A MODEL OF SOFTWARE COMPONENT INTERACTIONS USING THE CALL GRAPH TECHNIQUE

NOORAINI ISMAIL

MASTER OF SCIENCE (INFORMATION TECHNOLOGY)

UNIVERSITI UTARA MALAYSIA

2013

A MODEL OF SOFTWARE COMPONENT INTERACTIONS USING THE CALL GRAPH TECHNIQUE

A thesis submitted to the UUM College of Arts and Sciences in fulfilment of the requirements for the degree of Master of Science
Universiti Utara Malaysia

by
Nooraini Ismail

Permission to Use

In presenting this thesis in fulfilment of the requirements for a postgraduate degree from Universiti Utara Malaysia, I agree that the Universiti Library may make it freely available for inspection. I further agree that permission for the copying of this thesis in any manner, in whole or in part, for scholarly purpose may be granted by my supervisor(s) or, in their absence, by the Dean of Awang Had Salleh Graduate School of Arts and Sciences. It is understood that any copying, publication, or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to Universiti Utara, Malaysia for any scholarly use that may be made of any material from my thesis.

Requests for permission to copy or to make other use of materials in this thesis, in whole or in part, should be addressed to:

Dean of Awang Had Salleh Graduate School of Arts and Sciences
UUM College of Arts and Sciences
Universiti Utara, Malaysia
06010 UUM Sintok

Abstrak

Maklumat interaksi yang berkaitan dengan operasi antara komponen adalah penting, terutama apabila program perlu diubahsuai dan diselenggara. Oleh itu, komponen yang terlibat perlu dikenal pasti dan dipadankan berdasarkan keperluan sistem. Maklumat berkenaan boleh diperolehi menggunakan teknik kod ulasan. Walau bagaimanapun, proses ini mengambil masa yang panjang. Penyelidikan ini mencadangkan suatu model untuk mewakili maklumat tersebut yang mana ia diperolehi secara automatik daripada kod sumber untuk menyediakan paparan yang berkesan untuk perwakilan interaksi antara komponen perisian. Untuk mencapai objektif kajian, metodologi reka bentuk kajian yang mengandungi lima fasa telah diadaptasikan iaitu kesedaran kepada masalah, cadangan, pembangunan, penilaian dan kesimpulan. Fasa pembangunan mempunyai proses yang lebih terperinci yang mana maklumat interaksi antara komponen perlu diperolehi secara automatik menggunakan peralatan kejuruteraan balikan dan program tambahan. Program ini digunakan untuk mendapatkan maklumat perisian, maklumat interaksi komponen dalam program perisian, dan untuk mewakili model dalam bentuk *call graph*. Graf yang dihasilkan ini dinilai melalui dua cara, iaitu menggunakan alatan penggambaran yang bersesuaian dan juga melalui kajian oleh pakar. Alatan penggambaran digunakan untuk memaparkan graf yang dihasilkan daripada format teks ke paparan grafik. Proses penilaian model pula dijalankan melalui teknik kajian pakar. Hasil kajian ini menunjukkan bahawa model yang terhasil boleh digunakan dan dimanipulasikan untuk tujuan menggambarkan maklumat interaksi antara komponen. Model ini boleh digunakan untuk menyediakan paparan penggambaran bagi penganalisis untuk melihat interaksi maklumat yang relevan dalam komponen perisian. Ia juga dalam meningkatkan pemahaman mengenai integrasi komponen itu sendiri, supaya penganalisis boleh memanipulasi dan mengekalkan perisian untuk tujuan tertentu.

Kata kunci: Model interaksi komponen, Perwakilan *call graph*, Kefahaman program, Alatan penggambaran

Abstract

Interaction information that is related to operations between components is important, especially when the program needs to be modified and maintained. Therefore, the affected components must be identified and matched based on the requirement of the system. This information can be obtained through performing the code review technique, which requires an analyst to search for specific information from the source code, which is a very time consuming process. This research proposed a model for representing software component interactions where this information was automatically extracted from the source code in order to provide an effective display for the software components interaction representation. The objective was achieved through applying a research design methodology, which consists of five phases: awareness of the problem, suggestion, development, evaluation, and conclusion. The development phase was conducted by automatically extracting the components' interaction information using appropriate reverse engineering tools and supporting programs that were developed in this research. These tools were used to extract software information, extract the information of component interactions in software programs, and transform this information into the proposed model, which was in the form of a call graph. The produced model was evaluated using a visualization tool and by expert review. The visualization tool was used to display the call graph from a text format into a graphical view. The processed model evaluation was conducted through an expert review technique. The findings from the model evaluation show that the produced model can be used and manipulated to visualize the component interactions. It provides a process that allows a visualization display for analysts to view the interaction of software components in order to comprehend the components integrations that are involved. This information can be manipulated and improved the program comprehension, especially for other software maintenance purposes.

Keywords: Component interaction model, Call graph representation, Program comprehension, Visualization tool

Acknowledgement

Alhamdulillah. A thousand gratitude to Allah S.W.T for giving me strengths to finish my studies.

First of all, I would like to express my deepest gratitude to my supervisor, Dr Nor Laily Hashim for her excellent guidance, moral support and advice throughout my studies.

For my thesis examiners, who give me their valuable comments and feedbacks.

I would like to acknowledge my sponsor, Ministry of High Education for granting sponsorship of my research study.

My dearest thanks I own to my husband Mohd Hafiz Abdullah, for his encouragement, patient, prayers and support throughout my studies.

I also thanks to my family, my masters colleagues and friends for their help and encouraged me through this thesis. May Allah bless us, InsyaAllah.

Table of Contents

List of Tables

List of Figures

CHAPTER ONE
INTRODUCTION

1.1 Overview

This chapter includes an overall research plan of this study by introducing the research foundation and motivation to be undertaken in this research. It also includes a detailed description of the issues to be studied, the research objectives, scope of the research, the research framework, and the contribution expected to be gained in this research.

1.2 Introduction

A software component can be a single element of software that can be integrated with other components (Szyperski, 1998). Two components are integrated if they can potentially react to the same events (Fiege, 2005), which is bypassing messages through their interfaces when the components are provided or required for specific events (Inverardi et al., 2003). The communication between components typically is realized by procedure calls or any kind of messaging (Bure et al., 2009).

When new components are integrated, a newly added component has an effect on another component and it can be used by other components. Because of this situation, the program may crash or immediately stop the execution of the system. For this reason, a programmer must scan through the program and investigate which components are causing the errors.

The contents of the thesis is for internal user only

REFERENCES

Abran, A., Moore, J., Bourque, P., Dupuis, R., & Tripp, L. (2004). Guide to the software engineering body of knowledge: 2004 version: *IEEE Computer Society*, Los Alamitos, CA. Retrieved 23 July 2010, from http://www.swebok.org

Acharya, R. (2013). *Object-oriented design pattern extraction from Java source code*. Master's thesis. UPPSALA University.

Ackermann, C. & Lindvall, M. (2007). *Understanding change requests to predict software impact.* Paper presented at the Software Engineering Workshop, 2006. SEW'06.

Aldrich, J., Chambers, C., & Notkin D. (2002). Architectural reasoning in ArchJava. In Proceedings ECOOP 2002, volume 2374 of LNCS, pp 334-367. Berlin, DE: Springer Verlag.

Ali, Q. (2008). *Static program visualization within the ASF+ SDF meta-environment.* Master's thesis. University of Amsterdam, Holland, Netherlands.

Arafa, Y., Boldyreff, C., Tawil, A. H., & Liu, H. (2012). *A high level service-based approach to software component integration.* Paper presented at Sixth International Conference on the Complex, Intelligent and Software Intensive Systems.

Arboleda, H., Royer, J. C (2011). *A comparison of two Java component extraction approaches*. Paper presented at Proceedings of the 4th India Software Engineering Conference (ISEC '11), New York, NY, USA. pp. 155-164.

Ardakan, M. A. & Mohajeri, K. (2009). Applying design research method to IT performance management: Forming a new solution. *Journal of Applied Sciences, 9*(7), 1227-1237.

Arrington, C. T. (2001). *Enterprise Java with UML*. New York, NY: John Wiley & Sons.

Ayewah, N., Hovemeyer, D., Morgenthaler, J., Penix, J., & Pugh, W. (2008). Using static analysis to find bugs. *Software, IEEE, 25*(5), 22-29.

Baecker, R. (1988). *Enhancing program readability and comprehensibility with tools for program visualization*. Proceedings of the 10th international conference on Software engineering. pp. 356-366.

Becker, B. (2011). Modeling and verification of self-adaptive service-oriented systems. *Proceedings of the 5th Ph. D. Retreat of the HPI Research School on Service-oriented Systems Engineering*, *5*, 149.

Bergeron, J., Debbabi, M., Desharnais, J., Erhioui, M., Lavoie, Y., & Tawbi, N. (2001). Static detection of malicious code in executable programs. *International Journal of Requirements Engineering*, 184-189.

Bollobas, B. (1998). *Modern graph theory*. New York, NY: Springer-Verlag.

Briand, L., Labiche, Y., & Sówka, M. (2006). *Automated, contract-based user testing of commercial-off-the-shelf components.* Paper presented at the Proceedings of the 28th international conference on Software engineering.

Brooks, R. (1983). Towards a theory of the comprehension of computer programs. *International Journal of Man-Machine Studies*. 18, 543-554.

Broy, M. & Kruger, I. (1998). *Interaction interfaces-towards a scientific foundation of a methodological usage of message sequence charts.* Proceedings of the Second International Conference on Formal Engineering Methods.

Brusilovsky, P., Grady, J., Spring, M. & Lee, C. (2006) What should be visualized? Faculty perception of priority topics for program visualization. In: *ACMSIGCSE Bulletin, 38*(2), 44-48.

Bures, T., Hnetynka, P., & Malohlava, M. (2009). *Using a product line for creating component systems*. Paper presented at Proceeding of the 2009 ACM symposium on Applied Computing. pp. 501-508.

Caserta, P. & Zendra, O. (2011). Visualization of the static aspects of software: A survey. *IEEE Transactions Journal on Visualization and Computer Graphics, 17*(7), 913-933.

Chatzigeorgiou, A., Tsantalis, N., & Stephanides, G. (2006). *Application of graph theory to OO software engineering*. Paper presented at the Proceedings of the 2006 international workshop on interdisciplinary software engineering research, Shanghai, China.

Chen, H., Dean, D., & Wagner, D. (2004). *Model checking one million lines of C code*. Paper presented at the Proceedings of the 11th Annual Network and Distributed System Security Symposium (NDSS).

Chen, K. & Rajlich, V. (2000). Case study of feature location using dependency graph. Paper presented at the Proceedings of Intern. Workshop on Program Comprehension (IWPC'00), pp. 241-249.

Chen, K. & Rajlich, V. (2001). *RIPPLES: Tool for change in legacy software*. Paper presented at the Proceedings of IEEE International Conference on Software Maintenance.

Chikofsky, E. J. & Cross II, J. H. (1990). Reverse engineering and design recovery: A taxonomy in IEEE software. *IEEE Computer Society*, 13-17.Chouambe, L., Klatt, B., & Krogmann, K., (2008). Reverse engineering software-models of component-based systems. European Conference on Software Maintenance and Reengineering, pp. 93-102. IEEE.

Claub, M. (2001). *Generic modeling using UML extensions for variability*. Paper presented at the Workshop on Domain Specific Visual Languages at Object-Oriented Programming, Systems, Languages, & Applications.

Corritore, C. L. & Wiedenbeck, S. (2001) An exploratory study of program comprehension strategies of procedural and object-oriented programmers. *International Journal of Human-Computer Studies, 54*(1). pp. 1-23.

Crnkovic, I. (2003). *Component-based software engineering-new challenges in software development*. Paper presented at Proceedings of the 25th International Conference on Information Technology Interfaces (ITI).

Cross, J. H., Hendrix, T. D., & Maghsoodloo, S. (1998). The control structure diagram: An overview and initial evaluation. *Empirical Software Engineering, 3*(2), 131-158.

Deo, N. (2004). *Graph theory with applications to engineering and computer science*. New Delhi, India: PHI Learning Pvt. Ltd.

Détienne, F. & Bott, F. (2001) Software design—Cognitive aspects. New York, NY: Springer-Verlag.

Devadas, S., & Lehman, E. (2005). *Mathematics for computer science*. Retrieved 12 May 2010, from http://ocw.mit.edu/NR/rdonlyres/Electrical-Engineering-and-Computer-Science/6-042JSpring-2005/FF95BA8F-6457-4592-B473-349A5C1CA277/0/l8_graphs1.pdf

Diehl, S. (2007). *Software visualization: Visualizing the structure, behaviour, and evolution of software, Secaucus*, NJ. New York, NY: Springer-Verlag.

Eick, S., Graves, T., Karr, A., Mockus, A., & Schuster, P. (2002). Visualizing software changes. *IEEE Transactions on Software Engineering, 28*(4), 396-412.

El-Ramly, M. (2006). *Experience in teaching a software reengineering course.* Paper presented at the Proceedings of the 28th International Conference on Software Engineering, New York.

Emanuelsson, P., & Nilsson, U. (2008). A comparative study of industrial static analysis tools. *Electronic Notes in Theoretical Computer Science, 217*, 5-21.

Evalguide Online.org (n.d). Retrieved 18.06.2012, from: http://www.evalguide.ethz.ch/project_evaluation/introduction/goalbased_evaluation_EN

Ferrante, J., Ottenstein, K., & Warren, J. (1987). The program dependence graph and its use in optimization. *ACM Transactions on Programming Languages and Systems (TOPLAS), 9*(3), 319-349.

Fiege, L. (2005). *Visibility in event-based systems.* (PhD. Thesis). Technische Universität Darmstadt, Darmstadt, Germany.

Gansner, E. R. (2012). *Drawing graphs with Graphviz* technical report. Murray Hill, NJ: AT&T Bell Laboratories.

Garousi, V. & Koochakzadeh, N. (2010). An empirical evaluation to study benefits of visual versus textual test coverage information. *Testing—Practice and Research Techniques*, 189-193.

Garrison, R. (2000). Theoretical challenges for distance education in the 21st century: A shift from structural to transactional issues. *The International Review of Research in Open and Distance Learning, 1*(1).

Gelber, N. (2006). *Bridging component models and integration problems*. Växjö University, Sweden. Retrieved 20 February 2010, from http://urn.kb.se/resolve?urn=urn:nbn:se:vxu:diva-677.

Gibbons, A. (1985). *Algorithmic graph theory*. Cambridge, UK: Cambridge University Press.

Grove, D. & Chambers, C. (2001). A framework for call graph construction algorithms. *ACM Transactions on Programming Languages and Systems, 23*(6), 746.

Grove, D., DeFouw, G., Dean, J., & Chambers, C. (1997). Call graph construction in object-oriented languages. *ACM SIGPLAN Notices, 32*(10), 108-124.

Grubb, P. & Takang, A. (2003). *Software maintenance*. Singapore, Singapore: World Scientific Publishing Company, Incorporated.

Gueheneuc, Y. G. (2005). *Ptidej: Promoting patterns with patterns*. Paper presented at European Conference on Object Oriented Programming, workshop on Building a System with Patterns, Glasgow, Scotland.

Hopkins, J. (2000). Component primer. *Communications of the ACM, 43*(10), 27-30.

Inverardi, P. & Tivoli, M. (2003). Software architecture for correct components assembly. *Formal Methods for Software Architectures*, 92-121.

Iwanari, Y., Tasaki, M., Yokoo, M., Iwasaki, A., & Sakurai, Y. (2009). *Introducing communication in Dis-POMDPs with finite state machines.* Paper presented at IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technologies WI-IAT'09.

Jackson, C. & Pascual, R. (2008). Optimal maintenance service contract negotiation with aging equipment. *European Journal of Operational Research, 189*(2), 387-398.

Jin, Y. & Han, J. (2005). Specifying interaction constraints of software components for better understandability and interoperability. *COTS-Based Software Systems Journal*, 54-64.

Kaeli, A. & Calder, B. (1997). *Procedure mapping using static call graph estimation.* Paper presented at the Proceeding of Workshop Interaction between Compiler and Computer Architecture.

Koehler, J. & Vanhatalo, J. (2007). Process anti-patterns: How to avoid the common traps of business process modelling. *IBM WebSphere Developer Technical Journal, 10*(2), 4.

Koivulahti-Ojala, M. & Käkölä, T. (2010). *Framework for evaluating the version management capabilities of a class of UML modeling tools from the viewpoint of multi-site, multi-partner product line organizations.* Paper presented at the Proceedings of 43rd Hawaii International Conference on Systems Sciences (HICSS-43). IEEE.

Kothari, J., Denton, T., Shokoufandeh, A., Mancoridis, S., & Hassan, A. (2006). *Studying the evolution of software systems using change clusters.* Paper presented at the Proceedings of the 14th International Conference on Program Comprehension. IEEE Computer Society.

Kuck, D., Kuhn, R., Padua, D., Leasure, B., & Wolfe, M. (1981). *Dependence graphs and compiler optimizations.* Paper presented at the Proceedings of the 8th ACM SIGPLAN-SIGACT Symposium on Principles of programming languages.

Lanza, M. (2001). *The evolution matrix: Recovering software evolution using software visualization techniques.* Paper presented at the Proceedings of the 4th international workshop on principles of software evolution.

Lavagno, L., Martin, G. E., & Selic, B. (2003). *UML for real: Design of embedded real-time systems*. Alphen aan den Rijn: Netherlands, Kluwer Academic Publishers.

Lee, D. & Yannakakis, M. (1996). Principles and methods of testing finite state machines. *PROCEEDINGS-IEEE, 84*, 1090-1123.

Letovsky, S. (1986a) Cognitive processes in program comprehension. In *Empirical Studies of Programmers*. 58-79. New York, NY: Ablex Publishing Corporation.

Lilius, J. & Paltor, I. (1999). *Formalising UML state machines for model checking.* Paper presented at the Unified Modeling Language: UML'99: Beyond the Standard: Second International Workshop, Fort Collins, CO, October 28-30, 1999: Proceedings.

Lin, Y., Zhang, S., & Zhao, J. (2009). *Incremental call graph reanalysis for AspectJ software*. Paper presented at the International Conference on Software Maintenance. (ICSM 2009). IEEE.

Lindvall, M. & Publica, F. (2003). Impact analysis in software evolution. *Advances in computers, 59,* 130-211.

McMillan, J. H. & Schumacher, S. (1984). *Research in education: A conceptual introduction*. Boston, MA: Little, Brown.

Muller, H. & Klashinsky, K. (1988). *Rigi—A system for programming-in-the-large.* Paper presented at the Proceedings of the 10th International Conference on Software Engineering.

Murata, M., Tozawa, A., Kudo, M., & Hada, S. (2006). XML access control using static analysis. *ACM Transactions on Information and System Security, 9(3),* 324.

Najumudheen, E., Mall, R., & Samanta, D. (2009). A dependence graph-based representation for test coverage analysis of object-oriented programs.

SIGSOFT Software Engineering Notes, 34(2), 1-8. 100-113. Norwood, NJ: Ablex.

O'Brien, M. (2003) Program comprehension—A review & research direction technical report UL-CSIS-03-3, University Of Limerick.

Ore, O. (1967). *Theory of graphs*. Providence, RI: American Mathematical Society.

Pennington, N. (1987). Comprehension strategies in programming. G. M. Olson, S. Sheppard, & E. Soloway, (eds.). Empirical Studies of Programmers: Second Workshop.

Piel, E. & Gonzalez-Sanchez, A. (2009). *Data-flow integration testing adapted to runtime evolution in component-based systems.* Paper presented at the Proceedings of the ESEC/FSE workshop on Software integration and evolution@ runtime.

Price, B. A., Small, I. S., & Baecker, R. M. (1992). *A taxonomy of software visualization*. Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences, vol 2, pp. 597-606.

Prowell, S. (2005). *Using Markov chain usage models to test complex systems*. Paper presented at the Proceeding 38th Annual Hawaii International Conference. System Sciences (HICSS'05), Big Island, Hawaii.

Rader, J. (1997). *Mechanisms for integration and enhancement of software components.* Paper presented at the Proceedings Fifth International Symposium on Assessment of Software Tools and Technologies.

Rajan, H. & Sullivan, K. (2005). *Classpects: Unifying aspect- and object-oriented language design.* Paper presented at the Proceedings of 27th International Conference on Software Engineering (ICSE).

Rajlich, V. (1997). *A model for change propagation based on graph rewriting.* Paper presented at the 13th International Conference on Software Maintenance (ICSM'97), October 1-3 2007, Wayne State University (pp. 84), Bari, ITALY.

Rajlich, V., Damaskinos, N., Linos, P., & Khorshid, W. (2006). VIFOR: A tool for software maintenance. *Software: Practice and Experience, 20*(1), 67-77.

Reekie, H. & Lee, E. (2002). *Lightweight component models for embedded systems.* Electronics Research Laboratory, College of Engineering, University of California.

Royer, J. C. (2010). *Short draft about the Java component extractor*. Academic Society For Competition Law, INRIA Nantes, France: Mines de Nantes.

Ruiz, M., Espana, S., & Gonzalez, A. (2012). *Model-driven organisational reengineering: A framework to support organisational improvement.* Paper presented at the Informatica (CLEI), 2012 XXXVIII Conferencia Latinoamericana En.

Saidi, H. (2008). Logical foundation for static analysis: Application to binary static analysis for security. *ACM SIGAda Ada Letters, 28*(1), 96-102.

Schilling, J. W. (2007). A cost effective methodology for quantitative evaluation of software reliability using static analysis. The University of Toledo.

Schilling, W. & Alam, M. (2008). *A methodology for quantitative evaluation of software reliability using static analysis.* Paper presented at the Proceedings of the 2008 Annual Reliability and Maintainability Symposium. volume 00.

Scholz, B., Zhang, C., & Cifuentes, C. (2008). *User-input dependence analysis via graph reachability*. Mountain View, CA: Sun Microsystems, Inc.

Sherburne, P. & Fitzgerald, C. (2004). You don't know jack about VoIP. *Queue, 2*(6), 30-38.

Shneiderman, B. & Mayer, R. (1979) Syntactic semantic interactions in programmer behavior: A model and experimental results. In *International Journal of Computers & Information Sciences, 8*(3), 219-238.

Soloway, E. & Ehrlich, K. (1984) Empirical studies of programming knowledge. In *IEEE Transactions on Software Engineering*, SE-10, 595-609.

SoMoX SOftware MOdel eXtractor, Retrieved 10.02.2013, from http://www.somox.org

Suman, R. R. & Mall, R. (2009). State model extraction of a software component by observing its behaviour. *SIGSOFT Software Engineering Notes, 34*(1), 1-7.

Szyperski, C. (1998). *Component software: Beyond object-oriented programming*, New York, NY: ACM Press/Addison-Wesley Publishing Co.

The JCE checker. (2013). Retrieved 2 November 2013, from http://www.emn.fr/z-info/jroyer/JCE/index.html.

Travis, D. (2007). Retrieved 24 July 2013. http://www.userfocus.co.uk/articles/expertreviews.html

Vaishnavi, V. K. & Kuechler, W. (2008). *Design science research methods and patterns* (1st Ed.). Boca Raton, FL: Auerbach Publications.

Von Mayrhauser, A. & Vans, A. M. (1995). Program understanding: Models and experiments. *Advances in Computers, 40*(4), 25-46.

Wen, L. & Dromey, R. (2004). *From requirements change to design change: A formal path.* Paper presented at SEFM 2004. Proceedings of the Second International Conference on Software Engineering and Formal Methods.

Winslow, L. E. (1996). Programming pedagogy—A psychological overview. *ACM SIGCSE Bulletin*, *28*(3), 17-22.

Wu, X. & Woodside, M. (2004). Performance modelling from software components. *ACM SIGSOFT Software Engineering Notes, 29*(1), 301.

Xue, J., Hu, C., Wang, K., Ma, R., & Leng, B. (2009). *Constructing a knowledge base for software security detection based on similar call graph.* Paper presented Second International Conference on Computer and Electrical Engineering.

Zhang, L., Marinov, D., Zhang, L., & Khurshid, S (2011). *An empirical study of JUnit test-suite reduction*. Paper presented at 22nd IEEE International Symposium on Software Reliability Engineering.

Zwillinger, D. (2002). *CRC standard mathematical tables and formulae.* Boca Raton, FL: CRC Pr I Llc.