TRACEABILITY APPROACH FOR MANAGING CHANGES INVOLVING SOFTWARE TESTING ARTEFACTS

OTHMAN MOHD YUSOP

A thesis submitted in fulfilment of the
requirements for the award of the degree of
Doctor of Philosophy (Computer Science)

Faculty of Computing
Universiti Teknologi Malaysia

FEBRUARY 2017

ALHAMDULILLAH Praise be to Allah, and may His peace and blessings be upon Muhammad s.a.w,

For my beloved parents Hjh Asmah Basir, Haji Mohd Yusop Nami, Hjh Sainah Dahlan, my beloved wife, Noraini binti Shaari and my bubbly little daughter, Dhia Humaira' binti Othman who have given me the strength and courage.

ACKNOWLEDGEMENT

I would like to take this opportunity to thank my main supervisor, Prof. Dr. Suhaimi Ibrahim for his encouragement, advice and inspiration throughout this research. Special thanks go to colleges and staff of Advanced Informatics School, Universiti Teknologi Malaysia for your constant support, technical guidance and constructive review of this research work, who involved directly or indirectly in the project.

A special thanks to Haji Azri Haji Azmi, Saiful Adli Ismail, Mdm Haslina Sarkan, Dr. Mohd Nazri Kama, Dr. Norziha Megat Tajuddin, Dr. Suriayati Chuprat, Mdm Yazriwati Yahya, Dr. Ganthan for motivational words, expertises, and relentlessly persuading me for this project submission. Your invaluable advices are highly appreciated.

I would like to extend my many thanks to Prof. Dr. Abdul Samad Haji Ismail the Dean of Faculty of Computing, Prof. Madya Dr. Norfidah Ithnin and management team of Faculty of Computing on your efforts and patience for allowing myself to complete this project.

A great gratitude also goes to the Universiti Teknologi Malaysia and Kementerian Pengajian Tinggi for sponsoring my three year PhD study.

OMY, Presint 18, Putrajaya

ABSTRACT

Software change is inevitable for software product to remain relevant and reusable. As software evolves over time due to specific changes at any point in time during software development and maintenance, the managing aspect of changes may get more complicated and risky. The outdated links would cause the affected artefacts to be not updated timely and effectively. Most of the existing traceability approaches and tools are dedicated and limited to high level artefacts such as requirements and fewer capability made available to address the lower level artefacts such as classes and codes. Most maintainers limit their links to begin at the requirement perspective but there is no valid traceability link being made to support the fine grained level involving testing components. This thesis proposes a new traceability approach to manage changes with the emphasis on the integration of the development artefacts and testing artefacts. The working artefacts cover requirements, packages, classes, methods, test case, and codes. The proposed approach provides a know-how solution to the IEEE 829:2010 standards associated to testing that demands for the support at testing perspective. This approach has the capability to horizontally and vertically manage artefacts from requirement down to code and vice versa. The proposed traceability approach was applied to a case study of a software development project called On-Board Automobile (OBA) with a complete set of documentation including test cases. The evaluation results prove that the proposed traceability approach is significant and useful in managing software changes involving testing artefacts.

ABSTRAK

Pindaan perisian tidak dapat dielakkan bagi produk perisian untuk ianya kekal berfungsi dan berkebolehan untuk diguna semula. Oleh kerana perisian berubah sepanjang masa disebabkan wujudnya pindaan tertentu pada mana-mana tempat semasa pembangunan perisian dan penyenggaraan, aspek pengurusan pindaan boleh menjadi lebih rumit dan berisiko. Jalinan yang luput mungkin menyebabkan artifak yang terlibat tidak dapat dikemaskini dalam masa yang ditetapkan dan menjadikannya tidak berkesan. Kebanyakan pendekatan jejak semasa dan alatan adalah khusus dan terhad kepada artifak aras tinggi seperti keperluan berbanding sokongan untuk menangani artifak aras lebih rendah seperti kelas dan kod. Kebanyakan penyenggara menghadkan jalinan jejak untuk bermula pada perspektif keperluan tetapi tiada jalinan jejak yang sah untuk menyokong aras butiran halus yang melibatkan komponen pengujian. Tesis ini mencadangkan pendekatan jejak yang baharu untuk menguruskan pindaan dengan penekanan kepada integrasi terhadap artifak pembangunan dan artifak pengujian. Artifak yang diusahakan merangkumi keperluan, pakej, kelas, kaedah, kes pengujian dan kod. Pendekatan yang dicadangkan menyediakan penyelesaian pengetahuan untuk IEEE 829: 2010 standard berkaitan dengan pengujian yang memerlukan sokongan pada perspektif pengujian. Pendekatan ini mempunyai keupayaan untuk menguruskan secara jalinan mendatar dan jalinan menegak untuk urusan artifak daripada fasa keperluan ke kod dan sebaliknya. Pendekatan jejak yang dicadangkan ini diaplikasikan dengan bantuan kajian kes projek pembangunan perisian yang dipanggil *On-Board Automobile* (OBA) termasuk satu set dokumentasi yang lengkap bagi kes-kes pengujian. Keputusan penilaian membuktikan bahawa keberkesanan pendekatan yang dicadangkan adalah signifikan dan berguna dalam menguruskan perubahan perisian yang melibatkan artifak pengujian.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Overview

This chapter introduces the research and elaborates its background in detail and subsequently describes the problem statement, research objectives, research scopes and the significance of this study.

1.2 Background Problem

It is inevitable for the software to evolve in order to cope with emerging changes. Changes in software could mostly would occur due to internal or external requests i.e. user's requests. User change requirements can occur after the delivery of the complete software artefacts such as a system, documentations, or during the progression of the software development phases involving requirement, design and testing. Hence it is widely accepted within software engineering community that software changes or software evolution is a part of software maintenance process. The term software evolution and software maintenance are used interchangeably and in short practitioners classify it as maintenance (Bennett and Rajlich, 2000). In maintenance process, traceability is important. Facilitating the changes during maintenance using traceability is a crucial step in ensuring the affected artefacts across software development phases, are well maintained (Rochimah *et al.*, 2009; Shahid and Ibrahim, 2013).

During the maintenance phase, some activities needs to be managed. These activities are maintaining traceability, change management, and maintaining affected software artefacts. Prior to implementing any changes, ones need to comprehend before and after changes take place. For example, changes made at testing level

or any other level throughout software development phases, often occurred without updating the relevant documentations (Mäder and Egyed, 2012). Due to this negligence, it automatically causes the traceability link between the affected artefacts becoming obsolete. In addition, some organizations are neglecting the traceability implementation due to laborious works that need to be considered (Lormans *et al.*, 2004). In spite of the existence of change management tools to update the traceability link, the maintenance routine they need to incur, is not practical and time consuming (Heindl and Biffl, 2005). Most of the change management tools support either at file structure or at source code file level and whenever an update gets introduced at either level, the traceability links that supposed to be maintained, are left forgotten and outdated. Hence, updating the traceability link upon changes among the artefacts at higher and lower level are often a failure whenever there is a change occurred (Cleland-Huang *et al.*, 2003).

In the context of the bigger system, there are possibly a lot of artefacts need to be managed and tested. Each of the changes made has to be retested and verified to ensure changes realisation. Traceability ensures the right artefacts get amended and tested. Poorly maintenance of the traceability link due to an increase of testing activities as an example, can affect negatively on the cost of maintenance (Sherba, 2005; Tamai and Kamata, 2009). Freese (2003), stated the expenditure for software maintenance can increase to as much as 80% of the total lifecycle cost of a software system. In addition, traceability maintenance requires consistent updating and due to the nature of larger system having many artefacts to maintain, improper and imprecise traceability approach could happen (Sherba, 2005). Therefore traceability link maintenance is considered as an essential element of software development lifecycle (Cleland-Huang *et al.*, 2014).

Changes can span from coarse artefacts (i.e. SDP, SRS, SDD, STD, and STR) to fine-grained artefacts (i.e. test objects, classes, methods) (Omar, 2013; Shahid and Ibrahim, 2016). The problem may arise while doing maintenance if software changes impacting more than a single software artefact. Thus, it causes more than one artefacts need to be amended accordingly. The number of software changes can grow from single artefacts to many artefacts. In order to manage and resolve many changed artefacts, managing via traceability must be implemented. Currently, many works have been done on focusing changes from requirement perspective (Ibrahim *et al.*, 2005; Rochimah *et al.*, 2009; Omar, 2013; Shahid and Ibrahim, 2013). There are other works too that focusing changes from other phases in software development lifecycle i.e. component level (Mei *et al.*, 2002), modelling language (Murta *et al.*, 2008a),

methods and classes level (Junqueira *et al.*, 2008), architectural level (Nguyen *et al.*, 2005), and test-driven (Freese, 2003). The latter was focusing on broader perspective, specifically at process level instead of file structure or fine-grained artefacts. (Cleland-Huang *et al.*, 2014) even added in their research studies on the current trend of software traceability, in which they quoted the USA Federal Aviation Administration (FAA) and USA Food and Drug Administration (FDA) are emphasising on software traceability from source code to software requirement and therefore it is clear indication of the current traceability trend, there is still a lack of consideration from testing artefacts perspective.

From traceability issues, some challenges that are related to tracing the prospect impacting software artefacts can occur as pointed out by this study (Ibrahim *et al.*, 2005). The main problem to maintainer is that seemingly small changes can ripple-effect throughout the system to cause substantial impact elsewhere. Besides tracing the changes from one artefact to another across entire software development lifecycle, establishing traceability itself is rarely done (Grechanik *et al.*, 2007).

Based on the above scenarios, it is evident at the time of this research was conducted, there are fewer efforts done to manage affected testing artefacts via traceability. Implementing changes can be exhaustive, frustrating and costly due to the amount of laborious manual works and the issues will worsen if the software developers are clueless of how to identify the affected artefacts due to poor traceability practices and undesirable change management. Therefore, it is an important necessary to establish traceability link to manage changes made from any phases of software development process that particularly might affect testing artefacts as well. The traceability establishment could span from the requirement to testing phases and vice versa and the link will not be limited to different phases but the establishment of the traceability link will accommodate artefacts within the same phase of software development lifecycle.

1.3 Problem Statement

In maintenance process, managing changes through traceability link will ensure the affected artefacts i.e. functional and non-functional requirements, design model and component, and test artefacts get amended accordingly as the new change was introduced. In the real process, the work will be more simplified if the developers or

maintainers can find their own way to establish the traceability link between the high level and lowest level artefacts.

Current traceability approaches as discussed in section 1.2, have shown limited coverage of traceability links whereas changes could happen from a testing perspective. Due to highlighted issues, extending the traceability coverage is essential and will be made helpful to support changes from testing standpoint as well. Therefore this research will focus on finding support through traceability approach whilst managing the affected artefacts from testing perspective.

This research is intended to deal with the lack of traceability support on test artefacts as discussed in previous section. The support will cover artefacts from a higher level of abstraction to the lowest level of abstraction. It will be done through the establishment of traceability either horizontally (across different phases of the lifecycle) or vertically (within a phase in the lifecycle) and the output of this research is expected of an improvement of managing involving test artefacts via traceability approach. Hence the hypothesis leads to this research question:

"How affected test artefacts due to changes across and within phases of software development lifecycle could be managed effectively through traceability approach?"

To be able to answer the above questions, a set of sub-questions is formed below to provide detail insight of the outlining research problem:

RQ1: Why current traceability approaches do not satisfy the developer/tester during software testing lifecycle?

RQ2: How to facilitate changes that are potentially affected and propagated to other artefacts during managing the evolving test artefacts?

RQ3: How traceability approach will improve the developers/testers tasks in coping up with changes across the entire phases and within the same phase of software development lifecycle?

RQ4: How to validate the effectiveness of the proposed traceability approach to some significant level?

1.4 Objectives of the Study

The research objectives based on the problem statement, are as follows:

(i) To study and investigate the current issues of traceability approaches to manage changes involving test artefacts.

(ii) To develop a new traceability approach that support affected changes across phases and within a phase in software development lifecycle as well as from coarse level to the fine-grained level of artefacts.

(iii) To design and formulate algorithms to support the proposed approach.

(iv) To evaluate the effectiveness of the proposed approach against the existing approaches through obtained results.

1.5 Scope of the Study

This section describes the boundaries of this research. The scope of this study covers the following:

(i) This research focuses on existing traceability approaches that relate to managing affected test artefacts upon changes, either explicitly or implicitly across or within phases of software development lifecycle.

(ii) This study covers from coarse level artefacts (i.e. SDP, SRS, SDD, STR, and STD) to fine-grained level (i.e. requirement indexes, objects, classes, components, packages, methods, test cases, test suites and data elements) and all artefacts have a unique item identifications which conformed to (MIL-STD-498, 2005) documentation standard. Even though these research artefacts seem to bound to a specific documentation but the importance of this study is more on managing the artefacts via traceability itself.

(iii) A medium size (approximately above 5000 usable line of code/LOC) case study based on the objected oriented approach will be adopted to address issues of managing the involving artefacts from testing perspective. The LOC was measured using an opened source tool, LOCMetrics by McCabe Metric (McCabe, 1976) and the technique was adopted by Software Engineering Institute (SEI) for LOC counting standards (Nguyen *et al.*, 2007).

(iv) The effectiveness of the traceability approach will be quantitatively measured using precision and recall from Information Retrieval (IR) field. As these two parameters are used to prove an efficacy of any approach that adopts IR technique (Buckland and Gey, 1994).

1.6 Significant of the Study

Harrold (2009) stated in her research, the most expensive activities could occur after software development completion is software retesting during maintenance testing phase. Further quote, 50% of software maintenance budget will consume by retesting activities. In software testing alone, a big chunk of budget, at estimation nearly 80% will spend for retest. Retest is needed to ensure change imposed on the system, does not propagate to the untouched features. Building up a model that can trace and foresee the candidate impacted artefacts is crucial during software maintenance and furthermore, maintenance testing is the most non-trivial part of the maintenance activities. To build up such a model, there are features needed to be considered; a traceability features. Traceability is necessary due to its capability to establish links among the artefacts. Time reduction whilst performing maintenance testing is an important factor too, hence the automated proposed approach.

Tracing and managing ever grow software artefacts due to changes are seemingly never ending maintenance activities. Updating the traceability links due to changes in any phases is making the software system itself growing, expanding, evolving, etc. Changes are inevitable due to factors namely, outdated software system, new platform of operating system, change requirements, new project management approach, new developers'techniques or methodologies, etc.

Poor in identifying which artefacts being affected due to changes indicate that traceability approach used is weak and poor. The issue gets further worse if the software artefacts are bigger in volume which involves many lines of source codes, documents, etc. Hence this study will provide a support for developers at testing lifecycle specifically maintenance testing, to tracing back which test component or test objects, test cases are affected due to changes in requirement. Found bugs will be included and tightly coupled with test cases/test script that initiated the bugs. The latter will be stored together with the test case inside a repository. In addition, in order to manage the evolution of the artefacts, the repository will be used to store the

traceability links update.

This study is targeting the developer's awareness of how important to keep their artefacts manageable and traceable right at their finger tips. Thus eliminating time and cost consuming at the later stage of maintenance testing phases. The test managers and testers would be having a complete transparent view from tester's perspective (abstract level) right down into developer's perspective (logical level) i.e. tracing bugs and the management level can make firm decision over the changes execution.

1.7 Thesis Outline

This thesis discusses on specific issues associated with managing involving test artefacts via software traceability. It highlights the limitation of current approaches in resolving the outdated traceability links upon software changes from a testing perspective. This thesis is organised as follows:

Chapter 2: Discusses the literature review about change management, existing traceability approaches and managing test artefacts during software change. This chapter also highlights some limitations of the existing traceability approaches that support changes from testing perspective. A comparative study was tabulated and limitations of the existing approaches are highlighted in table form. This study will lead to an opportunity for improvement in proposing a new software traceability approach.

Chapter 3: Highlights a research methodology that discusses the research design, formulation of research procedures and activities and the theoretical framework. This chapter also discusses on research instruments, evaluation criteria, assumption and limitation.

Chapter 4: Presents an explanation of the conceptual detailed of the new traceability approach in managing affected test artefacts during changes. A set of formal notations is used to represent the conceptual part of the approach. This is followed by a detailed discussion of the proposed approach. It explains two part of traceability; horizontal (explicit) traceability and vertical (implicit) traceability. This chapter explains the design and functionality of a developed tools to support

REFERENCES

Aizenbud-Reshef, N., Nolan, B. T., Rubin, J. and Shaham-Gafni, Y. (2006). Model traceability. *IBM Syst. J.* 45(3), 515–526.

Al-Kilidar, H., Cox, K. and Kitchenham, B. (2005). The use and usefulness of the ISO/IEC 9126 quality standard. In *Empirical Software Engineering, 2005. 2005 International Symposium on*. IEEE, 7–pp.

Anderson, K. M., Sherba, S. A. and Lepthien, W. V. (2002). Towards large-scale information integration. In *Proceedings of the 24th International Conference on Software Engineering*. Orlando, Florida: ACM. ISBN 1-58113-472-X, 524–534. doi:10.1145/581339.581403.

Antoniol, G., Canfora, G., Casazza, G., Lucia, A. D. and Merlo, E. (2002). Recovering Traceability Links between Code and Documentation. *IEEE Trans. Softw. Eng.* 28(10), 970–983.

Beizer, B. (1990). *Software testing techniques (2nd ed.)*. Van Nostrand Reinhold Co. ISBN 0-442-20672-0.

Bennett, K. H. and Rajlich, V. T. (2000). Software maintenance and evolution: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering*. ACM. ISBN 1-58113-253-0, 73–87. doi:10.1145/336512.336534.

Bertolino, A. (2007). Software Testing Research: Achievements, Challenges, Dreams. In *2007 Future of Software Engineering*. IEEE Computer Society. ISBN 0-7695-2829-5, 85–103.

Boehm, B. W. (1979). Software engineering. In *Classics in software engineering*. (pp. 323–361). Yourdon Press. ISBN 0-917072-14-6.

Booch, G., Rumbaugh, J. and Jacobson, I. (1996). The unified modeling language. *Unix Review*. 14(13), 5.

Booth, Haas, McCabe, Newcomer, Champion, Ferris and Orchard (2005). *Web Services Architecture - W3C Working Group Note*.

Briand, L. C., Labiche, Y. and Yue, T. (2009). Automated traceability analysis for UML model refinements. *Inf. Softw. Technol.* 51(2), 512–527.

Buckland, M. K. and Gey, F. C. (1994). The relationship between recall and precision. *JASIS*. 45(1), 12–19.

Buckley, J., Mens, T., Zenger, M., Rashid, A. and Kniesel, G. (2005). Towards a taxonomy of software change. *Journal of Software Maintenance and Evolution: Research and Practice*. 17(5), 309–332. ISSN 1532-0618. doi:10.1002/smr.319.

Burnstein, I. (2003). *Practical Software Testing*. Springer Professional Computing. Springer. ISBN 978-0-387-95131-7.

Cai, J.-p., Lu, W.-l. and Xu, W.-y. (2012). A New Method of Version Management for Test Case. In Zhu, R. and Ma, Y. (Eds.) *Information Engineering and Applications*. (pp. 530–537). No. 154 in Lecture Notes in Electrical Engineering. Springer London. ISBN 978-1-4471-2385-9 978-1-4471-2386-6.

Choices, C. (2000). Marketing Technology to Software Practitioners. *IEEE Computer Society Press Los Alamitos, CA, USA*. Volume 17(Issue 1), Pages: 27 – 33. ISSN ISSN:0740-7459.

Cleland-Huang, Dekhtyar, A., Hayes, J. and Antoniol, G. (2006). *Grand challenges in traceability.* Technical Report Technical Report COET-GCT-06-01-0.9.

Cleland-Huang, J., Chang, C. and Christensen, M. (2003). Event-based traceability for managing evolutionary change. *IEEE Transactions on Software Engineering*. 29(9), 796–810. ISSN 0098-5589. doi:10.1109/TSE.2003.1232285.

Cleland-Huang, J., Gotel, O. C., Huffman Hayes, J., MÃd'der, P. and Zisman, A. (2014). Software traceability: trends and future directions. In *Proceedings of the on Future of Software Engineering*. ACM, 55–69.

Cleland-Huang, J., Hayes, J. H. and Domel, J. M. (2009). Model-based traceability. In *Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering*. IEEE Computer Society. ISBN 978-1-4244-3741-2, 6–10.

Cleland-Huang, J., Settimi, R., BenKhadra, O., Berezhanskaya, E. and Christina, S. (2005). Goal-centric traceability for managing non-functional requirements. In *Proceedings of the 27th international conference on Software engineering*. St. Louis, MO, USA: ACM. ISBN 1-59593-963-2, 362–371.

Collins-Sussman, Fitzpatrick, B., B.W. and Pilato (2004). *C.M. Version Control with Subversion*. O'Reilly.

Davis, J. and Goadrich, M. (2006). The relationship between Precision-Recall and ROC curves. In *Proceedings of the 23rd international conference on Machine learning*. ACM, 233–240.

De Lucia, A., Oliveto, R., Zurolo, F. and Di Penta, M. (2006). Improving

comprehensibility of source code via traceability information: a controlled experiment. In *Proceedings of the 14th IEEE International Conference on Program Comprehension*. 317–326.

Dekeyser, J., Boulet, P., Marquet, P. and Meftali, S. (2005). Model driven engineering for SoC co-design. In *IEEE-NEWCAS Conference, 2005. The 3rd International*. 21–25.

Dekhtyar, A., Hayes, J. H. and Antoniol, G. (2007). Benchmarks for traceability?

Dirckze, R. (2002). *Java Metadata Interface (JMI) - Version 1.0, Unisys Corporation and Sun Microsystems*.

Egyed, A. (2001). A scenario-driven approach to traceability. In *Proceedings of the 23rd International Conference on Software Engineering*. Toronto, Ontario, Canada: IEEE Computer Society. ISBN 0-7695-1050-7, 123–132.

Estublier, J., Leblang, D., Clemm, G., Conradi, R., Hoek, A. v. d., Tichy, W. and Wiborg-Weber, D. (2002). Impact of the research community for the field of software configuration management. In *Proceedings of the 24th International Conference on Software Engineering*. Orlando, Florida: ACM. ISBN 1-58113-472-X, 643–644. doi:10.1145/581339.581423.

Freese, T. (2003). Towards Software Configuration Management for Test-Driven Development. In *Software Configuration Management*. (pp. 143–150).

Gall, H., Hajek, K. and Jazayeri, M. (1998). Detection of Logical Coupling Based on Product Release History. In *Proceedings of the International Conference on Software Maintenance*. IEEE Computer Society. ISBN 0-8186-8779-7, 190.

Gayer, S., Herrmann, A., Keuler, T., Riebisch, M. and Antonino, P. O. (2016). Lightweight Traceability for the Agile Architect. *Computer*. 49(5), 64–71. ISSN 0018-9162. doi:10.1109/MC.2016.150.

Gotel, O., Cleland-Huang, J., Hayes, J. H., Zisman, A., Egyed, A., GrÃijnbacher, P., Dekhtyar, A., Antoniol, G., Maletic, J. and MÃd'der, P. (2012). Traceability fundamentals. In *Software and Systems Traceability*. (pp. 3–22). Springer.

Gotel, O. and Finkelstein, C. (1994). An analysis of the requirements traceability problem. In *Requirements Engineering, 1994., Proceedings of the First International Conference on*. 94–101.

Graham, D., Van Veenendaal, E., Evans, I. and Black, R. (2008). *Foundations of software testing: ISTQB certification*. Course Technology Cengage Learning.

Grechanik, M., McKinley, K. S. and Perry, D. E. (2007). Recovering and using use-case-diagram-to-source-code traceability links. In *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. ISBN 978-1-59593-811-4, 95–104.

Guess, V. C. (2002). *CMII of Business Process Infrastructure*. Hollis Publishing Company. ISBN 0972058206.

Harrold, M. J. (2000). Testing: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering*. Limerick, Ireland: ACM. ISBN 1-58113-253-0, 61–72. doi:10.1145/336512.336532.

Harrold, M. J. (2009). Reduce, reuse, recycle, recover: Techniques for improved regression testing. In *2009 IEEE International Conference on Software Maintenance*. September. Edmonton, AB, Canada, 5–5. doi:10.1109/ICSM.2009.5306347.

Hayes, J. H., Dekhtyar, A. and Sundaram, S. K. (2006). Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods. *IEEE Trans. Softw. Eng.* 32(1), 4–19.

Heindl, M. and Biffl, S. (2005). A case study on value-based requirements tracing. In *Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*. ACM, 60–69.

Hetzel, B. (1993). *The Complete Guide to Software Testing*. (2nd ed.). Wiley. ISBN 0471565679.

Ibrahim, N.B. Idris, M.Munro and A.Deraman (2005). A Requirements Traceability to Support Change Impact Analysis. *Asian Journal of IT (AJIT) vol.4(4)*.

Jones, C. (1991). *Applied software measurement: assuring productivity and quality*. McGraw-Hill, Inc. ISBN 0-07-032813-7.

Junqueira, D. C., Bittar, T. J. and Fortes, R. P. M. (2008). A fine-grained and flexible version control for software artifacts. In *Proceedings of the 26th annual ACM international conference on Design of communication*. Lisbon, Portugal: ACM. ISBN 978-1-60558-083-8, 185–192. doi:10.1145/1456536.1456576.

Kagdi, H., Maletic, J. I. and Sharif, B. (2007). Mining Software Repositories for Traceability Links. In *Proceedings of the 15th IEEE International Conference on Program Comprehension*. IEEE Computer Society. ISBN 0-7695-2860-0, 145–154.

Kasse, T. and A. MCQuaid, P. (2000). Software Configuration Management for Project Leaders. *ASQ*. 2.

Kazman, R., Abowd, G., Bass, L. and Clements, P. (1996). Scenario-Based Analysis of Software Architecture. *IEEE Softw.* 13(6), 47–55.

Keyes, J. (2004). *Software Configuration Management*. Auerbach Publications. ISBN 0849319765.

Kitchenham, B., Linkman, S. and Law, D. (1997). DESMET: a methodology for evaluating software engineering methods and tools. *Computing Control Engineering Journal*. 8(3), 120–126. ISSN 0956-3385. doi:10.1049/cce:19970304.

Knethen, A. v. (2002). Change-Oriented Requirements Traceability: Support for Evolution of Embedded Systems. In *Proceedings of the International Conference on Software Maintenance (ICSM'02)*. IEEE Computer Society. ISBN 0-7695-1819-2, 482.

Larsson, M., Larsson, M. and Larsson, M. (2000). Applying Configuration Management Techniques to Component-Based Systems.

Lawrie, D. J., Feild, H. and Binkley, D. (2006). Leveraged quality assessment using information retrieval techniques. In *14th International Conference on Program Comprehension*. 149–158.

Lázaro, M., Marcos, E. and Juan, R. (2005). Research in Software Engineering: Paradigms and Methods.

Lin, J., Lin, C. C., Cleland-Huang, J., Settimi, R., Amaya, J., Bedford, G., Berenbach, B., Khadra, O. B., Duan, C. and Zou, X. (2006). Poirot: A Distributed Tool Supporting Enterprise-Wide Automated Traceability. In *Proceedings of the 14th IEEE International Requirements Engineering Conference*. IEEE Computer Society. ISBN 0-7695-2555-5, 356–357.

Lormans, M., Van Dijk, H., Van Deursen, A., Nocker, E. and de Zeeuw, A. (2004). Managing evolving requirements in an outsourcing context: an industrial experience report. In *Software Evolution, 2004. Proceedings. 7th International Workshop on Principles of*. IEEE, 149–158.

Lucia, A. D., Fasano, F., Oliveto, R. and Tortora, G. (2007). Recovering traceability links in software artifact management systems using information retrieval methods. *ACM Trans. Softw. Eng. Methodol.* 16(4), 13. doi:10.1145/1276933.1276934.

Mäder, P. and Egyed, A. (2012). Assessing the effect of requirements traceability for software maintenance. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. IEEE, 171–180.

Mäder, P., Gotel, O. and Philippow, I. (2008). Rule-Based Maintenance of Post-Requirements Traceability Relations. In *2008 16th IEEE International*

Requirements Engineering Conference. September. Barcelona, Spain, 23–32. doi: 10.1109/RE.2008.24.

Mäder, P., Philippow, I. and Riebisch, M. (2007). Customizing Traceability Links for the Unified Process. *Lecture Notes in Computer Science*. 4880, 53.

Maletic, J. I., Collard, M. L. and Simoes, B. (2005). An XML based approach to support the evolution of model-to-model traceability links. In *Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering*. Long Beach, California: ACM. ISBN 1-59593-243-7, 67–72.

Maletic, J. I., Munson, E. V., Marcus, A. and Nguyen, T. N. (2003). Using a hypertext model for traceability link conformance analysis. In *Proc. of the Int. Workshop on Traceability in Emerging Forms of Software Engineering*. 47–54.

Manning, C. D., Raghavan, P., SchÃijtze, H. and others (2008). *Introduction to information retrieval*. vol. 1. Cambridge university press Cambridge.

Marcus, A. and Maletic, J. I. (2003). Recovering documentation-to-source-code traceability links using latent semantic indexing. In *Proceedings of the 25th International Conference on Software Engineering*. Portland, Oregon: IEEE Computer Society. ISBN 0-7695-1877-X, 125–135.

McCabe, T. J. (1976). A complexity measure. *Software Engineering, IEEE Transactions on*. (4), 308–320.

McMaster, S. and Memon, A. M. (2009). An Extensible Heuristic-Based Framework for GUI Test Case Maintenance. In *2009 International Conference on Software Testing, Verification, and Validation Workshops*. April. Denver, CO, USA, 251–254. doi:10.1109/ICSTW.2009.11.

Mei, H., Zhang, L. and Yang, F. (2001). A software configuration management model for supporting component-based software development. *SIGSOFT Softw. Eng. Notes*. 26(2), 53–58. doi:10.1145/505776.505790.

Mei, H., Zhang, L. and Yang, F. (2002). A component-based software configuration management model and its supporting system. *Journal of Computer Science and Technology*. 17(4), 432–441. doi:10.1007/BF02943283.

Memon, A. M. and Soffa, M. L. (2003). Regression testing of GUIs. In *Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering*. Helsinki, Finland: ACM. ISBN 1-58113-743-5, 118–127. doi: 10.1145/940071.940088.

MIL-HDBK-61A (1997). MILITARY HANDBOOK. *SCM Process*. 12(5.1), 2–2.

MIL-STD-498 (1995). Mil-std-498. *Crosstalk, the Journal of Defense Software Engineering*. 8(2), 2–5.

MIL-STD-498 (2005). MIL-STD-498 ROADMAP.

Miller, K. and Voas, J. (2006). Software test cases: is one ever enough? *IT Professional*. 8(1), 44–48. ISSN 1520-9202.

Mlynarski, M., GÃijldali, B., SpÃd'th, M. and Engels, G. (2009). From design models to test models by means of test ideas. In *Proceedings of the 6th International Workshop on Model-Driven Engineering, Verification and Validation*. Denver, Colorado: ACM. ISBN 978-1-60558-876-6, 1–10. doi:10.1145/1656485.1656492.

Mohan, K., Xu, P., Cao, L. and Ramesh, B. (2008). Improving change management in software development: Integrating traceability and software configuration management. *Decision Support Systems*. 45(4), 922–936. ISSN 0167-9236. doi: 10.1016/j.dss.2008.03.003.

Murta, L., CorrÃła, C., PrudÃłncio, J. G. and Werner, C. (2008a). Towards odyssey-VCS 2: improvements over a UML-based version control system. In *Proceedings of the 2008 international workshop on Comparison and versioning of software models*. Leipzig, Germany: ACM. ISBN 978-1-60558-045-6, 25–30. doi:10.1145/1370152.1370159.

Murta, L., Oliveira, H., Dantas, C., Lopes, L. G. and Werner, C. (2007). Odyssey-SCM: An integrated software configuration management infrastructure for UML models. *Sci. Comput. Program.* 65(3), 249–274.

Murta, L. G., Hoek, A. and Werner, C. M. (2008b). Continuous and automated evolution of architecture-to-implementation traceability links. *Automated Software Engg.* 15(1), 75–107.

Naslavsky, L., Ziv, H. and Richardson, D. J. (2007). Towards traceability of model-based testing artifacts. In *Proceedings of the 3rd international workshop on Advances in model-based testing*. London, United Kingdom: ACM. ISBN 978-1-59593-850-3, 105–114.

Naslavsky, L., Ziv, H. and Richardson, D. J. (2009). A model-based regression test selection technique. In *2009 IEEE International Conference on Software Maintenance*. September. Edmonton, AB, Canada, 515–518. doi:10.1109/ICSM.2009.5306338.

Nguyen, T., Munson, E. and Thao, C. (2005). Object-oriented configuration management technology can improve software architectural traceability. In *Third ACIS International Conference on Software Engineering Research, Management and Applications, 2005*. August. 86–93. doi:10.1109/SERA.2005.54.

Nguyen, V., Deeds-Rubin, S., Tan, T. and Boehm, B. (2007). A SLOC counting standard. In *COCOMO II Forum*, vol. 2007.

Oliveto, R., Antoniol, G., Marcus, A. and Hayes, J. (2007). Software Artefact Traceability: the Never Ending Challenge. *IEEE*. (ICSM 2007).

Omar, S. F. b. (2013). *A SOFTWARE TRACEABILITY APPROACH TO SUPPORT REQUIREMENT BASED TEST COVERAGE ANALYSIS*. Ph.D. Thesis. Universiti Teknologi Malaysia.

Osterweil, L. (1996). Strategic directions in software quality. *ACM Comput. Surv.* 28(4), 738–750. doi:10.1145/242223.242288.

Overmyer, S. P. (1990). DoD-Std-2167A and methodologies. *ACM SIGSOFT Software Engineering Notes*. 15(5), 50–59.

Palmer, J., R.H., T. and M., D. (2000). "Traceability," in Software Requirement Engineering. 2nd Edition. *IEEE Computer Society Press Los Alamitos, CA, USA*, 412–422.

Parveen, T., Tilley, S. and Gonzalez, G. (2007). A case study in test management. In *Proceedings of the 45th annual southeast regional conference*. Winston-Salem, North Carolina: ACM. ISBN 978-1-59593-629-5, 82–87. doi:10.1145/1233341.1233357.

Pinheiro, F. A. (2004). Requirements traceability. *Perspectives on software requirements*, 91–113.

Poshyvanyk, D. and Marcus, A. (2007). Using traceability links to assess and maintain the quality of software documentation. *Proc. of TEFSE*. 7, 27–30.

Powers, D. M. (2011). Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation.

Pressman, R. S. (1992). *Software Engineering (3rd Ed.): A Practitioner's Approach*. New York, NY, USA: McGraw-Hill, Inc. ISBN 0-07-050814-3.

Pressman, R. S. (2005). *Software engineering: a practitioner's approach*. Palgrave Macmillan.

Qusef, A., Bavota, G., Oliveto, R., De Lucia, A. and Binkley, D. (2014). Recovering test-to-code traceability using slicing and textual analysis. *Journal of Systems and Software*. 88, 147–168. ISSN 0164-1212. doi:10.1016/j.jss.2013.10.019.

Rafati, A., Lee, S. P., Parizi, R. M. and Zamani, S. (2015). A Test-to-code Traceability Method Using .NET Custom Attributes. In *Proceedings of the 2015 Conference on Research in Adaptive and Convergent Systems*. RACS. New York, NY, USA: ACM. ISBN 978-1-4503-3738-0, 489–496. doi:10.1145/2811411.2811553.

Rajlich and Gosavi (2002). A Case Study of Unanticipated Incremental Change. In *Proceedings of the International Conference on Software Maintenance (ICSM'02)*. IEEE Computer Society. ISBN 0-7695-1819-2, 442.

Ren, Y., Xing, T., Quan, Q. and Zhao, Y. (2010). Software Configuration Management of Version Control Study Based on Baseline. In *2010 International Conference on Information Management, Innovation Management and Industrial Engineering (ICIII)*, vol. 4. November. 118–121. doi:10.1109/ICIII.2010.506.

Rijsbergen, C. J. V. (1979). *Information Retrieval*. (2nd ed.). Newton, MA, USA: Butterworth-Heinemann. ISBN 978-0-408-70929-3.

Robson, C. (2002). *Real World Research: A Resource for Social Scientists and Practitioner-Researchers (Regional Surveys of the World)*. {Blackwell Publishing Limited}. ISBN 0631213058.

Rochimah, S., Wan Kadir, W. M. N. and Abdullah, A. H. (2009). Multifaceted Requirement Traceability Approach to Support Software Evolution. *5th PostGraduate Annual Seminars - PARS*.

Rumbaugh, J., Jacobson, I. and Booch, G. (2004). *Unified Modeling Language Reference Manual, The*. Pearson Higher Education.

Sengupta, S., Kanjilal, A. and Bhattacharya, S. (2008). Requirement Traceability in Software Development Process: An Empirical Approach. In *Rapid System Prototyping, 2008. RSP '08. The 19th IEEE/IFIP International Symposium on*. ISBN 1074-6005, 105–111.

Shahid, M. and Ibrahim, S. (2013). A New Model For Requirements to Code Traceability to Support Code Coverage Analysis. *Asian Academic Research Journal of Multidisciplinary (AARJMD)*. 1(14), 159–172.

Shahid, M. and Ibrahim, S. (2016). Change impact analysis with a software traceability approach to support software maintenance. In *2016 13th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*. January. 391–396. doi: 10.1109/IBCAST.2016.7429908.

Sherba, S. A. (2005). Towards automating traceability: an incremental and scalable approach.

Sherba, S. A., Anderson, K. M. and Faisal, M. (2003). A Framework for Mapping Traceability Relationships. *2 ND INTERNATIONAL WORKSHOP ON TRACEABILITY IN EMERGING FORMS OF SOFTWARE ENGINEERING AT 18TH IEEE INTERNATIONAL CONFERENCE ON AUTOMATED SOFTWARE ENGINEERING*, 32–39.

Skoglund, M. and Runeson, P. (2004). A case study on regression test suite maintenance in system evolution. In *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.* Chicago, IL, USA, 438–442. doi: 10.1109/ICSM.2004.1357831.

Sokolova, M. and Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing & Management*. 45(4), 427–437. ISSN 0306-4573. doi:10.1016/j.ipm.2009.03.002.

Spanoudakis, G., Zisman, A., PÃl'rez-Minana, E. and Krause, P. (2004). Rule-based generation of requirements traceability relations. *Journal of Systems and Software*. 72(2), 105–127.

Spillner, A., Linz, T. and Schaefer, H. (2006). *Software Testing Foundations: A Study Guide for the Certified Tester Exam*. Rocky Nook. ISBN 3898643638.

Tamai, T. and Kamata, M. I. (2009). Impact of requirements quality on project success or failure. In *Design Requirements Engineering: A Ten-Year Perspective*. (pp. 258–275). Springer.

Tassey, G. (2002). *The economic impacts of inadequate infrastructure for software testing*. Technical report.

Venn, J. (1880). I. On the diagrammatic and mechanical representation of propositions and reasonings. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*. 10(59), 1–18.

Wissink, T. and Amaro, C. (2006). Successful Test Automation for Software Maintenance. In *2006 22nd IEEE International Conference on Software Maintenance*. September. Philadelphia, PA, USA, 265–266. doi:10.1109/ICSM.2006.63.

Zimmermann, T., Spanoudakis, G., Perez Minana, E. and Krause, P. (2005). Tracing software requirements artifacts. *Proceedings of the International Conference on SE Research and Practice*.