UNIVERSITI PUTRA MALAYSIA

SLICING ASPECT-ORIENTED PROGRAM USING ASPECT- ORIENTED DEPENDENCE FLOW GRAPH FOR SOFTWARE MAINTENANCE

SYARBAINI AHMAD

FSKTM 2016 47

SLICING ASPECT-ORIENTED PROGRAM USING ASPECT- ORIENTED DEPENDENCE FLOW GRAPH FOR SOFTWARE MAINTENANCE

By

SYARBAINI AHMAD

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

October 2016

DEDICATION

وَإِذَاقِيلَ انْشُزُوافَانْشُزُوا يَرْفَعِ

اللهُ الذِيْنَ امَنُوا مِنْكُمْ وَالّذِيْنَ أُوتُو الْعِلْمَ دَرَجَـتٍ وَاللهُ بِمَا تَعْمَلُـوْنَ خَـبِيْـر

And when you are told to rise up, rise up. Allah will exalt in degree those of you who believe, and those who have been granted knowledge. And Allah is Well-Acquainted with what you do.

[Al-Mujadalah 58:11]

Dedicated to my parent;

to my wife and kids;

to my brothers;

to my family;

Thank you for make me stronger on my journey

Abstract of thesis presented to the Senate of Universiti Putra Malaysia in fulfillment of the requirement for the Degree of Doctor of Philsophy

SLICING ASPECT-ORIENTED PROGRAM USING ASPECT- ORIENTED DEPENDENCE FLOW GRAPH FOR SOFTWARE MAINTENANCE

By

SYARBAINI AHMAD

October 2016

Chairman : Professor Abdul Azim Abdul Ghani, PhD
Faculty : Computer Science and Information Technology

Program slicing is useful for debugging, testing, and maintaining software systems due to availability of information about the structure and relationship of the program modules. In general, program slicing can be performed either based on control flow graph (CFG) or dependence graph (DG). However, in the case of aspect-oriented programming (AOP), aspect-oriented control flow graph (AOCFG) or aspect-oriented dependence graph (AODG) individually is not enough to model the features of Aspect-oriented (AO) programs. Thus, a suitable graph model for aspect-oriented program slicing is required to gather information on the structure of aspect-oriented programs.

In this thesis, the concept of slicing aspect-oriented programs for maintenance purpose is proposed. In order to aid in slicing an aspect-oriented program, a graph model known as Aspect-Oriented–Dependence Flow Graph (AODFG) is proposed to represent the structure of aspect-oriented programs. The graph is formed by merging AOCFG and AODG. As a consequence, more information about dependencies involving the features of AOP, such as join point, advice, aspects, their related constructs, and the flow of control are able to be gathered. Based on AODFG, slicing criteria are defined for aspect-oriented features. A prototype tool called Aspect-Oriented Slicing Tool (AOST) was developed to implement AODFG.

The prototype tool was evaluated for its applicability by checking the consistency of output by analysing ten AspectJ programs taken from AspectJ Development Tools. The analysis showed the outputs from the prototype are consistent with those from AODG and AOCFG. Furthermore, a one-shot experimental case study involving experts was conducted to find out the effect of AOST in terms of effectiveness, understandability, and modifiability for maintenance purpose. The results of the experiement show positive responses which are more than 85% of the experts says that AOST supports their understanding of the programs structure, helps in identifying aspect-oriented features, and effectively represents the program structure.

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

PENGHIRISAN PROGRAM BERORIENTASI ASPEK MENGGUNAKAN GRAF ALIRAN KEBERGANTUNGAN BERORIENTASI ASPEK UNTUK PENYELENGGARAAN PERISIAN

Oleh

SYARBAINI AHMAD

Oktober 2016

Pengerusi : Profesor Abdul Azim Abdul Ghani, PhD
Fakulti : Sains Komputer dan Teknologi Maklumat

Penghirisan program sangat berguna untuk proses nyahpijat, pengujian, dan penyelenggaraan sistem perisian disebabkan oleh ketersediaan maklumat mengenai struktur dan hubungan modul program. Secara umumnya, penghirisan program dapat dilakukan sama ada berdasarkan graf kawalan aliran atau graf kebergantungan. Walau bagaimanapun, dalam kes pengaturcaraan berorientasi aspek (AOP), graf kawalan aliran berorientasi aspek (AOCFG) atau graf kebergantungan berorientasi aspek (AODG) secara bersendirian tidak cukup untuk memodelkan ciri-ciri program berorientasi aspek (AO). Oleh itu, suatu model graf yang sesuai untuk penghirisan program berorientasi aspek diperlukan untuk mengumpul maklumat mengenai struktur program berorientasi aspek.

Dalam tesis ini, konsep penghirisan program berorientasi aspek untuk tujuan penyelenggaran dicadangkan. Sebagai bantuan dalam menghiris proram berorientasi aspek, satu model graf dikenali dengan *Aspect-Oriented Dependence Flow Graph* (AODFG) dicadangkan untuk mewakili struktur program berorientasi aspek. Graf tersebut dibentuk dengan menggabungkan AOCFG dan AODG. Hasilnya, lebih banyak maklumat mengenai kebergantungan yang melibatkan ciri-ciri AOP seperti *join point*, *advice*, *aspect*, binaan yang berkaitan, dan aliran kawalan dapat dikumpul. Berdasarkan AODFG, kriteria penghirisan ditakrif untuk ciri-ciri berorientasi aspek. Satu alatan prototaip dikenali sebagai *Aspect-Oriented Slicing Tool (AOST)* telah dibangunkan untuk mengimplemen AODFG.

Alatan prototaip tersebut telah dinilai kebolehgunaannya melalui penyemakan konsistensi output dengan menganalisis sepuluh program AspectJ yang diambil dari *AspectJ Development Tools*. Analisis menunjukkan output prototaip konsisten dengan output AODG dan AOCFG. Tambahan, satu kes kajian eksperimen bentuk tunggal melibatkan pakar telah dijalankan untuk mendapat tahu efek AOST terhadap

keefektifan, kebolehfahaman, dan kebolehubahan bagi tujuan penyelenggaraan. Keputusan eksperimen menunjukkan respon yang positif yang mana lebih daripada 85% pakar mengatakan bahawa AOST menyokong kefahaman mereka tentang struktur program, membantu dalam mengenalpasti ciri-ciri berorientasi aspek, dan secara efektif mewakili struktur program.

ACKNOWLEDGEMENTS

In the name of Allah, the Beneficent the Compassionate and who giving me strength, patience, and motivation to complete this study. This is the opportunity for me to present my gratitude towards the great peoples that patiently support during my study. Particularly Prof. Dr. Abdul Azim Abdul Ghani, the research supervisory committee leader, who has always spend his time to share his knowledge, invaluable guidance, fruitful discussion, gold recommendations and suggestions also encourage me continuously in every single stage of this study. Also a great thanks to my second supervisor Assoc. Prof. Dr Nor. Fazlida Mohd Sani for her support, attentions during my research work and the guidance in each discussion during all steps of this work. Also to Assoc. Prof. Dr. Rodziah Atan for her victorious guidance, encouragement and help during the time of doing the research. Thousands grateful to Mr. Fairol Zamzuri Che Sayuti for his standing up me during the hard times and his never ending support.

Next, my acknowledgement belong to International Islamic University College Selangor (KUIS) for funding my study. Finally, I would like to express my deepest gratitude to my family and friends for their lasting support and encouragement. This thesis would not have been possible without the foundation created by them. Thank you!

I certify that a Thesis Examination Committee has met on 13 October 2016 to conduct the final examination of Syarbaini bin Ahmad on his thesis entitled "Slicing Aspect-Oriented Program using Aspect- Oriented Dependence Flow Graph for Software Maintenance" in accordance with the Universities and University Colleges Act 1971 and the Constitution of the Universiti Putra Malaysia [P.U.(A) 106] 15 March 1998. The Committee recommends that the student be awarded the Doctor of Philosophy.

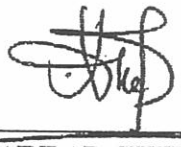
Members of the Thesis Examination Committee were as follows:

Masrah Azrifah binti Azmi Murad, PhD
Associate Professor
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
(Chairman)

Abu Bakar bin Md Sultan, PhD
Professor
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
(Internal Examiner)

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

Joao Baptista Da Silva Araujo Jr, PhD
Professor
Universidade Nova De Lisboa
Portugal
(External Examiner)



NOR AINI AB. SHUKOR, PhD
Professor and Deputy Dean
School of Graduate Studies
Universiti Putra Malaysia

Date: 26 January 2017

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:

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

Nor Fazlida Mohd Sani, PhD
Associate Professor
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
(Member)

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

BUJANG BIN KIM HUAT, PhD
Professor and Dean
School of Graduate Studies
Universiti Putra Malaysia

Date:

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: 13 October 2016

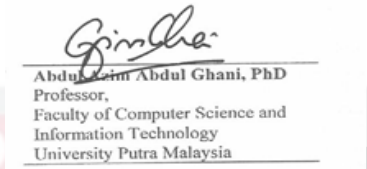
Name and Matric No.: Syarbaini Ahmad / GS22466

Declaration by Members of Supervisory Committee

This is to confirm that:

- the research conducted and the writing of this thesis was under our supervision;
- supervision responsibilities as stated in the Universiti Putra Malaysia (Graduate Studies) Rules 2003 (Revision 2012-2013) were adhered to.

Signature:
Name of Chairman of Supervisory Committee:

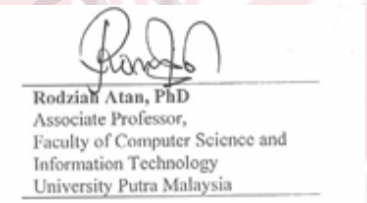


Abdul Azim Abdul Ghani, PhD
Professor,
Faculty of Computer Science and Information Technology
University Putra Malaysia

Signature:
Name of Member of Supervisory Committee:

Nor Fazlida Mohd Sani
Associate Professor,
Faculty of Computer Science and Information Technology
University Putra Malaysia

Signature:
Name of Member of Supervisory Committee:



Rodziah Atan, PhD
Associate Professor,
Faculty of Computer Science and Information Technology
University Putra Malaysia

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AJDT	AspectJ development tools
AO	aspect-oriented
AOCFG	aspect-oriented control flow graph
AODFG	aspect-oriented dependence flow graph
AODG	aspect-oriented dependence graph
AOP	aspect-oriented programming
AOST	aspect-oriented slicing tool
ASDG	aspect-oriented system dependence graph
CASDG	Concurrent Aspect-oriented System Dependence Graph
CFG	control flow graph
DADG	Dynamic Aspect-Oriented Dependence Graph
DDS	distributed dynamic slicing
DDST	Dynamic Dependence Slicing Tool
DFG	dependence flow graph
DG	dependence graph
DPDG	distributed program dependence graph
DU	define-use
EASDG	extended aspect-oriented system dependence graph
GUI	graphical user interface
IDE	integrated development environment
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineer
ISO	International Organization for Standardization
JDK	Java Development Kit
JRE	Java Runtime Environment

MtDG	multithreaded dependence graph
OO	object-oriented
OOP	object-oriented programming
PARC	Palo Alto Research Center
RUP	Rational Unified Process
SDG	system dependence graph
SDST	Static Dependence Slicing Tools
TBDS	trace File Based Dynamic Slicing
UD	use-define
UML	Unified Modelling Language

CHAPTER 1

INTRODUCTION

1.1 Background Study

Software maintenance is a modification of a software product after delivery, to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment (Grubb & Takang, 2003). The basic understanding about software maintenance is, it is not only about program. It is a temptation to think of activities carried out exclusively on programs such as source code, object code and documentation of any facet of the program such as requirements analysis, specifications, design, and user manuals. Software maintenance are most likely working with reverse engineering technique such as re-engineering, re-use, refactoring and slicing.

Maintainability of software system is something that is notoriously difficult to describe. Certain aspect may be easy to measure. But, practically there are many different ways to measure, especially for complex software. Different development method also will make the maintenance activity have a different way of measure. For example, the development for procedural method is different compare to object-oriented measurement. Even, object-oriented is also different compare to aspect-oriented.

Aspect-oriented (AO) is a post Object Oriented (OO) paradigm proposed by Kiczales et. al. (2001) with the goal of enhancing software maintainability through new modularization mechanisms for encapsulating crosscutting concern. It is very useful in software engineering to lead the reducing of complexity in the development cycle especially in the maintenance phase. Separation of concerns (Laddad, 2003) are able to identify, encapsulate and manipulate in isolated way only those parts of software that are relevant to a concept, objective or intention given. It is aimed at breaking the hegemony of the dominant decomposition (Colyer & Clement, 2005).

Slicing is a reverse engineering technique in software maintenance to extract a part of codes with respect to some special computation. It was first introduced by Weiser with the procedural programming in 1979 (Weiser, 1979). Slicing was first developed to facilitate debugging, but it is then found helpful in many aspects of the software development life cycle especially software maintenance. Since the last two decades there are many types of slicing have been presented (Xu, Qian,Zhang,Wu, & Chen, 2005) such as hybrid slicing, relevant slicing, union slicing etc. but the major focus of research in slicing are static and dynamic slicing. Ishio, Kusumoto, and Inoue (2004) claim that because of the features of aspect-oriented programming (AOP), debugging, testing, and verifying program can be more complex than with the traditional programming technique. In this domain, slicing can be more useful technique. It can be used to find the part of the program that affects the criteria, or is affected by the criteria.

Dependence flow graph (DFG) is a combination of two types of graph which are *dependence graph (DG)* and *control flow graph (CFG)*. DG is a directed graph normally used to represent dependencies of several objects towards each other. In OO (Zhao, 2002), DG is a collection of method dependence graph representing a *main()* method or a method in a class of the program, or some additional arcs to represent direct or indirect dependencies between a call and the called method and transitive interprocedural data dependencies. DG in AO used to represent the dependencies between the concept of join points, advice, aspects and their associated constructs. CFG is a model of node (or point) corresponds to a program statements, and each arc (or directed edge) indicates the flow of control from one statement to another (EL-Manzalawy, 2004).

1.2 Problem Statement

Many people currently, would search through the code and communicate with a logging method in appropriate location. If the system properly designed and developed, the code might require very minimum changes in only a few places. For most system, though, insertions would be necessary in many places. If the system were object-oriented, we might build a logging class and then use an instance of it to handle the logging. We might need to understand the hierarchy of classes to handle a complex interaction in different files and databases. In AOP, conceptually crosscutting concerns produce a list of codes and classes using standard language constructs expressly because these concerns was tangled with the basic functionality of the code. The repeated rounds of program transformation and analysis are difficult to handle and gives rise to the increase of the system complexity in the maintenance perspectives.

There are many researches focus in getting the best way to represent the architecture of aspect code in order to understand the program. Some of them are using control flow graph (Gold, 2010; Bernardi & di Lucca, 2007; Yin, Jiang, Yin, Zhou, & Li, 2009; Cacho, Filho, Garcia, & Figueiredo, 2008), dependence graph (Arora, Bhatia, & Singh, 2012; Würthinger, 2007), call graph (Ishio et al., 2004; Lin, Zhang, & Zhao, 2009) etc. as their representation tool. The control flow graph allows us to formulate a simple algorithm, based on abstract interpretation, which finds possible-paths constants without the need for program transformations. However, the complexity of the program is related to the algorithm that uses control flow graphs (Pingali, Beck, Johnson, Stodghill, & Moudgill, 1991) and dependence graph (Arora et al., 2012).

There is a different problem when using dependence graph. The dependence graphs can only construct the relationship of data dependency once at the beginning of the code before we identified the slicing area. It can only represent single process and cannot handle multiple programs at once. The complexity of algorithms that use the various dependence graphs is very high, and none of them finds possible paths (Duanzhi, 2010). Therefore, the slicing based on dependence graph must be study in deep and detail.

Control flow graphs show the block of sequence consecutive statement flow of control from the beginning and leaves at the end without halt or possibility of branching until the end of the program. It can present the flow of processes even in complicated

branches. Nevertheless, the more branches in the program the harder to eliminate dead code in low-level code. It makes hard to extract information branches and loops in the program because different branches taken, different number of loop iterations in the execution.

An ideal program representation for dependence flow graph would have a local execution semantics from which an abstract interpretation can easily derived. It would also be a sparse representation of program dependencies, in order to yield an efficient algorithm. There are some advantages and disadvantages of dependence graph and control flow graph. Why not we mix the advantages of dependence graph and control flow graph and prove a better solution to represent the architecture of aspect code. We view as a data structure that can be traversed efficiently for dependence information and it can be viewed as a precisely defined language with a local operational semantics and prove a better solution to represent the architecture of aspect code. We view as a data structure that can be traverse efficiently for dependence information, and it can be viewed as a precisely defined language with a local operational semantics.

Moreover, for more effective software maintenance, program slicing has been advocated to clearly extract all statements that may possibly be affected in the aspect-oriented program. Ishio, Kusumoto, and Inoue (2002) have proposed a step to slice the aspect-oriented programs. However, Ishio's work focuses on dependence graph which cannot be used directly. In this thesis, the idea of control flow graph and dependence graph are merged to form a single graph for the purpose of slicing of aspect-oriented programs.

1.3 Research Objectives

The main objective of this research is to propose a static slicing approach on aspect-oriented programs for maintenance purposes. In order to achieve this objective, the research needs to fulfil the following sub-objectives:

- To propose an aspect-oriented dependence flow graph to represent aspect-oriented programs.
- To formulate static slicing criteria for aspect-oriented programs by defining criteria involving features of aspect-oriented programs.
- To develop a prototype tool that implements the aspect-oriented dependence flow graph with the defined slicing criteria.

1.4 Scope of the Research

This research is about slicing of aspect-oriented programs to help maintainers in maintaining software that was developed. Aspect-oriented programs will be represented by a proposed graph that is able to show the aspect-oriented features. Control flow graph and dependence graph are the basis for formation of the proposed graph for the purpose of slicing aspect-oriented programs. The slicing criteria for the proposed graph extends the slicing criteria for object-oriented. This research uses static slicing as the technique to slice the proposed graph. This idea is implemented in a prototype tool. Furthermore,

this work focuses on AspectJ language which is seamless aspect-oriented extension of JAVA programming language.

The consistencty of output from the tool been investigated through construction of graphs for ten AspectJ programs of various sizes ranging from 87 to 2285 lines of code with the number of aspects is between 1 to 9. The ten programs were taken from AspectJ Development Tools. These programs are commonly used by other researchers in aspect-oriented research. Moreover, the evaluation of the study is conducted by running an experiment involving 20 experts.

1.5 Thesis Organization

This thesis is divided into eight chapters. The first chapter is the introduction of the thesis. It describes the background study, problem statement, research objectives and scope of study. Literature review is in Chapter two. The chapter does the discussion about overview of software maintenance, reverse engineering and its comprehension tools. After that is a brief of aspect-oriented paradigm and slicing as one of the reverse engineering technique in software engineering.

Chapter three is a discussion about research methodology. This chapter is about the way how this research was completely conducted. It begins by understanding the flow of idea in literature study. Next the chapter briefly explains a proposal on static slicing technique for maintaining aspect-oriented program. After that, the application of proposed technique to the tool that developed as a prototype. At the end, experiment was conducted in order to evaluate the proposed technique.

Chapter four is a discussion about the concept of aspect-oriented dependence flow graph and its original study. The scope of discussion in this chapter is about to bring the original study of DFG into the implementation of AODFG as a specifically focus to the aspect-oriented. The discussion begins with the Du and Ud chain and DFG concept. Based on the concept, the proposed aspect-oriented dependence flow graph (AODFG) is conceptually designed. The concept of AODFG is based on the control flow and dependencies in aspect-oriented.

Chapter five then continue from discussion in Chapter 4 and specifically discussed about the proposal of slicing approach for aspect-oriented dependence flow graph. The discussion begins with introducing slicing approach in original study and followed by the work on the program slicing in the perspective of aspect-oriented.

Chapter six is a discussion about the implementation of slicing AODFG by using AOST which is a prototype to support AODFG. This chapter begins with detail out the requirements needed by AODFG. The clearly defined requirements help to design the prototype. It then developed by following the design.

Chapter 7 is a description about the experiment and survey that we were contribute to support the proposed concept. This chapter described about the way how the experiment and survey are implemented. There are two evaluations provided. The first evaluation is an empirical evaluation by introducing the technique to the expertise in the area of study. The second evaluation is to know the capability of AOST to used and compare with the traditional techniques. The details of discussion can be referred to the particular given. Chapter eight is the conclusion of thesis and some suggestion for future works.

REFERENCES

Abran A., Nguyenkim H. (1993). Measurement of the Maintenance Process from a Demand-Based Perspective. *Journal of Software: Evolution and Process*, 5(2): 63- 90.

Agrawal, H., & Horgan, J. (1990). Dynamic Program Slicing, *ACM SIGPLAN Notices*, 25(6): 246-256.

Al-Fawareh, H. J. K. (2001). *Slicing Object-oriented Programs for Maintenance Purpose*. Ph.D Thesis, Universiti Putra Malaysia.

April, A. (2010). Studying supply and demand of software maintenance and evolution services. *Seventh International Conference on the Quality of Information and Communications Technology*, 352-357.

Arora, V., Bhatia, R. K., & Singh, M. (2012). Evaluation of flow graph and dependence graphs for program representation. *International Journal of Computer Applications*, 56(14):18–23.

Baniassad, E. & Clarke, S. (2005). *Aspect-Oriented Analysis and Design: The Theme Approach*. Reading, Mass.: Addison-Wesley.

Basili, V., Shull, F., & Lanubile, F. (1999). Building Knowledge Through Families of Experiments, *IEEE Transactions on Software Engineering*, 25: 435-437.

Batista, T., Chavez, C., Garcia, A., Rashid, A., Sant'Anna, C., Kulesza, U., & Filho, F. C. (2006). Reflections on architectural connection: seven issues on aspects and ADLs. *Proceedings of the 2006 International Workshop on Early Aspects at ICSE06*, 3-10.

Bernardi, M. L., & di Lucca, G. A. (2007). An interprocedural aspect control flow graph to support the maintenance of aspect oriented systems. *IEEE International Conference on Software Maintenance (ICSM2007)*, 2-5 Oct, 435–444.

Beszedes, A., Gergely, T., Mihaly Szabo, Z., Csirik, J. & Gyimothy, T. (2001). Dynamic Slicing Method for Maintenance of Large C Programs, *Proceedings of the Fifth European Conference on Software Maintenance and Reengineering (CSMR'01)*, 105-113 .

Braak, T. (2006). Extending Program Slicing in Aspect-Oriented Programming with Inter-Type Declarations, *5th Twente Student Conference on IT*, Enschede.

Cacho, N., Filho, F. C., Garcia, A., & Figueiredo, E., (2008). EJFlow : Taming exceptional control flows in aspect-oriented programming. *Proceedings of the 7th international conference on Aspect-oriented software development (AOSD'08)*, 72–83.

Canfora, G., & Cimitile, A. (2002). Software Maintenance. In S. K. Chang (Eds.), *Handbook of Software Engineering and Knowledge Engineering Vol. 1* (pp. 91-120). Singapore:World Scientific.

Capilla, R., Duenas, J. C., & Ferenc, R. (2013). A retrospective view of software maintenance and reengineering research – a selection of papers from European Conference on Software Maintenance and Reengineering 2010. *Journal of Software: Evolution and Process*. 25(6):569-661.

Chapin, N. (2009). Software maintenance in complying with IT governance: A report from the field. *IEEE International Conference on Software Maintenance (ICSM2009)*, 499-502.

Chen, Z., Duan, Y., Zhao, Z., Xu, B., & Qian, J. (2011). Using Program Slicing To Improve the Efficiency and Effectiveness of Cluster Test Selection, *International Journal of Software Engineering and Knowledge Engineering*, 21(06): 759–777.

Chikofsky, E. J., & Cross, J. H. (1990). Reverse Engineering and Design Recovery: A Taxonomy. *IEEE Software*, 7(1): 13-17.

Colyer, A., & Clement, A. (2005). Aspect-oriented Programming with AspectJ. *IBM System Journal*, 44(2): 301-308.

Coogan, K., Debray, S., Kaochar, T., & Townsend, G., Automatic static unpacking of malware binaries. *Proceedings 16th Working Conference on Reverse Engineering (WCRE2009)*, 167-176.

Desharnais, J., & April, A. (2010). Software maintenance productivity and maturity. *Proceedings of the 11th International Conference on Product Focused Software (PROFES '10)*, 121-125.

Duanzhi, C. (2010). A collection of program slicing. *International Conference on Computer Application and System Modeling (ICCASM 2010)*, 82–85.

EL-Manzalawy, Y. (2004). *Aspect Oriented Programming*. Retrieved from http://www.developer.com/print.php/3308941.

Erdil, K., Finn, E., Keating, K., Meattle, J., Park, S., & Yoon, D. (2003). *Software Maintenance As Part of the Software Life Cycle, Comp180: Software Engineering Project*. Department of Computer Science, Tufts University.

Gallagher, K. B., & Lyle, J. R. (1991). Using Program Slicing in Software Maintenance. *IEEE Transactions on Software Engineering*, 17(8): 751–761.

Gold, R. (2010). Control flow graphs and code coverage. *International Journal of Applied Mathematics and Computer Science*, 20(4):739–749.

Gold, R., (2014), Reductions of Control Flow Graphs. *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, *8*(3), 427–434. Retrieved from http://waset.org/publications/9997636/reductions-of-control-flow-graphs

Gradecki, J., & Lesiecki, N. (2003). *Mastering AspectJ: Aspect-Oriented Programming in Java*. Indianapolis: Wiley Publishing, Inc.

Grubb, P.A. & Takang, A.A. (2003). *Software Maintenance Concepts and Practice*. London: Thompson Computer Press.

Grundy, J. (1999). Aspect-oriented Requirements Engineering for Component-based Software Systems. *Proceedings IEEE International Symposium on Requirements Engineering*, 84–91.

Hunt, B., Turner, B., & McRitchie, K. (2008). Software Maintenance Implications on Cost and Schedule. *IEEE Aerospace Conference*, 1-6.

IEEE Std 1219-1998. (1998). *IEEE Standard for Software Maintenance*. Software Engineering Standard, IEEE Computer Society.

Ishio, T., Kusumoto, S., & Inoue, K. (2002). Program slicing tool for effective software evolution using aspect-oriented technique. *Sixth International Workshop on Principles of Software Evolution (IWPSE'02)*, 1–10.

Ishio, T., Kusumoto, S., & Inoue, K. (2004). Debugging support for aspect-oriented program based on program slicing and call graph. *Proceedings of 20th IEEE International Conference on Software Maintenance*, 178–187.

ISO/IEC Standard 14764. (2000). *Software Engineering – Software Life Cycle Processes – Maintenance*. International Organisation for Standardization: Geneva, Switzerland.

Jaffar, J., & Murali, V. (2014). A Path-Sensitively Sliced Control Flow Graph. *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE2014)*, 133–143.

Jia, L., Jing, Y., Hui, W. M., & Hone, J. C. (2008). Crosscutting Invariant and An efficient Checking Algorithm Using Program Slicing. *ACM SIGPLAN Notices*, 43(2): 12–20.

Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., and Griswold, W. G. (2001). An Overview of AspectJ. *Proceedings of the 15th European Conference on Object-Oriented Programming*, Springer-Verlag, 327–355.

Kiselev, I. (2003). *Aspect-oriented programming with AspectJ*. Indianapolis: SAMS Publishing.

M. Hassaan, D. Nguyen, and K. Pingali, (2015) "Kinetic dependence graphs," in *ASPLOS'15*,

Kulesza, U., Sant'Anna, C., Garcia, A., Coelho, R., Staa, A., & Lucena, C. (2006). Quantifying the Effects of Aspect-Oriented Programming: A Maintenance Study. *22nd IEEE International Conference on Software Maintenance*, 223–233.

Laddad, R. (2010). *AspectJ In Action*. 2nd Edition, Greenwich: Manning Publications Co.

Lallchandani, J. T., & Mall, R. (2009). Static Slicing of UML Architectural Models. *Journal of Object Technology*, 8(1): 159-188.

Larsen, L., & Harrold, M. J. (1996). Slicing Object-Oriented Software. *Proceedings of the 18th International Conference on Software Engineering*, 495-505.

Lehman, M. M. (1980). Programs, Lifecycles, and the Laws of Software Evolution. *Proceedings of the IEEE*. 68(9):1060-1076.

Lehman, M. M. (1996). Laws of Software Evolution Revisited. *Proceedings of the 5th European Workshop on Software Process Technology*, UK,108-124.

Li, B. (2002). Analyzing information-flow in java program based on slicing technique. *ACM SIGSOFT Software Engineering Notes*, 27(5): 98-103.

Lientz, B, & Swanson, E. (1980). *Software maintenance management*. Reading, Mass.: Addison-Wesley,(p.214).

Lin, Y., Zhang, S., & Zhao, J. (2009). Incremental call graph reanalysis for AspectJ software. *IEEE International Conference on Software Maintenance (ICSM2009)*, 306–315.

Mamone, S. (1988). The IEEE standard for software maintenance. *ACM SIGSOFT Software Engineering Notes.* 19:75-76.

Mohanty, S. R., Behera, P. K., & Mohapatra, D. P., (2015). Slicing Aspect-oriented program Hierarchically. *(IJCSIT) International Journal of Computer Science and Information Technologies*, *6*(6), 5004–5013.

Mohapatra, D. P., Kumar, R., Mall, R., Kumar, D. S., & Bhasin, M. (2006). Distributed dynamic slicing of Java programs. *Journal of Systems and Software*, 79(12): 1661–1678.

Mohapatra, D. P., Sahu, M., Kumar, R., & Mall, R. (2008). Dynamic Slicing of Aspect-Oriented Programs. *Informatica*, 32:261–274.

Moreira, A., Chitchyan, R., Araujo, J., & Rashid, A. (eds.) (2013). *Aspect-oriented Requirements Engineering*. New York: Springer.

Nanda, M.G., & Ramesh, S. (2006). Interprocedural Slicing of Multithreaded Programs with Applications to Java. *ACM Transactions on Programming Languages and System*, 28(6): 1008-1144.

Nishimatsu, A., Kusumoto, S., & Inoue, K. (1999). Experimental Evaluation of Program Slicing on Software Maintenance Process, *IEICE Transactions on Information and Systems*, Pt.1 (Japanese Edition), J82-D-1(8): 1121-1123.

Ohata, F., Hirose, K., Fujii, M., & Inoue, K. (2001) A Slicing Method for Object-Oriented Programs Using Lightweight Dynamic Information, *Proceedings of Eighth Asia-Pacific Software Engineering Conference*, 273-280.

Parizi, R. M. (2008). *Control Flow Structure and Graph Embodiment of Aspect-Oriented Programs (AOPs): Definitions, Algorithm, and tool Support*. Project for Master of Computer Science, Universiti Putra Malaysia.

Pigoski T. M., (1996). *Practical software maintenance: Best practices for managing your software investment*. NewYork: John Wiley & Son.(p. 384).

Pingali, K., Beck, M., Johnson, R., Stodghill, P., & Moudgill, M. (1991). Dependence flow graphs: an algebraic approach to program dependencies. *Proceedings of the 18th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '91)*, 67–78.

Raheman, S. R., Ray, A., & Pradhan, S. (2011). Dynamic Slicing of Aspect-Oriented Programs using AODG, *International Journal of Computer Science and Information Security*, 9(4): 123–126.

Rashid, A. (2004). *Aspect-oriented Database Systems*. Berlin: Springer-Verlag.

Ray, A., Mishra, S., & Mohapatra, D. P. (2013). An Approach for Computing Dynamic Slice of Concurrent Aspect-Oriented Programs. *International Journal of Software Engineering and Its Applications*, 7(1): 13–32.

Ren, Y., Chen, X., Xing, T., & Chai, X. (2011). Research on Software Maintenance Cost of Influence Factor Analysis and Estimation Method. *Proceedings of 3rd International Workshop on Intelligent Systems and Applications (ISA)*, May 28-29, Wuhan, 1–4.

Rombach, D. H., & Ulery, B. T. (1989). Improving Software Maintenance through Measrement. *Proceedings of the IEEE*, 77(4): 581-595.

Rumbaugh, J., Jacobson, I., & Booch, G. (2004). *The Unified Modeling Language Reference Manual*, Canada: Addison-Wesley.

Sahu, M., & Mohapatra, D. P. (2007). A Node-Marking Technique for Dynamic Slicing of Aspect-Oriented Programs. *10th International Conference on Information Technology (ICIT 2007)*, 155–160.

Sikka, P., & Kaur, K. (2013). Program Slicing Techniques and their Need in Aspect Oriented Programming. *International Journal of Computer Applications*, 70(3): 11–15.

Singh, R. (1996). International Standard ISO/IEC 12207 Software Life Cycle Processes. *Software Process: Improvement and Practice*, 2(1), 35-50.

Singh, Y., & Goel, B. (2007). A Step Towards Software Preventive Maintenance. *ACM Software Engineering Notes*, 32(4), article no.10.

Vanek, L., & Davis, L. (1990). Expert Dataflow and Static Analysis Tool. *IEEE Software*, 7(3): 63.

Weiser, M. (1979). *Program Slices: Formal, Psychological, and Practical Investigations of an Automatic Program Abstraction Method*. Ph.D Thesis, University of Michigan.

Weiser, M. (1982). Programmers Use Slices With Debugging. *Communications of the ACM*, 25(7): 446- 452.

Würthinger, T. (2007). *Visualization of program dependence graphs*. Master's Thesis, Johannes Kepler University Linz.

Xu, B., Qian, J., Zhang, X., Wu, Z., & Chen, L. (2005). A Brief Survey of Program Slicing, *ACM Sigsoft Software Engineering Notes*, 30(2):1-36.

Xu, G., & Rountev, A. (2007). Data-flow and Control-flow Analysis of AspectJ Software for Program Slicing. *Technical Report OSU-CISRC-5/07-TR46*, Computer Science and Engineering Research Center, Department of Computer Science and Engineering, Ohio State University.

Yatapanage, N., Winter, K., & Zafar, S. (2010). Slicing Behavior Tree Models for Verification. In C. S. Calude & V. Sassone (eds.), *Theoretical Computer Science IFIP AICT*, Vol. 323, (p.125–139). Springer.

Yin, W., Jiang, L., Yin, Q., Zhou, L., & Li, J. (2009). A control flow graph reconstruction method from binaries based on XML. *International Forum on Computer Science -Technology and Applications*, 226–229.

Zhang, G., & Mei, R. (2008). An Approach of Concurrent Object-oriented Program Slicing Base on LTL Property. *International Conference on Computer Science and Software Engineering 2008*, 650-653.

Zhang, S., Gu, Z., Lin, Y., & Zhao, J. (2008). Celadon : A Change Impact Analysis Tool for Aspect-Oriented Programs, *Proceedings ICSE Companion '08 Companion of 30th International Conference on Software Engineering*, 913–914.

Zhang, X., Gupta, R., & Zhang, Y., Precise Dynamic Slicing Algorithms, 2003, IEEE, 319–329.

Zhao, J. (2002). Slicing aspect-oriented software. *Proceedings of 10th International Workshop on Program Comprehension*, 251–260.

Zhi, Y. (2004). *Software Engineering Standards Manual*. China Standards Press.