# A DESIGN DOCUMENT GENERATOR TO SUPPORT A LINKAGE BETWEEN CODE AND DOCUMENT

IZZUL HIDAYAT NAISAN

A thesis submitted in fulfillment of the
requirements for the award of the degree of
Master of Science (Computer Science)

Faculty of Computing
Universiti Teknologi Malaysia

JULY 2013

***Dedicated to…***

*My beloved parents:*

*Naisan Yunus and Aisyah Umar*

*Even billions of thanks are not enough for your endless love, hope, and advice.*

*My sisters: Susanti, Rifqiyati, and Iin Fajrulhuda*

*To my brother: Ikhwanul Muslimin*

*Who always give the motivation to move on and growing better.*

*My beloved wife: Fariza Octrina*

*For all inspiration and encouragement, and may all things run well.*

*And to all friends and colleagues*

*Thanks for giving me spirit, support and enlightenment.*

# ABSTRACT

In recent years, more than half of current companies are still working on old systems. Software maintenance, testing, quality assurance, reuse, and integration are only a few examples of software engineering activities that involve old systems. A key aspect related to all these processes is the identification of the components of a system and the comprehension of the existing linkage between components. One way to identify these components of system is by analyzing its documentation. However, in many cases, documentation is a missing or obsolete item in old software maintenance. As the software technology advances evolve over time, there is a need to create and maintain a relationship between source code and documentation. This linkage needs to be maintained to ensure that software documentation remains consistent and up-to-date with code. This consistency will assist quality and reduce maintenance work. A new relationship model is proposed between documentation and low level software that includes the code and its associated structures and physical components. The significant benefits of this research can be observed at its ability to support document viewers to manage and maintain the existing system. A prototype was developed and an evaluation was carried out to realize the proof of concept. The results show that the proposed model is significant and provides a useful access between code and documentation.

# ABSTRAK

Sejak beberapa tahun kebelakangan ini, lebih daripada separuh syarikat-syarikat masih menggunakan sistem lama. Penyelenggaraan, pengujian, kepastian kualiti, guna-semula dan integrasi perisian adalah antara contoh-contoh aktiviti kejuruteraan perisian yang melibatkan sistem lama. Satu kunci aspek yang mengaitkan semua proses-proses ini adalah mengenalpasti komponen-komponen dan pemahaman terhadap jejakan yang sedia ada antara komponen-komponen. Salah satu cara untuk mengenalpasti komponen-komponen sistem ini adalah dengan menganalisa dokumen-dokumennya. Walau bagaimanapun, dalam banyak kes, dokumen didapati hilang atau pupus untuk pemeliharaan perisian legasi. Dalam kemajuan teknologi perisian selaras dengan perkembangan masa, terdapatnya suatu keperluan untuk membina dan menyelenggara jejakan antara sumber kod dan dokumen. Jejakan ini perlu diselenggara untuk memastikan dokumen perisian adalah sejajar dan terbaru atau dikemaskini bersama dengan kod. Wajaran ini akan membantu kualiti dan mengurangi kerja pemeliharaan. Model jejakan yang baru dicadangkan, antara dokumen dan aras rendah perisian termasuklah kod itu dan struktur berkaitan dengannya dan komponen-komponen fizikal. Kebaikan yang signifikan daripada penyelidikan ini boleh diperhatikan pada kemampuannya untuk menyokong pembinaan dokumen dalam menguruskan dan memelihara sistem yang sedia ada. Satu prototaip dibina dan diujikaji bagi mengesahkan konsep penyelidikan. Keputusan menunjukan bahawa model yang dicadangkan adalah signifikan dan menyediakan akses yang baik antara kod dan dokumentasi.

# TABLE OF CONTENTS


| CHAPTER | TITLE | PAGE |
|---|---|---|
| | **DECLARATION** | ii |
| | **DEDICATION** | iii |
| | **ACKNOWLEDGEMENT** | iv |
| | **ABSTRACT** | v |
| | **ABSTRAK** | vi |
| | **TABLE OF CONTENTS** | vii |
| | **LIST OF TABLES** | xi |
| | **LIST OF FIGURES** | xii |
| | **LIST OF ABBREVIATION** | xiv |
| | **LIST OF APPENDICES** | xv |
| 1 | **INTRODUCTION** | 1 |
| | 1.1 Introduction | 1 |
| | 1.2 Background of the Problem | 1 |
| | 1.3 Statement of the Problem | 3 |
| | 1.4 Objectives of the Study | 4 |
| | 1.5 Importance of Study | 4 |
| | 1.6 Scope of Work | 4 |
| | 1.7 Thesis Organization | 5 |
| | 1.8 Summary | 6 |

| | | | |
|---|---|---|---|
| **2** | **LITERATURE REVIEW** | | 7 |
| | 2.1 | Introduction | 7 |
| | 2.2 | Software Maintenance | 7 |
| | | 2.2.1 Definition of Software Maintenance | 8 |
| | | 2.2.2 Software Maintenance Categories | 9 |
| | | 2.2.3 Software Maintenance Processes | 10 |
| | 2.3 | Software Documentation | 11 |
| | | 2.3.1 Software Redocumentation | 14 |
| | | 2.3.2 Software Redocumentation Process | 15 |
| | | 2.3.3 Some Existing Tools on Software Redocumentation | 15 |
| | | 2.3.4 Summary of Software Redocumentation Tools | 18 |
| | 2.4 | Reverse Engineering | 19 |
| | | 2.4.1 Definition of Reverse Engineering | 20 |
| | | 2.4.2 Reverse Engineering Process | 20 |
| | | 2.4.3 Some Reverse Engineering Techniques | 22 |
| | | 2.4.4 Some Existing Tools on Reverse Engineering | 27 |
| | | 2.4.5 Summary of Reverse Engineering Tools | 30 |
| | 2.5 | Software Design | 32 |
| | 2.6 | Unified Modeling Language | 33 |
| | 2.7 | Software Documentation and Code Linkage | 34 |
| | 2.8 | Critical Evaluation of Existing Code and Documentation Linkage Approach | 35 |
| | 2.9 | Summary | 41 |
| **3** | **RESEARCH METHODOLOGY** | | 42 |
| | 3.1 | Introduction | 42 |
| | 3.2 | Research Design and Procedure | 42 |
| | 3.3 | Literature Review and Preparation Phase | 44 |
| | 3.4 | System Analysis and Design | 46 |
| | 3.5 | Prototype Development Phase | 47 |
| | 3.6 | Validation and Testing Phase | 48 |
| | 3.7 | Assisting Tools | 50 |
| | 3.8 | Summary | 51 |

| | | | |
|---|---|---|---|
| **4** | **MODELING AND DESIGN** | | 52 |
| | 4.1 | Introduction | 52 |
| | 4.2 | Overview of Documentation Generator System | 52 |
| | 4.3 | Prototype Architecture | 54 |
| | 4.4 | Prototype Classes | 55 |
| | 4.5 | Components in Documentation Generator Process | 58 |
| | | 4.5.1 Code Parser | 58 |
| | | 4.5.2 XML Reader | 63 |
| | | 4.5.3 Content and Diagram Manager | 66 |
| | | 4.5.4 Doc-Generator | 67 |
| | | 4.5.5 Code-Doc Connector | 68 |
| | | 4.5.6 Finder (Search) | 69 |
| | 4.6 | Documentation Generator Linkage Model | 70 |
| | 4.7 | Summary | 72 |
| **5** | **IMPLEMENTATION** | | **73** |
| | 5.1 | Introduction | 73 |
| | 5.2 | sddGen Implementation and User Interfaces | 73 |
| | 5.3 | Assisting Tools | 83 |
| | | 5.3.1 Code Parser | 83 |
| | | 5.3.2 Diagram Generator | 87 |
| | 5.4 | Summary | 90 |
| **6** | **FUNCTIONAL TESTING AND COMPARISON BETWEEN ARCHITECTURE** | | **91** |
| | 6.1 | Introduction | 91 |
| | 6.2 | Case Study | 92 |
| | | 6.2.1 Outlines of Case Study | 92 |
| | | 6.2.2 OBA Project Briefing | 93 |
| | 6.3 | Testing Criteria | 94 |
| | 6.4 | Testing Results | 94 |
| | 6.5 | Comparison of Existing and Proposed Software Linkage Approaches | 100 |
| | 6.6 | Summary | 102 |

**7 CONCLUSION AND FUTURE WORKS** 103

7.1 Introduction 103

7.2 Research Summary and Achievements 103

7.3 Contribution 105

7.4 Research Limitation and Future Works 106

7.5 Summary 107

**REFERENCES** 108

Appendix A – B 114- 162

# LIST OF TABLES


| TABLE NO. | TITLE | PAGE |
|---|---|---|
| 2.1 | Comparison of Software Redocumentation Tools | 19 |
| 2.2 | Comparative Study of Software Linkage Approaches | 39 |
| 3.1 | Research Questions, Objectives, Activities, and Deliverables | 44 |
| 4.1 | Translation of XML Input | 64 |
| 5.1 | Data Manipulation Language Syntaxes | 81 |
| 5.2 | Some Data Definition Language Syntaxes | 81 |
| 5.3 | Sample Table of Methods | 82 |
| 5.4 | Example Result-set of SELECT LIKE Query | 83 |
| 5.5 | Output Formats Directly Supported by Doxygen | 86 |
| 5.6 | Output Formats Indirectly Supported by Doxygen | 86 |
| 6.1 | List of Classes from Universal Report | 94 |
| 6.2 | List of Classes from Doxygen | 95 |
| 6.3 | List of Classes from Doxys | 96 |
| 6.4 | List of Classes from sddGen | 97 |
| 6.5 | Evaluation Results based on Bellay and Gall (1998) Criteria | 98 |
| 6.6 | Comparison of Existing and Proposed Approaches | 101 |

# LIST OF FIGURES


| FIGURE | TITLE | PAGE |
|---|---|---|
| 2.1 | Waterfall Software Development Life Cycle (Hung, 2009) | 8 |
| 2.2 | IEEE Maintenance Process | 10 |
| 2.3 | ISO Maintenance Process | 11 |
| 2.4 | Documentation Tree | 12 |
| 2.5 | Software Redocumentation Processes | 15 |
| 2.6 | Universal Report Output | 16 |
| 2.7 | Doxygen Interface | 17 |
| 2.8 | Javadoc Output | 18 |
| 2.9 | Reverse Engineering Process | 21 |
| 2.10 | CREP (Tortorella and Visaggio, 1997) | 22 |
| 2.11 | MDA-based Reverse Engineering | 25 |
| 2.12 | Search Based Reverse Engineering (Mitchell et al., 2002) | 26 |
| 2.13 | Rigi Output (Rigi Group, 2009) | 28 |
| 2.14 | Columbus Interface | 28 |
| 2.15 | CodeSurfer Output (Grammatech, 2009) | 30 |
| 2.16 | Hierarchy of UML diagrams (Wikipedia, 2009) | 34 |
| 2.17 | ENVISION Architecture (Zhou, 2008) | 36 |
| 2.18 | Traceability Tool Design (Asuncion, 2007) | 37 |
| 2.19 | CATIA Architecture (Ibrahim, 2006) | 38 |
| 3.1 | Activities in Literature Review and Preparation Phase | 45 |
| 3.2 | Activities in System Analysis and Design Phase | 47 |

| | | |
|---|---|---|
| 4.1 | Overview of Documentation Generator Process Model | 53 |
| 4.2 | *sddGen* Architecture | 54 |
| 4.3 | Classes of *sddGen* | 56 |
| 4.4 | Phase 1: Code Parser | 59 |
| 4.5 | Code Parser Architecture | 61 |
| 4.6 | Phase 2: XML Reader | 63 |
| 4.7 | XML file generated by code parse | 64 |
| 4.8 | Phase 3: Doc Generator Process | 67 |
| 4.9 | Phase 3: Doc-Generator Linkage | 70 |
| 5.1 | sddGen Home Screen | 74 |
| 5.2 | Content Manager | 75 |
| 5.3 | Diagram Manager | 75 |
| 5.4 | XML Reader | 76 |
| 5.5 | Files | 77 |
| 5.6 | Classes | 78 |
| 5.7 | Generator | 79 |
| 5.8 | Find Artifacts | 80 |
| 5.9 | Project Settings of Doxygen | 84 |
| 5.10 | Operating Mode of Doxygen | 85 |
| 5.11 | Doxygen Output Formats | 85 |
| 5.12 | Running Window of Doxygen | 87 |
| 5.13 | StarUML | 88 |

# LIST OF APPENDICES


| APPENDIX | TITLE | PAGE |
|---|---|---|
| A | Evaluation Steps | 114 |
| B | Source Code | 151 |

# CHAPTER 1

# INTRODUCTION

## 1.1 Introduction

This chapter explains about the overview of the thesis. It delivers a brief introduction to the study conducted. The topics tackled in this chapter are: problems background, problem statement, objectives of study, importance of study, work scope, and thesis outline.

## 1.2 Background of the Problem

It is clearly known that software systems have played a great role to a lot of companies in this era. The business processes have become so much involved with systems of a computer that a break down or interupption of the essential software system can cause loss of huge amount (Fahmi and Choi, 2007).

Nowadays, many companies are working on old systems. . Software maintenance, reverse engineering, quality assessment, components recycle and integration are only a few examples of software engineering activities relating old systems. A fundamental aspect related to all these processes is identification of artifacts used in a system and interpretation of relationships among them (Canfora and Di Penta, 2007).

One way to identify the components of system is by analyzing its documentation and see how it is relevant and consistent to software system. The documentation is important to interpret a system at a certain level of abstraction, in a circumscribed amount of time. It is necessitated, for example, when a system is relocated or re-engineered. It is able to be utilized to map functional change requests as communicated by end users onto technical change requests, and to calculate the cost of such change. Eventually, documentation will be needed in the process of outsourcing maintenance or when the new members need to learn about the environment (Van Deursen and Kuipers, 1999).

However, in many cases, documentation is a missing item in the maintenance of old software systems, many parts of documentation are obsolete and no longer up to date (Anquetil and De Oliviera, 2006). As current technology evolves, there is an emergence need for the corresponding documentation and source code to be synchronous each other. Unsynchronous relationship between them can cause a company to spend more time, money, and resource. Very often, maintainers team must work from the code to the result of any other available source of information in reverse order i.e. re-engineer the code to design abstract level. This work is very frustating as it is very difficult to handle, unless some preliminary study has been done. De Souza et al. (2005) reported that from 40% to 60% of the maintenance activity is spent on learning the software to interpret it and how the planned modification may be implemented. This is a costly operation that required researchers to give a special attention on document and code relationships.

This thesis is looking into a possibility of moving forward to support maintenance by addressing some problems associated to relationship between documentation and code. It enables maintenance team get the latest associated components of the software evolves over time. It brings easy maintenance and to documentation, which may otherwise become obsolete. Software components that are well-documented are easier to comprehend and consequently easier to reuse and make it maintainable. This is particularly useful in large software systems where the dependent modules and structures are more complicated to comprehend (Schugerl et al, 2009).

## 1.3 Statement of the Problem

Documentations which were traditionally prepared by developers in some cases are inconsistent as some change requests, updates, or bugs fixing somehow are not well updated with code. Developers tend to be focusing on source code rather than the documentation. Code is considered the most reliable source of software and evolve faster over time whereas its change is not synchronous and properly handled at documentation level. Unless some component linkage has been established, it is not easy to update the documentation manually. Some parts still require human interaction to complete the relationship (Van Deursen and Kuipers, 1999).

This research posits associated source code and documentation that may provide a better understanding of an existing system by maintenance team The connection built may make documentation consistent with the code at all times. The hypothesis leads to the following research questions.

The main research question is "*How to create and maintain a linkage between the source code and documentation in order to make them consistent and maintainable?"*

The sub-questions of the main research question are as follows.

(i) What are the issues with old systems pertaining to maintenance?

(ii) Why the use of current documentation still cannot satisfy the demand of work during software maintenance?

(iii) How can we develop documentation generator to support a linkage between source code and documentation?

(iv) How to test the functionality of the proposed prototype and compare it along with existing tools?

## 1.4 Objectives of the Study

Based on the problem statements mentioned above, this research encompasses a set of objectives that is associated to the milestones of research process. The research objectives are mentioned below.

(i) To investigate the issues and ways forward to improve the current system documentation.
(ii) To formulate and design a linking approach between code and documentation.
(iii) To develop a prototype to support the approach.
(iv) To test the functionality of proposed protoype and compare it along with existing tools.

## 1.5 Importance of Study

A lot of changes may cause the current documentation becomes irrelevant that makes maintenance team work harder to trace back the changes through the source code. Reconstructing and effectively generating the design documentation of existing software systems is even more difficult than initial design (Wong et al, 1995). Therefore, in order to reduce this problem, creating a linkage between source code and documentation is useful.

## 1.6 Scope of Work

In order to accomplish the objectives of this study, it is important to identify the scope, which covers the following aspects.

(i) The research subject of a case study will focus on some code written in an object-oriented programming language.

# REFERENCES

Al-Kilidar, H., Cox, K., and Kitchenham, B. (2005). The use and usefulness of the ISO/IEC 9126 quality standard. *International Symposium on Empirical Software Engineering*. 17-18 November.

Anderson, P. (2004). CodeSurfer/Path Inspector. *Proceedings of the 20th IEEE International Conference on Software Maintenance*. 11-14 September 2004. 508.

Anderson, P. and Zarins, M. (2005). The CodeSurfer software understanding platform. *Proceedings of the 13th International Workshop on Program Comprehension (IWPC'05)*. 15-16 May 2005.

Anquetil, N., K. M. Oliveira, and Dias, M.G.B. (2006). *Software Maintenance Ontology*. In *Ontologies for Software Engineering and Software Technology* (pp. 153-173). Berlin: Springerlink Berlin Heidelberg.

Appleton, B. (2009). *A Software Design Specification Template*. Last accessed on February 24, 2009. http://www.cmcrossroads.com/bradapp/docs/sdd.html.

Asuncion, H.U., Francois, F., Taylor, Richard N. (2007). An end-to-end industrial software traceability tool. *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. Dubrovnik, Croatia: ACM.

Bellay, B. and Gall, H. (1997). A comparison of four reverse engineering tools. *Proceedings of the Fourth Working Conference on Reverse Engineering*. 1997.

Bellay, B. and Gall, H. (1998). An evaluation of reverse engineering tool capabilities. *Journal of Software Maintenance* 10(5), 305-331. Wiley InterScience.

Booch, G., Rumbaugh, J., and Jacobson, I. (1999). *The Unified Modeling Language user guide*, Addison Wesley Longman Publishing Co., Inc.

Canfora, G. and Cimitile, A. (2001). *Software Maintenance*. In *Handbook of Software Engineering and Knowledge Engineering I*. World Scientific Pub.

Canfora, G. and Di Penta, M. (2007). New Frontiers of Reverse Engineering. *Future of Software Engineering (FOSE '07)*. 2007. IEEE, 326-341.

Chikofsky, E. J. and Cross, J. H. II (1990). Reverse engineering and design recovery: a taxonomy. *Software*. 7(1), 13-17. IEEE.

Chu, W. C., Lu, C. W., Chang, C.H., Chung, Y.C., Liu, X. Yang, H. (2002). *Reverse Engineering*. In *Handbook of Software Engineering and Knowledge Engineering II*. World Scientific Pub.

De Souza, S. C. B., Anquetil, N., de Oliveira, K.M. (2005). A study of the documentation essential to software maintenance. *Proceedings of the 23rd annual international conference on Design of communication: documenting & designing for pervasive information*. 2005. Coventry, United Kingdom: ACM, 68-75.

Dennis, S. M. (2003). Project in applied software engineering. *J. Comput. Small Coll*. 18(6), 22-27. ACM.

Din, D. (2009). *A Source Code Query to Support Structured Program Understanding*. Master of Science. Universiti Teknologi Malaysia, Skudai.

Douglas, K. (1999). API documentation from source code comments: a case study of Javadoc. *Proceedings of the 17th annual international conference on Computer documentation*. 1999. New Orleans, Louisiana, United States: ACM.

Fahmi, S. A. and Ho-Jin, C. (2007). Software Reverse Engineering to Requirements. *International Conference on Convergence Information Technology*. 2007.

Favre, L. (2008). Formalizing MDA-Based Reverse Engineering Processes. *Sixth International Conference on Software Engineering Research, Management and Applications (SERA '08).* 2008.

Favre, L. and Pereira, C. (2008). Formalizing MDA-Based Refactorings. *19th Australian Conference on Software Engineering (ASWEC 2008)*. 2008.

Ferenc, R., Beszedes A., Tarkiainen, M., and Gyimothy, T. (2002). Columbus - reverse engineering tool and schema for C++. *Proceedings of International Conference on Software Maintenance*. 2002. IEEE.

FrondEndt Art Ltd. (2009). *Columbus/CAN*. Last accessed on February 6, 2009. http://frontendart.com/products_col.php.

Grammatech. (2009). *CodeSurfer Screenshoots*. Last accessed on February 12, 2009. http://www.grammatech.com/products/codesurfer/screenshots.html.

Heesch, D. V. (2009). *Doxygen.* Last accessed on February 8, 2009. http://www.stack.nl/~dimitri/doxygen/.

Hung, T. (2009). *Software Development Process*. Last accessed on February 3, 2009. http://cnx.org/content/m14619/latest/.

Ibrahim, S. (2006). *A Document-Based Software Traceability to Support Change Impact Analysis of Object-Oriented Software*. Doctor of Philosophy. Universiti Teknologi Malaysia, Skudai.

IEEE (1990). *IEEE Standard Glossary of Software Engineering Terminology*. *IEEE Std 610.12-1990*. New York: IEEE.

IEEE (1998). *IEEE Standard for Software Maintenance. IEEE Std 1219-1998*. New York: IEEE.

ISO/IEC (2001). *Software Engineering – Product quality – Part1: Quality Model. ISO/IEC 9126-1*. Geneva, ISO/IEC.

Klump, R. (2001). Understanding object-oriented programming concepts. *Power Engineering Society Summer Meeting*. 2001. IEEE.

Lavanya, K. C., Bala, K., Mohanty, H., and Shyamasundar, R.K. (2005). How Good is a UML Diagram? A Tool to Check It. *TENCON 2005*. 2005. IEEE Region 10.

Lazaro, M. and Marcos, E. (2005). Research in Software Engineering: Paradigms and Methods. *Philosophical Foundations of Information Systems Engineering*. 2005. Porto, Portugal.

Lee, M. L. (1995). *Change Impact Analysis of Object-Oriented Software.* Master Degree. George Mason University, Washington.

Lientz, B. P. and Swanson, E. B. (1980). *Software Maintenance Management*. Reading, Massachusetts: Addison-Wesley.

Martin, R. C. (1995). *Designing Object-Oriented C++ Applications Using the Booch Method*. New Jersey: Prentice Hall.

Microsoft (2009). *Sandcastle – Documentation Compiler for Managed Class Libraries*. Last accessed on February 11, 2009. http://www.codeplex.com/Sandcastle.

Mitchell, B. S., Mancoridis, S., and Martin, T. (2002). Search based reverse engineering. *Proceedings of the 14th international conference on Software engineering and knowledge engineering*. Ischia, Italy: ACM.

Mohamad, R. N. (2009). *A Program Visualization to Support Change Impact Analysis*. Master of Science. Universiti Teknologi Malaysia, Skudai.

Muller, H. A. and K. Klashinsky (1988). Rigi: a system for programming-in-the-large.., *Proceedings of the 10th International Conference on Software Engineering*. 1988. IEEE: 80-86.

NASA (1991). *NASA Software Documentation Standard. NASA-STD-2100-91.* Washington: NASA.

Nosek, J. T. and Palvia, P. (1990). Software maintenance management: changes in the last decade. *Journal of Software Maintenance* 2(3), 157-174. ACM.

Object Management Group. (2009). *Unified Modeling Language: Superstructure, Version 2.0. OMG Specification: formal/2007-02-03 2007.* Last accesed on February 16, 2009. http://www.omg.org/docs/formal/07-02-03.pdf.

Pankratius, V., Stucky, W., and Vossen, G. (2005). Aspect-oriented re-engineering of e-learning courseware. *The Learning Organization*. 12(5), 457-470. Emerald

Polo, M. and Piattini, M. (1999). MANTEMA: a software maintenance methodology based on the ISO/IEC 12207 standard. *Proceedings of Fourth IEEE International Symposium and Forum on Software Engineering Standards.* 1999.

Powell, Gavin. 2006. *Beginning XML Databases (Wrox Beginning Guides).* Wrox Press Ltd., Birmingham, UK, UK.

Pressman, R. S. (1992). *Software Engineering: A Practitioner's Approach*. New York: McGraw-Hill.

Ramesh, B. and Jarke, M. (2001). Toward reference models for requirements traceability. *IEEE Transactions on Software Engineering*. 27(1), 58-93. IEEE.

Rostkowycz, A. J., Rajlich, V., and Marcus, A. (2004). A case study on the long-term effects of software redocumentation. *Proceedings of 20th IEEE International Conference on Software Maintenance, 2004.*

Schugerl, P., Rilling, J., and Charland, P. (2009). Beyond generated software documentation: A web 2.0 perspective. *IEEE International Conference on Software Maintenance 2009 (ICSM 2009)*.

Sukanya, R., Susan-Elliott, S., and Derek, J.R. (2009). Cross-artifact traceability using lightweight links. *Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering*. IEEE.

Sun Microsystems (2009). *Javadoc Tool Home Page*. Last accessed on February 19, 2009. http://java.sun.com/j2se/javadoc/.

Tadonki, C. (2004). Universal Report: A Generic Reverse Engineering Tool. *12th IEEE International Workshop on Program Comprehension*. Bari, Italy: IEEE, 266-267.

Teitelbaum, T. (2000). CodeSurfer. *SIGSOFT Softw. Eng. Notes*. 25(1): 99.

Thames Corp. (2006). *Automobile Onboard Autocruise*. France.

Tilley, S. (2008). Three Challenges in Program Redocumentation for Distributed Systems. *IEEE International Systems Conference 2008 (SysCon 2008).* Montreal, Canada: IEEE.

Tortorella, M. and Visaggio, G. (1997). CREP-Characterizing Reverse Engineering Process component methodology. *Proceedings of International Conference on.Software Maintenance*. 1997.

Universal Software (2009). *Universal Report*. Last accessed on February 9, 2009. http://www.omegacomputer.com/.

Vaclav, R. (1997). Incremental Redocumentation with Hypertext. *Proceedings of the 1st Euromicro Working Conference on Software Maintenance and Reengineering (CSMR '97)*. IEEE.

Van Deursen, A. and Kuipers, T. (1999). Building documentation generators. *Proceedings of IEEE International Conference on Software Maintenance, 1999. (ICSM '99)*. IEEE.

Wikipedia (2010). *Software Design*. Last accessed on March 25, 2010. http://en.wikipedia.org/wiki/Software_design.

Wong, K., Tilley, S. R., Muller, H.A., and Storey, M.A.D. (1995). Structural redocumentation: a case study. *Software*. 12(1), 46-54. IEEE.

Yuliana, O.Y.; Chittayasothorn, S.; , "A Conceptual Schema Based XML Schema with Integrity Constraints Checking," *Convergence and Hybrid Information Technology, 2008. ICHIT '08. International Conference on*, vol., no., pp.19-24, 28-30 Aug. 2008

Zelkowitz, M. V. and Wallace, D. R. (1998). Experimental Models for Validating Technology. *Computer*. 31(5), 23-31

Zhou, X., Huo Z., Huang, Y., and Xu, J. (2008). Facilitating Software Traceability Understanding with ENVISION. *Proceeding of 32nd Annual International Conference in Computer Software and Applications (COMPSAC '08)*. Turku, Finland: IEEE, 295-302.