A DOCUMENT-BASED SOFTWARE TRACEABILITY TO SUPPORT CHANGE IMPACT ANALYSIS OF OBJECT-ORIENTED SOFTWARE

SUHAIMI BIN IBRAHIM

UNIVERSITI TEKNOLOGI MALAYSIA

A DOCUMENT-BASED SOFTWARE TRACEABILITY TO SUPPORT CHANGE IMPACT ANALYSIS OF OBJECT-ORIENTED SOFTWARE

SUHAIMI BIN IBRAHIM

A thesis submitted in fulfilment of the
requirements for the award of the degree of
Doctor of Philosophy

Faculty of Computer Science and Information System
Universiti Teknologi Malaysia

MAY 2006

ALHAMDULILLAH ……

For my beloved parents,

my wife, Hjh. Aslamiah bt. Md Nor

and my children, Muhammad Nazri and Noor Aini

who have given me the strength and courage.

ACKNOWLEDGEMENT

I would like to take this opportunity to thank my main supervisor, Dato' Prof. Dr. Norbik Bashah Idris for his encouragement, advice and inspiration throughout this research. Special thanks go to my co-supervisor, Prof. Dr. Aziz Deraman, the faculty Dean of Technology and Information Science, Universiti Kebangsaan Malaysia for his constant support, technical guidance and constructive review of this research work.

I am especially indebted to Prof. Malcolm Munro of Durham University, United Kingdom, a renown expert in software impact analysis for his early support through insightful ideas and constructive comments on the draft model and approach to this research. At the beginning of this journey he guided and encouraged me throughout my stay at Durham, giving me inspiring and fruitful experience in the research area.

A great gratitude also goes to the Universiti Teknologi Malaysia for sponsoring my three year PhD study and to MOSTI for funding the IRPA research project. My thanks also go to all CASE (Centre For Advanced Software Engineering) staff and individuals who have been involved directly or indirectly in the project. Lastly, my appreciation also goes to post-graduate students of CASE, Universiti Teknologi Malaysia, Kuala Lumpur for their participation in the controlled experiment.

ABSTRACT

The need for software modifications especially during the maintenance phase, is inevitable and remains the most costly. A major problem to software maintainers is that seemingly small changes can ripple through the entire system to cause major unintended impacts. As a result, prior to performing the actual change, maintainers need mechanisms in order to understand and estimate how a change will affect the rest of the system. Current approaches to software evolution focus primarily on the limited scope of change impact analysis e.g. code. This research is based on the premise that a more effective solution to manage system evolution can be achieved by considering a traceability approach to pre-determine the potential effects of change. The aim of this research is to establish a software traceability model that can support change impact analysis. It identifies the potential effect to software components in the system that does not lie solely on code but extends to other high level components such as design and requirements. As such, in this research, modification to software is therefore considered as being driven by both high level and low level software components. This research applies a comprehensive static and dynamic analysis to provide better impact infrastructures. The main research contribution in this thesis can be seen in the ability to provide a new software traceability approach that supports both top-down and bottom-up tracing. In further proving the concept, some software prototype tools were developed to automate and support the potential effects. The significant achievement of the model was then demonstrated using a case study on a non-trivial industrial application software, and evaluated via a controlled experiment. The results when compared against existing benchmark proved to be significant and revealed some remarkable achievements in its objective to determine change impacts.

ABSTRAK

Keperluan terhadap pengubahsuaian perisian terutama pada fasa penyenggaraan adalah suatu yang tidak dapat dielakkan dan masih melibatkan kos yang tinggi. Permasalahan utama kepada penyenggara perisian ialah pindaan yang nampaknya agak kecil boleh merebak ke seluruh sistem lalu menyebabkan impak luar jangka yang besar. Lantaran itu, sebelum kerja penyenggaraan dilakukan, penyenggara perlukan beberapa mekanisma untuk memahami dan menganggar bagaimana pindaan akan menjejaskan bahagian lain dalam sistem. Pendekatan semasa terhadap evolusi perisian lebih memfokuskan kepada skop analisa impak pindaan yang terhad, contohnya kod sumber. Kajian ini berasaskan kepada landasan iaitu penyelesaian efektif untuk menangani evolusi sistem boleh dicapai dengan mengambilkira pendekatan jejakan bagi mengenalpasti terlebih dahulu impak pindaan berpotensi. Tujuan kajian ini ialah untuk menghasilkan satu model jejakan perisian yang boleh membantu menganalisa impak pindaan. Ia mengenalpasti kesan berpotensi terhadap komponen perisian dalam sistem yang tidak terletak semata-mata kepada kod tetapi meluas kepada komponen aras tinggi seperti rekabentuk dan keperluan. Oleh itu dalam kajian ini, pengubahsuaian terhadap perisian boleh diambilkira dengan melibatkan kedua-dua komponen perisian aras tinggi dan aras rendah. Penyelidikan ini menggunakan analisa statik dan dinamik yang komprehensif untuk menyediakan infrastruktur impak yang lebih baik. Sumbangan utama kajian dalam tesis ini boleh dilihat dari segi keupayaan menyediakan satu pendekatan perisian jejakan baru yang membantu kedua-dua jejakan atas-bawah dan bawah-atas. Bagi membuktikan lagi pengesahan konsep, beberapa alatan prototaip perisian dibina untuk mengautomasi dan membantu kesan berpotensi. Pencapaiannya yang signifikan kemudian dipersembahkan dengan menggunakan satu kajian kes yang merupakan satu aplikasi perisian berasaskan industri dan dinilai melalui ekperimen terkawal. Keputusan yang diperolehi apabila dibandingkan dengan penanda aras terbukti ianya signifikan dan memperlihatkan beberapa pencapaian yang memberangsangkan dalam matlamatnya untuk mengenalpasti impak pindaan.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS AND SYMBOLS

AIS	- Actual Impact Set
ANOVA	- Analysis of Variance
AST	- Analysis Syntax Tree
CASE	- Computer Aided Software Engineering
CBO	- Coupling Between Object Classes
CCB	- Change Control Board
CI	- Component Identification
CMM	- Capability Maturity Model
CSC	- Computer Software Components
CSCI	- Computer Software Configuration Item
CSU	- Computer Software units
DIF	- Documentation Integration Facility
DIT	- Depth of Inheritance tree
FSM	- Functional Specification Manual
GEOS	- Global Entity Operations System
HTML	- Hypertext Markup Language
IA	- Impact Analysis
LCOM	- Lack of Cohesion Metric
LOC	- Lines of code
MDD	- Model Dependency Descriptor
NOC	- Number of Children
OBA	- Automobile Board Auto Cruise
OMT	- Object Modeling Technique
OOP	- Object Oriented Programming
PCR	- Problem Change Report
PIS	- Primary Impact Set

RFC	- Response for a Class
RUP	- Rational Unified Process
SADT	- Structured Analysis Design Technique
SCM	- Software Configuration Management
SDLC	- Software Development Lifecycle
SDP	- Software Development Plan
SIS	- Secondary Impact Set
SPEM	- Software Process Engineering Meta Model
SPS	- Specification Product System
SRS	- Software Requirement Specification
STD	- Software Test Description
SUM	- Software User Manual
TRAM	- Tool for Requirement and Architectural Management
UML	- Unified Modeling Language
VG	- Values of complexity
WMC	- Weighted Methods per Class
XML	- Extensible Markup Language

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Introduction

This chapter provides an introduction to the research work presented in this thesis. It describes the research overview that motivates the introduction of *a document-based software traceability to support change impact analysis of object-oriented software*. This is followed by a discussion on the research background, problem statements, objectives and importance of the study. Finally, it briefly explains the scope of work and the structure of the thesis.

1.2 Introduction to Software Evolution

It is unanimously accepted that software must be continuously changing in order for the software to remain relevant in use. The need for changing a software system to keep it aligned with the users' need and expectations has long been recognized within the software engineering community. Due to inherent dynamic nature of business application, software evolution is seen as the long term result of software maintenance. In the current decade, the term *software evolution* is often used as a synonym for software maintenance, broadly defined as modification of a software product after delivery (IEEE, 1998a), and both software maintenance and evolution assume changing the code as their basic operation.

The term software evolution lacks a standard definition, but some researchers and practitioners use it as a preferable substitute for maintenance (Bennett and Rajlich, 2000). In short, one could say that the evolution of a software system is the results of observing the changes made to the system components over a long time span. Consequently, the study of software evolution has aimed at analyzing the process of continuous change to discover trends and patterns, such as for example, the hypothesis that software evolution behaves as feedback processes (Lehman and Ramil, 2000).

The maintenance process describes how to organize maintenance activities. Kitchenham et al. (1999) identify a number of domain factors believed to influence the maintenance process, namely i) *maintenance activity type*, ii) *product*, iii) *peopleware* and iv) *process organization*. *Maintenance activity type* covers the corrections, requirements changes and implementation changes. *Product* deals with the size and product composition. *Peopleware* describes the skills and user requests. *Process organization* manages the group and engineering resources that include methods, tools and technology.

Software maintenance is basically triggered by a change request. Change requests are typically raised by the clients or internal development staff for the demand to do software modification. Traditionally, software modification can be classified into maintenance types that include corrective, adaptive, perfective and preventive. Chapin et al. (2001) view software evolution in slightly different perspective. They use the classification based on maintainers' activity of mutually exclusive software evolution clusters ranging from the support interface, documentation, software properties and business rules. Each cluster is characterized by some maintenance types. They conclude that different types of maintenance or evolution may have different impact on software and business processes.

For whatever reason of modification it may be, the real world software systems require continuous changes and enhancements to satisfy new and changed user requirements and expectations, to adapt to new and emerging business models and organizations, to adhere to changing legislation, to cope with technology

innovation and to preserve the system structures from deterioration (Canfora, 2004). Webster et al. (2005) in their study on risk management for software maintenance projects propose some taxonomy of risk factors that cover requirements, design, code, engineering specialities and legacy. Some common risk factors identified are incomplete specifications and limited understanding, high level complexity of the required change, direct or indirect impacts on current system's functionalities, and inadequate test planning and preparation. The fact about software change is that the changes made by user requirements and operations are propagated onto software system (Bohner, 2002). These changes will require a substantial extension to both the database and code. As change is the basic building block of software evolution, software change is considered a key research challenge in software engineering (Bennett and Rajlich, 2000).

A large portion of total lifecycle cost is devoted to introduce new requirements and remove or change the existing software components (Ramesh and Jarke, 2001). This intrinsically requires appropriate software traceability to manage it. However, there are at least three problems observed by the investigation of the current software traceability approaches. First, most of the Computer Aided Software Engineering (CASE) tools and applications more focus on the high level software and yet are directly applicable to software development rather than maintenance. While, the low level software e.g. code, is given less priority and very often left to users to decide. This makes the software change impact analysis extremely difficult to manage at both levels. Secondly, there exists some research works (Jang et al., 2001; Lee et al, 2000; Tonella, 2003) on change impact analysis but the majority confine their solution at the limited space i.e. code, although more evolvable software can be achieved at the meta model level. Finally, no proper visibility being made by the ripple effects of a proposed change across different levels of workproduct. If this can be achieved, a more concrete estimation can be predicted that can support change decision, cost estimation and schedule plan.

The key point to the above solutions is the software traceability. Software traceability provides a platform as to how the relationships within software can be established and how the change impact can be implemented. All these issues require

an extensive study on the existing traceability and impact analysis in the existing software system and research works before a new model and approach can be decided.

1.3 Background of the Problem

Software traceability is fundamental to both software development and maintenance of large systems. It shows the ability to trace information from various resources that requires special skill and mechanism to manage it. In this research problem, it focuses on software traceability to support change impact analysis. Following are some major issues to the research problems.

1.3.1 Broader Maintenance Perspective

Many researchers have been working on the code-based maintenance for their software evolution as mentioned earlier. This type of maintenance is more focused but limited as it deals with a single problem, i.e. source code. However, managing software change at a restricted level is not enough to appreciate the actual impacts in the software system. Other levels of software lifecycle such as requirements, testing and design should also be considered as they are parts of the software system. To observe the impact at the broader perspective is considerably hard as it involves software traceability within and across different workproducts.

Software workproducts refer to explicit products of software as appeared in software documention. For instance, test plans, test cases, test logs and test results are the workproducts of testing document. Architectural design model and detailed design model are the workproducts of design document. Code is by itself a workproduct of source code. Requirements and specifications are the workproducts of software requirements specification document. Thus, at the broader perspective of change impact, it observes the impact within and across different workproducts such

within code and from code to design, specification, etc. Within a workproduct, the software components may be decomposed further into smaller elements, called artifacts. For instance, code can be decomposed into classes, methods, attributes and variables of software artifacts. In this thesis the term components and artifacts are sometimes used interchangeably which refer to the same thing.

The fact about this traceability approach is that if the component relationships are too coarse, they must be decomposed to understand complex relationships. On the other hand, if they are too granular, it is difficult to reconstruct them into more recognized, easily understood software components. However, it is perceived that there should be some tradeoffs between these two granularities in order to observe the ripple effects of change.

1.3.2 Effective Change Communication

There is a need to communicate and share information within software development environment e.g. between software developers and management (Lu and Yali, 2003). In Software Configuration Management (SCM) for example, the Change Control Board (CCB) needs to evaluate change requests before the actual change is implemented. This requires CCB to consult other staff such as maintainers, software engineers and project manager. A maintainer himself needs to examine among other tasks how much and which part of the program modules will be affected for change and regression testing, its complexity and what types of test cases and requirements will be involved.

The software engineers need to examine the types of software components to use and more critically to identify or locate the right affected software components (Bohner, 2002). The software manager is more concerned about the cost, duration and staffing before a change request is accepted. This certainly requires a special repository to handle software components with appropriate links to various levels in software lifecycle. Some of this information is not readily available in any CASE

tools that require the software development staff to manually explore the major analyses in some existing software models and documentations.

1.3.3 Importance of System Documentation

To established organizations with software engineering practices in place, software engineers always refer to system documentation as an important instrumentation for communication (IEEE, 1998). The high level management finds documentation very useful when communicating with the low level developers or vice-versa. Despite this, many software engineers are still reluctant to make full use of the system documentation particularly to deal with software evolution (Nam et al., 2004). To them, documentation is abstract, seldom up-to-date and time consuming. However, many still believe that documentation is more significant if the software it documents is more visible and traceable to other parts of software components. For example, within documentation the user can visualize the impacted requirements and design classes in the database repository.

1.3.4 Incomplete Maintenance Supported CASE Tools

Many CASE (Computer Aided Software Engineering) tools that exist today are aimed at addressing the issues of software development rather than software maintenance (Pressman, 2004). Some claim that their tools can support both software development and maintenance, however their applications are mainly centered around managing, organizing and controlling the overall system components, very few focus on the impact analysis of change requests. Deraman (1998) relates the current CASE tools as applications that do provide special upfront consistency checking of software components during development but tend to ignore its equally important relationships with code as it proceeds towards development and maintenance. The traceability relationships between code and its upper

functionalities are not explicitly defined and very often left to software engineers to decide.

From the above scenarios, there is a need to integrate both the high level and low level software abstracts such that the effects of component traceability can be applied in the system thoroughly. Two main issues need to be addressed here firstly, the software traceability within a software workproduct and secondly, the traceability across many workproducts. Software traceability in this context reflects the underlying infrastructures of ripple effects of change that attempts to incorporate both the techniques and models in implementing a change impact.

1.4 Statement of the Problem

This research is intended to deal with the problems related to requirements traceability for change impact analysis as discussed in Section 1.2. The main question is *"How to produce an effective software traceability model and approach that can integrate the software components at different component levels to support change impact analysis of software maintenance?"*

The sub questions of the main research question are as follows:

i. Why the current maintenance models, approaches and tools are still not able to support potential effects of change impact in the software system?
ii. What is the best way to capture the potential effects of software components in the system?
iii. How to measure the potential effects of a proposed change?
iv. How to validate the usefulness of software traceability for software maintenance?

Sub question (i) will be answered via literature reviews in Chapter 2. This chapter will provide a special attention to explore the software evolution, its models and traceability issues. From the impact analysis perspective, this chapter will

present a study on the detailed traceability process, the techniques and existing tools used. The strengths and drawbacks are drawn based on a comparison framework in order to propose a new model and approach to support change impact analysis.

The above study provides some leverage to answer the sub question (ii). Chapter 3 describes a design methodology and evaluation plan before the research is carried out. Sub question (iii) will be counter balanced by a solution to measure the potential effects. The sub questions (ii) and (iii) will be further explained in the traceability modeling and implementation as described in Chapter 4 and 5. Lastly, sub question (iv) leads to the evaluation of the model and approach quantitatively and qualitatively as described in Chapter 6.

1.5 Objectives of the Study

The above problem statement serves as a premise to establish a set of specific objectives that will constitute major milestones of this research.

To this end, the objectives of this research are listed as follows

1) To build a new software traceability model to support change impact analysis that includes requirements, test cases, design and code.
2) To establish a software traceability approach and mechanism that cover features including artifact dependencies, ripple effects, granularity and metrics.
3) To develop software supporting tools to support the proposed model and approach.
4) To demonstrate and evaluate the practicability of the software traceability model and approach to support change impact analysis.

1.6 Importance of the Study

Software maintenance is recognized as the most expensive phase of the software lifecycle, with typical estimate of more than sixty percent of all effort expended by a development organization, and the percentage continues to rise as more software is produced (Han, 2001). As software changes are introduced, avoiding defects becomes increasingly labor intensive. Due to lack of advanced technologies, methods and tools, doing software modification has been difficult, tedious, time consuming and error prone. Software maintainers need mechanisms to understand and solve maintenance tasks e.g. how a change impact analysis can be made for a software system.

Bohner and Arnold (1996) describes a benefit of change impact analysis as

…by identifying potential impacts before making a change, we can greatly reduce the risks of embarking on a costly change because the cost of unexpected problems generally increases with the lateness of their discovery.

Clearly, the software change impact is an important activity that needs to be explored in order to improve software evolution and traceability is seen as a core infrastructure to support impact analysis.

1.7 Scope of Work

Software traceability can be applied to some applications such as consistency-checking (Lucca et al., 2002) defect tracking (McConnel, 1997), cross referencing (Teng et al., 2004) and reuse (Ramesh and Jarke, 2001). The techniques and approaches used may differ from one another due to different objectives and feature requirements. Some of these approaches are geared toward system development while others are designed for system evolution.

In this scope of research, it needs to explore a software traceability specifically to support a change impact analysis within which it should be able to capture the impacts of change requests. The models and techniques used should allow the implementation of impacts across different workproducts. It needs to capture the software knowledge from the latest correct version of a *complete system* prior to implementation. It is assumed that a new traceability approach needs to develop some reverse engineering tools if ones are not available in the research community to support and simplify the capturing process.

The term a *complete system* here may refer to a very large scope as it may govern all the software models including the business model, specification, high level design, documentation, code, etc. These models are comprised within the software lifecycle of software specification, design, coding and testing. However, for the sake of research and implementation, this work will focus on some relevant information that includes a set of functional requirements, test cases, design and code as follows. These software components or artifacts are seen to be the *software development baseline* (MIL-STD-498, 2005). *Software development baseline* reflects the software products that are used and confined to the internal development staff rather than the external users or clients to support software evolution. Thus, this baseline model is chosen to represent a smaller scope of a large *complete system.*

The new work should be derived from a set of system documentation adhering to a certain software engineering standard e.g. MIL-STD-498 (MIL-STD-498, 2005). Nevertheless, it should not be tied up with the documentation environment as the main focus is not on the traceability and impact in system documentation but rather to its information contents. With simple interface (not within this scope) it should allow system documentation to view the traceable software components as a result of implementing software traceability system.

In this research approach, the scope is decided on object-oriented system to address the change impact analysis. It should be noted that this research is not concerned with correcting the software components but only reporting them. The

software engineers or maintainers should consider these results as an assistance to facilitate their initial prediction of change impact.

1.8 Thesis Outline

This thesis covers some discussions on the specific issues associated to software traceability for impact analysis and understanding how this new research is carried out. The thesis is organized in the following outline.

Chapter 2: Discusses the literature review of the software evolution and traceability. Few areas of interest are identified from which all the related issues, works and approaches are highlighted. This chapter also discusses some techniques of impact analysis and software traceability. Next, is a discussion on some existing models and approaches by making a comparative study based on a defined comparison framework. This leads to improvement opportunities that form a basis to develop a new proposed software traceability model.

Chapter 3: Provides a research methodology that describes the research design and formulation of research problems and validation considerations. This chapter leads to an overview of data gathering and analysis including benchmarking. It is followed by some research assumptions.

Chapter 4: Discusses the detailed model of the proposed software traceability for impact analysis. A set of formal notations are used to represent the conceptual model of the software traceability. It is followed by some approaches and mechanisms to achieve the model specifications.

Chapter 5: Presents the design and functionality of some developed tools to support the software traceability model. This includes the implementation of the design and component tools.

Chapter 6: The software traceability model is evaluated for its effectiveness, usability and accuracy. The evaluation criteria and methods are described and implemented on the model that includes modeling validation, a case study and experiment. This research performs evaluation based on quantitative and qualitative results. Quantitative results are checked against a benchmark set forth and qualitative results are collected based on user perception and comparative study made on the existing models and approaches.

Chapter 7: The statements on the research achievements, contributions and conclusion of the thesis are presented in this chapter. This is followed by the research limitations and suggestions for future work.

REFERENCES

Anderson, P., Reps T., Teitelbaum, T. and Zarins M. (2003). Tool Support for Fine-Grained Software Inspection. *IEEE Software*. 1-9.

Antoniol, G., Canfora, G.A., Gasazza, G., and Lucia, A. (2002). Recovering Traceability Links Between Code and Documentation. *IEEE Transactions on Software Engineering*. 28(10): 970-983.

Antoniol, G., Caprile, B., Potrich, A. and Tonella, P. (2000). Design-Code Traceability for Object Oriented Systems. *The Annals of Software Engineering*. 9: 35-58.

Antoniol, G., Penta, M.D. and Merlo, E. (2004). An automatic Approach to Identify Class Evolution Discontinuities. *Proceedings of the 7th International Workshop on Principles of Software Evolution*. IEEE Computer Society. 31-40.

Arnold, R.S. and Bohner, S.A. (1993). Impact Analysis – Towards A Framework for Comparison. *Proceedings of the Software Maintenance Conference*. September 27-30. USA: IEEE Computer Society. 292-301.

Bennett, K. (1996). Software Evolution: Past, Present and Future. *Information and Software Technology*. 38(11): 673-680.

Bennett, K., Rajlich, V. (2000). Software maintenance and Evolution: A Roadmap. *Proceedings of the Conference on the Future of Software Engineering*. May. USA: ACM Press. 73-87.

Bianchi, A. , Fasolino, A.R., Visaggio, G. (2000). An Exploratory Case Study of the Maintenance Effectiveness of Traceability Models. *International Workshop on Program Comprehension*. June 10-11. Ireland: IEEE Computer Society. 149-158.

Bieber M., Kukkonen H. and Balasubramanian V. (2000). Hypertext Functionality. *ACM Computing Surveys*. 1-6.

Bieman, J.M.and Kang, B.K. (1998). Measuring Design-Level Cohesion. *IEEE Transactions on Software Engineering.* 24(2): 111-124.

Boehm, B. and Sullivan, K. (2000). Software Economics: A Roadmap. *Proceedings of the Conference on the Future of Software Engineering.* ACM. 319-344.

Bohner, S.A. (1991). Software Change Impact Analysis for Design Evolution, *Eighth International Conference on Software Maintenance and Reengineering*. IEEE Computer Society Press. 292-301.

Bohner, S.A. (2002). Software Change Impacts: An Evolving Perspective. *Proceedings of the International Conference on Software Maintenance*. October 4-6. Canada: IEEE Computer Society. 263-271.

Bohner, S.A. and Arnold, R.S. (1996). *Software Change Impact Analysis*. California: IEEE Computer Society press.

Bohner, S.A. and Gracanin, D. (2003). Software Impact Analysis in a Virtual Environment, *Proceedings of the 28th Annual NASA Goddard Software Engineering Workshop*. December 3-4. USA: IEEE Computer Society. 143-151.

Booch, G., Rumbaugh, J. and Jacobson, I. (1999). *The Unified Modeling Language User Guide*. USA: Addison-Wesley.

Brand, V.D, Jong, H.A., Klint P. and Kooiker A.T. (2003). A Language Development Environment for Eclipse. *Proceedings of the 2003 Object Oriented Programming, Systems, Languages and Applications*. October. USA: ACM Press. 55-59.

Briand, L.C. and Pfahl, D. (2000). Using Simulation for Assessing the Real Impact of Test Coverage on Defect-Coverage. *IEEE Transactions on Reliability*. 49(1): 60-70.

Brown, A.B., Keller, A. and Hellerstein, J.L. (2005). A model of configuration complexity and its application to a change management system Integrated Network Management. *IFIP/IEEE International Symposium*. 631–644.

Buchsbaum, A., Chen, Y.F., Huang, H., Koutsofios, E., Mocenigo, J., Rogers, A. (2001). Visualizing and Analyzing Software Infrastructures. *IEEE Software*. 62-70.

Burd, E. and Munro, M. (1999). An initial approach towards measuring and characterizing software evolution. *Proceedings of Sixth Working Conference on Reverse Engineering*.168–174.

Canfora, G. (2004). Software Evolution in the era of Software Services. *Proceedings of the seventh International Workshop on Software Evolution (IWPSE'04)*. September 6-7. Japan: IEEE Computer Society. 9-18.

Chapin, N., Hale, J.E. and Khan, K.M. (2001). Types of software evolution and software maintenance. *Journal of Software Maintenance and Evolution: Research and Practice* 13(1). 3-30.

Chaumun, M.A., Kabaili, H., Keller, R.K., Lustman, F. and Saint-Denis, G. (2000). Design Properties and Object-Oriented Software Changeability. *Proceedings of the Fourth European Conference on Software Maintenance*. February 29-March 3. Switzerland: IEEE Computer Society. 45-54.

Chen, K., Rajlich, V. (2000). Case study of feature location using dependence graph. *Eigth International Workshop on Program Comprehension*. June 10-11. Ireland: IEEE Computer Society Press. 241-247.

Chen, X., Tsai, W.T., Huang, H. (1996). Omega – An integrated environment for C++ program maintenance. *IEEE Computer Society Press*. 114-123.

Columbus. (2005). *Analyzer Front End Company for Reverse Engineering Community*. Last accessed on October 24, 2005. http://www.frontendart.com

Cooper, D., Khoo, B., Konsky, B.R., Robey, M. (2004). Java Implementation Verification Reverse Engineering. *Australian Computer Science Conference*. Australian Computer Society. 26: 203-211.

Cui, L. and Wang, H. (2004). Version management of EAI based on the semantics of workflow. *Proceedings of the Eight International Conference on Computer Supported Cooperative Work in Design*. November 6-10. USA: ACM. 341 – 344.

Darcy, D.P and Kemerer, C.F. (2005). OO Metrics in Practice. *IEEE Software*. 22(6): 17 – 19.

Deraman, A. (1998). A framework for software maintenance model development. *Malaysian Journal of Computer Science* . 11(2): 23-31.

Desai, N., Lusk, A., Bradshaw, R. and Evard, R. (2003). BCFG: A Configuration Management Tool for Heterogeneous Environments. *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER'03)*. December 1-4. Hong Kong: IEEE Computer Society. 500-503.

Devandu, P.,Chen, Y.F., Muller, H. and Martin J. (1999). Chime: Customizable Hyperlink Insertion and Maintenance Engine for Software Engineering Environments. *Proceedings of the 21st International Conference on Software Engineering*. May 16-22. USA: IEEE Computer Society. 473-482.

Dorrow, K. (2003). Flexible Fault Tolerance in Configurable Middleware for Embedded Systems. *Proceedings of the 27th Annual International Computer Software and Applications Conference.* November 3-6. USA: IEEE Computer Society. 563-569.

Egyed, A. (2003). A Scenario-Driven Approach to Trace Dependency Analysis. *IEEE Transactions on Software Engineering.* 29(2): 324-332.

Engles, G. and Groenewegen, L. (2000). Object-Oriented Modeling: A Roadmap. *Proceedings of the International Conference on the Future of Software Engineering*. June 4-11. Ireland: ACM Press. 103-117.

Estublier, J., Leblang, D. and Clemm, G. (2004). Impact of the Research Community for the Field of Software Configuration Management. *International Conference of Software Engineering.* May 23-28. Scotland: ACM. 643-644.

Finkelstein, A. and Kramer, J. (2000). Software Engineering: A Roadmap. *Proceedings of the Conference on the Future of Software Engineering*. June 4-11. Ireland: ACM Press. 3-23.

Fiutem, R. and Antoniol, G. (1998). Identifying Design-Code Inconsistencies in Object-Oriented Software: A Case Study. *Proceedings of International Conference on Software Maintenance*. USA: IEEE Computer Society Press. 94-102.

French, J.C., Knight, J.C. and Powell, A.L. (1997). Applying Hypertext Structures to Software Documentation. *Information Processing and Management*. 33(2): 219-231.

Garlan, D. (2000). Software Architecture: A Roadmap. *Proceedings of the Conference on the Future of Software Engineering*. June 4-11. Ireland: ACM Press. 91-101.

Gotel, O. and Finkelstein, A. (1994). An Analysis of the Requirements Traceability Problem, *Proceedings of the First International Conference on Requirements Engineering*. USA: IEEE Computer Society. 94-101.

GrammaTech. (2005). *CodeSurfer Application*. Last accessed on October 24, 2005. http://www.grammatech.com/products/codesurfer/

Gupta, S.C., Nguyen, T.N. and Munson, E.V. (2003). The Software Concordance: Using a Uniform Document Model to Integrate Program Analysis and Hypermedia. *Proceedings of the Tenth Asia-Pacific Software Engineering*. December 10-12. Thailand: IEEE Computer Society. 164-173.

Hagen, V., McDonald, A., Atchison, B., Hanlon, A. and Lindsay, P. (2004). SubCM: A Tool for Improved Visibility of Software Change in an Industrial Setting. *IEEE Transactions on Software Engineering*. 30(10): 675-693.

Han, J. (2001). TRAM: A Tool for Requirements and Architecture Management. Proceedings of the 24th Australian Conference on Computer Science. Australia: *IEEE Computer Society*. 60-68.

Hartmann, J. Huang, S. and Tilley, S. (2001). Documenting Software Systems with View II: An Integrated Approach Based on XML. *Proceedings of Annual ACM Conference on Systems Documentation (SIGDOC 2001)*. October 21-24. USA: ACM. 237-246.

Hichins, M. and Gallagher, K. (1998). Improving visual impact analysis. *Proceedings of the International Conference on Software Maintenance*. November 16-20. USA: IEEE Computer Society. 294-301.

Hoffman, M.A. (2000). *A methodology to support the maintenance of object-oriented systems using impact analysis*. Louisiana State University: Ph.D. Thesis.

Horwitz, S., Reps, T. and Binkley, D. (1990). Interprocedural slicing using dependence graphs. *ACM Transactions on Progrramming Languages and Systems*. 12(1): 26-60.

Huang, J.C., Chang, C.K. and Christensen, M. (2003). Event-Based Traceability for Managing Evolutionary Change. *IEEE Transactions of Software Engineering*. 29(9): 796-810.

IEEE. (1998a). *IEEE Standard For Software Maintenance*. New York, IEEE Std. 1219-1998.

IEEE. (1998b). *IEEE Standard for Software Configuration Management Plans*. New York, IEEE-Std 828-1998.

IEEE. (1998c). *IEEE Standard for Software Test Documentation*. New York, IEEE-Std 829-1998.

IEEE. (1998d). *IEEE Recommended Practice For Software Requirements Specifications*. New York, IEEE-Std 830-1998.

IEEE. (1998e). *Guides- A Joint Guide Developed by IEEE and EIA- Industry Implementation of International Standard ISO/IEC 12207*. New York, IEEE/EIA 12207.

IEEE. (1998f). *IEEE Recommended Practice For Software Design Descriptions.* New York, IEEE-Std 1016-1998.

ISO/IEC 12207. (2005). *ISO/IEC 12207 Software Lifecycle Processes***.** Last accessed on October 24, 2005. www.abelia.com/docs/12207cpt.pdf

Jacobson, I., Booch, G., and Rumbaugh, J. (1999). *Unified Software Development Process.* USA: Addison-Wesley.

Janakiraman, G., Santos, J.R., Turner, Y. (2004). Automated System Design for Availability. *Proceedings of the 2004 International Conference on Dependable Systems and Networks*. June 28-July 1. Italy: IBM. 411-420.

Jang, K.Y., Munro, M., Kwon,Y.R. (2001). An Improved Method of Selecting regression Tests for C++ Programs. *Journal of Software Maintenance And Evolution: Research And Practice*. 13: 331-350.

J-STD-016-1995. (2005). *A Comparison of IEEE/EIA 12207, J-STD-016, and MIL-STD-498.* Last accessed on October 24, 2005. http://www.abelia.com/pubsmain.htm

Kaibaili, H., Keller, R.K. and Lustman, F. (2001). Cohesion as Changeability Indicator in Object-Oriented Systems. IEEE Computer Society. 39-46.

Keller, A. Hellerstein, J.L., Wolf, J.L. and Wu, K.L. (2004). The CHAMPS System: Change Management with Planning and Scheduling. *Network Operations and Management Symposium*. April 19-23. Korea: IEEE/IFIP. 395 – 408.

Kendall, K.E. and Kendall, J.E. (1998). *System Analysis and Design*. Fourth Edition. USA: Prentice Hall.

Kichenham, B.A., Travassos, G.H., Mayrhauser, A.V. and Schneidewind, N. (1999). Towards an Ontology of Software Maintenance. *Journal of Software Maintenance: Research and Practice*. 11: 365-389.

Kullor, C. and Eberlain, A. (2003). Aspect-Oriented Requirements Engineering for Software Product Lines. *Proceedings of the Tenth International Conference and Workshop on the Engineering of Computer-Based System.* USA: IEEE Computer Society. 98-107.

Kung, D.C., Liu, C.H. and Hsia, P. (2000). An object-oriented Web test model for testing Web applications. *The 24th Annual International Conference on Computer Software and Applications.* 537-542.

Lakhotia, A., Deprez, J.C. (1999). Restructuring Functions with Low Cohesion. *IEEE Computer Society Press.* 381-390.

Lazaro M. and Marcos E. (2005). Research in Software Engineering: Paragidms and Methods. PHISE'05.

Lee, J.K. and Jang, W.H. (2001). Component Identification Method with Coupling and Cohesion. *Proceedings of the Eight Asia-Pacific Software Engineering Conference*. December 4-7. China: IEEE Computer Society. 79-86.

Lee, M., Offutt, A.J. and Alexander, R. (2000). Algorithmic analysis of the impacts of changes to object-oriented software. *Proceedings of the 34th International Conference on Technology of Object-Oriented Languages and Systems*. July 30 - August 4. USA: IEEE Computer Society.. 61-70.

Lehman M.M. and Ramil J.F. (2000). Towards a Theory of Software Evolution – and Its Practical Impact. *International Symposium on Principles of Software Evolution. November 1-2*. Japan: IEEE Computer Society. 2-11.

Lehman, M.M. and Ramil, J.F. (2002). Software Evolution and Software Evolution Processes. *Annals of Software Engineering*. 14: 275-309.

Lindvall, M. (1997). Evaluating Impact Analysis - A case Study. *Journal of Empirical Software Engineering*. 2(2): 152-158.

Lindvall, M. and Sandahl, K. (1998). Traceability Aspects of Impacts Analysis in Object-Oriented System. *Journal of Software Maintenance Research and Practice.* 10: 37-57.

Lu X. and Yali G. (2003). Risk Analysis in Project of Software Development. *Proceedings of the Engineering Management Conference on Management Technologically Driven Organizations*. November 2040. New York: IEEE Computer Society. 72-75.

Lucca, G.A., Di Penta, M. and Gradara, S. (2002). An Approach to Classify Software Maintenance Requests. *Proceedings of the International Conference of Software Maintenance*. October 3-6. Canada: IEEE Computer Society. 93-102.

Maarek, Y., Berry, D. and Kaiser, G. (1991). An Information Retrieval Approach for Automatically Constructing Software Libraries. *IEEE Transactions of Software Engineering.* 17(8). 800-813.

Madhavji, N.H. (1992). Environment Evolution: The Prism Model of Changes. *IEEE Transaction on Software Engineering*. 8(5). 380-392.

Marcus A. and Meletic J.I. (2003). Recovering ocumentation-to-Sourcce-Code Traceability Links Using Latent Semantic Indexing. *Proceedings of the Twenty Fifth International Conference on Software Engineering*. May 3-10. USA: IEEE Computer Society. 125-135.

Mayrhauser, A.V. and Lang, S. (1999). A Coding Scheme to Support Systematic Analysis of Software Comprehension. *IEEE Transaction of Software Engineering*. 25(4). 526-540.

McCabe. (2005). *McCabe Software- Assuring Quality Throughout the Application Lifecycle*. Last accessed on October 24, 2005. http://www.mccabe.com

McConnel, S. (1997). Gauging Software Readiness with Defect Tracking. *IEEE Software*. 14(3): 136-135.

MIL-STD-498. (2005). *Roadmap for MIL-STD-498*. Last accessed on October 24, 2005. http://www2.umassd.edu/SWPI/DOD/MIL-STD-498/ROADMAP.PDF

Murphy, G.C., Notkin, D. and Sullivan, K. (1995). Software Reflexion Models: Bringing the Gap Between Source and High-Level Models. *Proceedings of Third Symposium Foundations of Software Engineering*. October 12-15. USA: ACM. 18-28.

Nam, C., Lim, J.L., Kang, S. Bae, Lee J.H. (2004). Declarative development of Web applications with active documents**.** Proceedings of the Russian-Korean International Symposium on Science and Technology. IEEE Computer Society. 68-72.

Natarajan, B., Gokhale, A., Yajnik, S. and Schmidt, D.C. (2000). DOORS: Towards High-Performance Fault Tolerant. *Proceedings of the Second International Symposium on Distributed Objects and Applications*. September 21-23. Belgium: IEEE Computer Society. 39 – 48.

Nguyen, T.N., Munson, E.V., Boyland, J.T. and Thao, C. (2004). Flexible Fine-grained Version Control for Software Documents. *Proceedings of the 11th Asia-Pacific Software Engineering*. November 30 - December 3. Korea: IEEE Computer Society. 212-219.

Nuseibeh, B. and Easterbrook S. (2000). Requirements Engineering: A Roadmap. *Proceedings of the International Conference on the Future of Software Engineering*. June 4-11. Ireland: ACM Press. 35-46.

Perry, D.E. (1994). Dimensions of Software Evolution. *Proceedings of the International Conference on Software Maintenance.* USA: IEEE Computer Society. 296-303.

Pressman. R.S. (2004). *Software engineering, a practitioner's approach*. Sixth ed. New York: Mc Graw Hill.

Rajlich, V. and Wilde, N. (2002). The Role of Concepts in Program Comprehension, *Proceedings of 10th International Workshop on Program Comprehension.* June 27-29. France: IEEE Computer Society. 271-278.

Ramesh, B. and Jarke, M. (2001). Toward Reference Models for Requirements Traceability, *IEEE Transactions on Software Engineering*. 27(1): 58-93.

Ramesh, B. (2002). Process Knowledge Management With Traceability, *IEEE Software*. 19(3): 50-52.

Rational (2005). *Rational Software Corporation*. Last accessed on October 24, 2005. www.rational.com.

Recon2. (2005). *Recon2 Tool for C Programmers*. Last accessed on October 24, 2005. http://www.cs.uwf.edu/~recon/recon2/

Rilling, J., Seffah, A., Bouthlier, C. (2002). The CONCEPT Project – Applying Source Code Analysis to Reduce Information Complexity of Static and Dynamic Visualization Techniques. *Proceedings of the First International Workshop on Visualization Software for Understanding and Analysis*. UK: IEEE Computer Society. 99-99.

Saemu, J., Prompoon, N. (2004). Tool and guidelines support for Capability Maturity Model's software subcontract management. *Proceedings of the 11th Asia-Pacific Software Engineering Conference.* 158–165.

Sefika, M., Sane A. and Campbellk, R.H. (1996). Monitoring Compliance of a Software System with its High-Level Design Models. *Proceedings of Eighteen International Conference of Software Engineering*. March 25-29. Germany: IEEE Computer Society. 387-396.

Singer, J. and Vinson, N.G. (2002). Ethical Issues in Emperical Studies of Software Engineering. IEEE Transactions on Software Engineering. 28(12): 1171-1180).

Small, A.W.; Downey, E.A. (2001). Managing Change: Some Important Aspects. *Proceedings of the Change Management and the New Industrial Revolution (IEMC'01)*. October 7-9. USA: IEEE Computer Society. 50-57.

Sneed, H.M. (2001). Impact Analysis of maintenance tasks for a distributed object-oriented system, *Proceedings of the International Conference on Software Maintenance.* November 6-10. Italy: IEEE Computer Society. 180-189.

Sommerville, I. (2004). *Software Engineering*. Seventh Ed. USA: Addison-Wesley Publishing Company.

Sulaiman, S. (2004). *A Document-Like Software Visualization Method for Effective Cognition of C-Based Software Systems*, Universiti Teknologi Malaysia: Ph.D. Thesis.

Teng, Q., Chen, X., Zhao, X., Zhu, W. and Zhang L. (2004). Extraction and visualization of architectural structure based on cross references among object files**.** *Proceedings of the 28th Annual International Conference on Computer Software and Applications.* September 27-30. Hong Kong: IEEE Computer Society. 508-513.

Tonella, P. (2003). Using a Concept Lattice of Decomposition Slices for Program Understanding and Impact Analysis, *IEEE Trans. Software Engineering*. 2(6): 495-509.

Turver, R.J. and Munro, M. (1994). An Early impact analysis technique for software maintenance, *Journal of Software Maintenance: Research and Practice*. 6 (1): 35-52.

Tvedt, R.T., Costa, P., Lindvall, M. (2002). Does the Code Match the Design? A Process for Architecture Evaluation. *Proceedings of the Eighteen International Conference on Software Maintenance*. October 4-6. Canada: IEEE Computer Society. 393-401.

Tvedt, R.T., Lindvall, M. and Costa, P. (2003). A Process for Software Architecture Evaluation Using Metrics. *Proceedings of the 27th Annual NASA Goddard/IEEE Software Engineering Workshop*. December 5-6. Maryland: NASA Goddard/IEEE Software. 175-182.

Wan Kadir, W.N. (2005). *Business Rule-Driven Object-Oriented Design*. University of Manchester: Ph.D. Thesis.

Webster, P.B., Oliveira, K.M. and Anquetil, N. (2005). A Risk Taxonomy Proposal for Software Maintenance. *Proceedings of the 21st IEEE International Conference on Software Maintenance*. 453-461.

Wei, Y., Zhang, S., Zhong, F. (2003). A Meta-Model for Large-Scale Software System. *International Conference on Systems, Man and Cybernatics*. October 5-8. USA: IEEE Computer Society. 501-505.

Weiser, M. (1984). Program slicing. *IEEE Transactions on Software Engineering*. 10(4): 352-357.

White, B.A. (2000). *Software Configuration Management Strategies and Rational ClearCase*. USA: Addison-Wesley.

Wilde, N., Buckellew, M., Page, H., Rajlich, V. (2001). A case study of feature location in unstructured legacy Fortran code. *Proceedings of the Fifth European Conference on Software Maintenance and Reengineering*. March 14-16. Portugal: IEEE Computer Society. 68-76.

Wilde, N., Casey, C. (1996). Early Field Experience with the Software Reconnaissance Technique for Program Comprehension. *Proceedings of the Third Workshop Conference of Reverse Engineering*, November 8-10. USA: IEEE Computer Society. 270-276.

Wilde, N., Gomez, J.A., Gust, T. and Strasburg D. (1992). Locating User Functionality in Old Code. *Proceedings of International Conference on Software Maintenance.* November 9-12. Sweden: IEEE Computer Society. 200-205.

Snelting G. (2000). Understanding Class Hierarchies Using Concept Analysis. ACM Transactions on Programming Languages and Systems. 22(3):123-135.

Walrad, C. and Strom, D. (2002). The Importance of Branching Models in SCM. *Journal of Computer*. 35(9): 31-38.

Wilkie, F.G. and Kitchenham, B.A. (1998). Coupling Measures and Change Ripples in C++ Application Software. Proceedings of the Third International Conference on Empirical Assessment and Evaluation in Software Engineering. April 12-14. UK: IEEE Computer Society. 112-130.

Zelkowitz, M.V. and Wallace. D.R. (1998). Experimental Models for Validating Technology. IEEE Computer. 31(2): 23-31.

Zhoa, J. (2002). Change impact analysis to support architectural evolution. *Software Maintenance and Evolution: Research and Practice.* 14: 317-333.