintenance. *Communications of the ACM*, *21*(6), 466–471. https://doi.org/10.1145/359511.359522

M. Nasrabadi, N. (2007). Pattern Recognition and Machine Learning. *Journal of Electronic Imaging*, *16*(4), 049901. https://doi.org/10.1117/1.2819119

Ma, Y.-S., Offutt, J., & Kwon, Y.-R. (2006). MuJava. *Proceeding of the 28th International Conference on Software Engineering - ICSE '06*, 827. https://doi.org/10.1145/1134285.1134425

Mala, D. J., Ruby, E., & Mohan, V. (2010). a Hybrid Test Optimization Framework – Coupling Genetic Algorithm With Local Search Technique. *Computing and Informatics*, *29*(1), 133–164.

Malhotra, R., & Chug, A. (2016). Software Maintainability: Systematic Literature Review and Current Trends. *International Journal of Software Engineering and Knowledge Engineering*, *26*(08), 1221–1253. https://doi.org/10.1142/S0218194016500431

Martínez, Y., Cachero, C., & Meliá, S. (2013). Empirical study on the maintainability of Web applications: Model-driven Engineering vs Code-centric. *Empirical Software Engineering*, 1–34. https://doi.org/10.1007/s10664-013-9269-5

Masri, W., Abou-Assi, R., El-Ghali, M., & Al-Fatairi, N. (2009). An empirical study of the factors that reduce the effectiveness of coverage-based fault localization. *Proceedings of the 2nd International Workshop on Defects in Large Software Systems Held in Conjunction with the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2009) - DEFECTS '09*, 1. https://doi.org/10.1145/1555860.1555862

McConnell, S. C. (2004). *Code Complete, Second Edition* (Vol. 136, Issue 1). Microsoft Press. https://doi.org/10.1039/c0an90005b

McCusker, K., & Gunaydin, S. (2015). Research using qualitative, quantitative or mixed methods and choice based on the research. *Perfusion*, *30*(7), 537–542. https://doi.org/10.1177/0267659114559116

Mens, T., Wermelinger, M., Ducasse, S., Demeyer, S., Hirschfeld, R., & Jazayeri, M. (2005). Challenges in Software Evolution. *Eighth International Workshop on Principles of Software Evolution (IWPSE'05)*, *2005*, 13–22. https://doi.org/10.1109/IWPSE.2005.7

Mohapatra, S. K., & Prasad, S. (2013). Evolutionary Search Algorithms for Test Case Prioritization. *2013 International Conference on Machine Intelligence and Research Advancement*, 115–119. https://doi.org/10.1109/ICMIRA.2013.29

Moon, S., Kim, Y., Kim, M., & Yoo, S. (2014). Ask the Mutants: Mutating Faulty Programs for Fault Localization. *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation*, 153–162. https://doi.org/10.1109/ICST.2014.28

Müller, K.-R., Tangermann, M., Dornhege, G., Krauledat, M., Curio, G., & Blankertz, B. (2008). Machine learning for real-time single-trial EEG-analysis: From brain–computer interfacing to mental state monitoring. *Journal of Neuroscience Methods*, *167*(1), 82–90. https://doi.org/10.1016/j.jneumeth.2007.09.022

Myers, G. J. (2004). The art of software testing, Second Edition. In *... 1991., Proceedings of the IEEE 1991 National*.

Naish, L., Lee, H. J., & Ramamohanarao, K. (2011). A model for spectra-based software diagnosis. *ACM Transactions on Software Engineering and Methodology*, *20*(3), 1–32. https://doi.org/10.1145/2000791.2000795

Neill, J. (2007). *Qualitative versus Quantitative Research: Key Points in a Classic Debate*. Netcraft.

Nguyen, T. T., Nguyen, H. A., Pham, N. H., Al-Kofahi, J., & Nguyen, T. N. (2010). Recurring bug fixes in object-oriented programs. *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - ICSE '10*, *1*, 315. https://doi.org/10.1145/1806799.1806847

Nidhra, S., & Dondeti, J. (2012). Black Box and White Box Testing Techniques. *International Journal of Embedded Systems and Applications (IJESA)*, *2*(2), 29–50.

Nosek, J. T., & Palvia, P. (1990). Software maintenance management: Changes in the last decade. *Journal of Software Maintenance: Research and Practice*, *2*(3), 157–174. https://doi.org/10.1002/smr.4360020303

Nuseibeh, B., & Easterbrook, S. (2000). Requirements engineering. *Proceedings of the Conference on The Future of Software Engineering - ICSE '00*, *41*(4), 35–46. https://doi.org/10.1145/336512.336523

O'Brien, M. P. (2007). Evolving a Model of the Information-Seeking Behaviour of Industrial Programmers. In *Computer Science: Vol. Ph.D.* University of Limerick.

Ouni, A., Kessentini, M., Sahraoui, H., & Boukadoum, M. (2013). Maintainability defects detection and correction: a multi-objective approach. *Automated Software Engineering*, *20*(1), 47–79. https://doi.org/10.1007/s10515-011-0098-8

P. Alexandre, A. Rui, W. E. W. (2014). *A Survey on Fault Localization Techniques*. *1*(2), 171–186. https://doi.org/10.1.1.167.966

Papadakis, M., & Le Traon, Y. (2014). Effective fault localization via mutation analysis. *Proceedings of the 29th Annual ACM Symposium on Applied Computing - SAC '14*, 1293–1300. https://doi.org/10.1145/2554850.2554978

Pargas, R. P., Harrold, M. J., & Peck, R. R. (1999). Test-data generation using genetic algorithms. *Journal of Software Testing, Verification and Reliability*, *282*(January), 263–282. http://profs.info.uaic.ro/~ogh/files/sbse/articles/sbse-articles-testing/pargas99testdata.pdf

Patton, R. (2001). Software Testing. In *Sams Publishing*. http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:No+Title#0

Pearson, S., Campos, J., Just, R., Fraser, G., Abreu, R., Ernst, M. D., Pang, D., & Keller, B. (2017). Evaluating and Improving Fault Localization. *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), August 2016*, 609–620. https://doi.org/10.1109/ICSE.2017.62

Perez, A., Abreu, R., & Riboira, A. (2014). A dynamic code coverage approach to maximize fault localization efficiency. *Journal of Systems and Software*, *90*(1), 18–28. https://doi.org/10.1016/j.jss.2013.12.036

Prakash, B. V. A., Ashoka, D. V, & Aradya, V. N. M. (2015). Application of Data Mining Techniques for Defect Detection and Classification. In S. C. Satapathy, B. N. Biswal, S. K. Udgata, & J. K. Mandal (Eds.), *Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA) 2014* (Vol. 327). Springer International Publishing. https://doi.org/10.1007/978-3-319-11933-5

Putano, B. (2017). *Most Popular and Influential Programming Languages of 2018*. https://stackify.com/popular-programming-languages-2018/

Rangwala, S. S., & Dornfeld, D. A. (1989). Learning and Optimization of Machining Operations Using Computing Abilities of Neural Networks. *IEEE Transactions on Systems, Man and Cybernetics*, *19*(2), 299–314. https://doi.org/10.1109/21.31035

Rawat, M. S., & Dubey, S. K. (2012). Software defect prediction models for quality improvement: A literature study. *International Journal of Computer Science Issues*, *9*(5 5-2), 288–296.

Rodríguez, D., Ruiz, R., Riquelme, J. C., & Aguilar–Ruiz, J. S. (2012). Searching for rules to detect defective modules: A subgroup discovery approach. *Information Sciences*, *191*, 14–30. https://doi.org/10.1016/j.ins.2011.01.039

Rutenbar, R. a. (1989). Simulated annealing algorithms: An overview. In *IEEE Circuits and Devices Magazine* (Vol. 5, Issue 1, pp. 19–26). https://doi.org/10.1109/101.17235

Sahoo, S. K. (2012). *A Novel Invariants-Based Approach For Automated Software Fault Localization*. University of Illinois at Urbana-Champaign.

Salas, E., Rosen, M. A., & DiazGranados, D. (2010). Expertise-Based Intuition and Decision Making in Organizations. *Journal of Management*, *36*(4), 941–973. https://doi.org/10.1177/0149206309350084

Santelices, R., Jones, J. a., Yanbing Yu, & Harrold, M. J. (2009). Lightweight fault-localization using multiple coverage types. *2009 IEEE 31st International Conference on Software Engineering*, 56–66. https://doi.org/10.1109/ICSE.2009.5070508

Schneidewind, N. F. (1987). The State of Software Maintenance. *IEEE Transactions on Software Engineering*, *SE-13*(3), 303–310. https://doi.org/10.1109/TSE.1987.233161

Sharif, K. Y. (2012). *Observing Open Source Programmers' Information Seeking* (Issue June). University of Limerick.

Simon, H. A. (1959). Theories of Decision-Making in Economics and Behavioral Science. *The American Economic Review*, *49*(3), 253–283. http://www.jstor.org/stable/1809901

Srinivas, M., & Patnaik, L. M. (1994). Genetic algorithms: a survey. *Computer*, *27*(6), 17–26. https://doi.org/10.1109/2.294849

Stack Overflow. (2016). *Developer Hiring Landscape 2016 Global Report*.

Stefik, M., & Bobrow, D. G. (1985). Object-Oriented Programming: Themes and Variations. *The AI MagazineAI Magazine*, *6*(4), 40–62. https://doi.org/10.1609/aimag.v6i4.508

Stephen R., S. (2010). *Object-Oriented and Classical Software Engineering* (8th ed.). The McGraw-Hill Companies, Inc.

Sun, C., Zhai, Y. M., Shang, Y., & Zhang, Z. (2013). BPELDebugger: An effective BPEL-specific fault localization framework. *Information and Software Technology*, *55*(12), 2140–2153. https://doi.org/10.1016/j.infsof.2013.07.009

Swanson, E. B. (1976). The Dimensions of Maintenance. *Proceedings of the 2nd International Conference on Software Engineering*, 492–497. https://doi.org/10.1017/CBO9781107415324.004

Venkatasubramanian, V., & Chan, K. (1989). A neural network methodology for process fault diagnosis. *AIChE Journal*, *35*(12), 1993–2002. https://doi.org/10.1002/aic.690351210

Weiser, M. (1984). Program Slicing. *IEEE Transactions on Software Engineering*, *SE-10*(4), 352–357. https://doi.org/10.1109/TSE.1984.5010248

Wong, W. E., & Debroy, V. (2009). Software Fault Localization. *Technology*, 1–6.

Wong, W. E., Debroy, V., Gao, R., & Li, Y. (2014). The DStar Method for Effective Software Fault Localization. *IEEE Transactions on Reliability*, *63*(1), 290–308. https://doi.org/10.1109/TR.2013.2285319

Wong, W. E., Gao, R., Li, Y., Abreu, R., & Wotawa, F. (2016). A Survey on Software Fault Localization. *IEEE Transactions on Software Engineering*, *42*(8), 707–740. https://doi.org/10.1109/TSE.2016.2521368

Wong, W. E., & Qi, Y. (2006). Effective program debugging based on execution slices and inter-block data dependency. *Journal of Systems and Software*, *79*(7), 891–903. https://doi.org/10.1016/j.jss.2005.06.045

Wong, W. E., & Qi, Y. (2009). BP Neural Network-Based Effective Fault Localization. *International Journal of Software Engineering and Knowledge Engineering*, *19*(04), 573–597. https://doi.org/10.1142/S021819400900426X

Xuan, J., & Monperrus, M. (2014a). Learning to Combine Multiple Ranking Metrics for Fault Localization. *2014 IEEE International Conference on Software Maintenance and Evolution*, 191–200. https://doi.org/10.1109/ICSME.2014.41

Xuan, J., & Monperrus, M. (2014b). Test case purification for improving fault localization. *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2014*, 52–63. https://doi.org/10.1145/2635868.2635906

Xue, X., & Namin, A. S. (2013). How significant is the effect of fault interactions on coverage-based fault localizations? *International Symposium on Empirical Software Engineering and Measurement*, 113–122. https://doi.org/10.1109/ESEM.2013.22

Yu, K., Lin, M., Gao, Q., Zhang, H., & Zhang, X. (2011). Locating faults using multiple spectra-specific models. *Proceedings of the 2011 ACM Symposium on Applied Computing - SAC '11*, 1404. https://doi.org/10.1145/1982185.1982490

Zeller, A., & Hildebrandt, R. (2002). Simplifying and isolating failure-inducing input. *IEEE Transactions on Software Engineering*, *28*(2), 183–200. https://doi.org/10.1109/32.988498

Zhang, L., Zhou, W., & Jiao, L. (2004). Wavelet Support Vector Machine. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, *34*(1), 34–39. https://doi.org/10.1109/TSMCB.2003.811113

Zhang, P., Mao, X., Lei, Y., & Zhang, Z. (2014). Fault localization based on dynamic slicing via JSlice for Java programs. *2014 IEEE 5th International Conference on Software Engineering and Service Science*, 565–568. https://doi.org/10.1109/ICSESS.2014.6933631

Zhao, L., & Hayes, J. H. (2011). Rank-based refactoring decision support: Two studies. *Innovations in Systems and Software Engineering*, *7*(3), 171–189. https://doi.org/10.1007/s11334-011-0154-3

BIODATA OF STUDENT

Muhammad Luqman bin Mahamad Zakaria was born on 16th Mac 1990 in Subang Jaya Medical Centre, Malaysia. He obtained his primary school study at SK Bandar Tun Hussein Onn from 1997-2002. He move further to secondary school at SMK Bandar Tun Hussein Onn (2) from 2003- 2007.

He was enrolled n Universiti Tun Hussein Onn Malaysia at 2008 and graduated with Diploma in Information Technology in 2011. He then further his Bachelor Degree at the same university on 2011 and graduated in Bachelor of Information technology in 2013. During his study, he had received Chancelor Award during Diploma and Bachelor graduation.

In 2014, he was enrolled as a student at Universiti Putra Malaysia, where he is currently pursuing his PhD degree in Software Engineering.

LIST OF PUBLICATIONS

Zakaria, M. L. M., Sharif, K. Y., Ghani, A. A. A., Wei, K. T., & Zulzalil, H. (2016). fault localization by using hybrid genetic algorithm. 4TH International Conference e-proceeding of the 3rd World Conferences Artificial Intelligence & Computer Science 2016.

Zakaria, M. L. M., Sharif, K. Y., Ghani, A. A. A., Wei, K. T., & Zulzalil, H. (2018). Hybrid Genetic Algorithm for Improving Fault Localization. Advanced Science Letters, 24(3), 1587–1590. https://doi.org/10.1166/asl.2018.11115

Mahamad Zakaria, M. L., Sharif, K. Y., Abd. Ghani, A. A., Koh, T. W., & Zulzalil, H. (2018). Finding multiple fault by using Hybrid genetic Algorithm. 2018 Global Conference on Engineering and Applied Science (GCEAS).