CASCON 2007 Workshop Report
Traceability in Software Engineering - Past, Present and Future

IBM Technical Report: TR-74-211
October 25, 2007

Workshop Chairs and Presenters:

Juergen Rilling	Concordia University
Philippe Charland	Defence Research and Development Canada - Valcartier
René Witte	University of Karlsruhe

Abstract

Many changes have occurred in software engineering research and practice since 1968, when software engineering as a research domain was established. One of these research areas is traceability, a key aspect of any engineering discipline, enables engineers to understand the relations and dependencies among various artifacts in a system.

It is a well-known fact that even in organizations and projects with mature software development processes, software artifacts created as part of these processes end up being disconnected from each other. This lack of traceability among software artifacts is caused by several factors, including: (1) the fact that these artifacts are written in different languages (natural language vs. programming language); (2) they describe a software system at various abstraction levels (requirements vs. implementation); (3) processes applied within an organization do not enforce maintenance of existing traceability links; and (4) a lack of adequate tool support to create and maintain traceability.

Consequently, one of the major challenges for maintainers while performing a maintenance task is the need to comprehend this multitude of often disconnected artifacts created originally as part of the software development process. From a maintainer's perspective, it therefore becomes essential to establish and maintain the semantic connections among these artifacts.

The research community's focus has been initially on developing new process models and tools to support the development of monolithic applications written in third generation procedural languages. More recently, the focus has switched to developing and maintaining modern distributed systems deployed across a networked infrastructure using a variety of methods that must address the heterogeneous nature of today's Web-based systems.

However, this lack of traceability among software artifacts becomes a major challenge for many software maintenance activities. As a result, during the comprehension of existing software systems, software engineers are required to spend a large amount of effort on synthesizing and integrating information from various sources to establish links among these artifacts. Existing and current research in software traceability focuses on reducing the cost associated with this manual effort by developing automatic assistance in establishing and maintaining traceability links among software artifacts.

This half-day working session was structured into three parts: past, present and future. The first half contained four talks introducing different aspects of traceability, while the second half was devoted to an interactive discussion section involving the workshop's participants.

Workshop Presentations Overview

Presentation 1: A Historical View of Traceability in Software Engineering

Juergen Rilling, *rilling@cse.concordia.ca*

The presentation started with a historical retro perspective of key developments in the area of traceability over the last 30 years. In the late 70s, Pierce introduced a tool to trace requirements as part of a Navy project [1], storing manually entered links among source code and requirements in a database. During the 80s, CASE tools [2] attempted to address the traceability issue by supporting the creation and maintaining of traceability matrices among different software life cycle artifacts. The 90s were characterized by several paradigm shifts: (1) A shift in the software life cycle models from classical, document driven waterfall models to iterative process models; (2) a shift in the programming language paradigm from functional to object-oriented programming; and (3) a technology shift from mainframes to PCs. All of these paradigm

shifts introduced new traceability challenges with respect to linking and maintaining links among the various artifacts created as part of various software life cycle models.

Presentation 2: Traceability in Avionic Software Development

Philippe Charland, *philippe.charland@drdc-rddc.gc.ca*

The word “avionics” is a contraction of the phrase “aviation electronics.” It refers to the electronic and electrical systems on an aircraft such as the communication, navigation, display and control systems. Today, these systems are a complex combination of computers, sensors, actuators, as well as control and display units, which have to be interconnected and integrated within a very compact space. These real-time systems have to maintain high standards of safety and reliability in a high-stress environment. Avionics is the cornerstone of modern aircraft, as more and more vital functions on both military and civil aircraft involve electronic devices. After the cost of the airframe and engines, it is the most expensive item on an aircraft [3].

The rapid increase in the use of software in airborne systems, such as avionics, and equipment used on aircraft and engines resulted in a need for industry-accepted guidance for satisfying airworthiness requirements. DO-178B, “Software Considerations in Airborne Systems and Equipment Certification,” was published to satisfy this need [4].

DO-178B is a software development guideline. It was jointly created by the Radio Technical Commission for Aeronautics (RTCA) and the European Organization for Civil Aviation Equipment (EUROCAE) for safety-critical software used in aircraft. Its purpose is to provide guidelines for the production of software for airborne systems and equipment that performs its intended function with a level of confidence in safety that complies with airworthiness requirements [4]. The Federal Aviation Administration (FAA), European Aviation Safety Agency (EASA), and Transport Canada accept use of DO-178B as a means to certify avionic software.

Traceability is a key aspect of compliance to DO-178B. A large body of traceability and requirements documentation is developed and maintained in parallel with the software development process to prove that [5]: (1) The software completely satisfies all the specified system requirements; (2) Every single code instruction of the software is necessary and serves its intended purpose; and (3) No unintended code exists in the software, and whatever non-essential code that may exist for portability, robustness or similar reasons will not detrimentally impact the software's reliability from a safety perspective.

Traceability evidence enables auditors to verify that the code meets the lowest level requirements, and that those in turn meet higher level requirements, and so on, all the way up to the system level requirements [5]. System level requirements are generally related to avionic functions, such as navigation and flight management, and cover both hardware and software aspects.

The key to meet the DO-178B objectives in a cost-effective manner is to maintain a constant focus on requirements. For example, as test cases are designed and conducted, requirements coverage analysis is carried out to assess that all requirements have been tested. Structural coverage analysis is also performed to determine the extent to which requirements-based tests exercised the code. In this manner, structural coverage is used as a means of assessing overall test completion [3].

Presentation 3: Traceability - Integration of Maintenance Processes and Knowledge Resources

Juergen Rilling, rilling@cse.concordia.ca

In a complex application domain like software maintenance, knowledge needs to be continually integrated from different sources (e.g., source code repositories, documentation, test case results), at different levels of scope (from single variables to complete system architecture), and across different relations (e.g., static, dynamic) [6, 7]. Consequently, one of the major challenges for maintainers while performing a maintenance task is the need to comprehend this multitude of often disconnected artifacts, created originally as part of the software development process [7]. From a maintainer’s perspective, it therefore becomes essential to establish and maintain the semantic connections among these artifacts. The presentation provided a review of current traceability approaches and then focused on the integration and traceability between knowledge, maintenance processes and resources. The problem of knowledge and process integration is not unique to software maintenance. Other application domains, including Internet

search engines (e.g., Google[1]) or online shopping sites (e.g., Amazon[2]) are facing similar challenges. Common to these application domains is that they use different information resources to support users in a given context. For example, during an online shopping session, users will typically find related information on other relevant products, customer reviews, product summaries, etc. Tools and techniques are utilized to enhance the shopping experience and to facilitate the online shopping process. The two major challenges in applying similar approaches to support software evolution are: (1) The lack of formal models to represent and link relevant knowledge resources and process activities; and (2) the resulting lack of a suitable metaphor to model the interaction between users, the model and the relevant resources.
In this presentation, a brief review of current approaches to traceability was given, covering topics from requirements traceability to source code traceability. The presentation further focused on a current research effort that models the interactions between maintainers, maintenance process model, and relevant knowledge resources to support the maintenance process [8]. In this research, a unified formal model is introduced to integrate both process models and knowledge resources using an ontological representation. The resulting ontological representation allows to take advantage of automated reasoning services provided by ontology reasoners to infer implicit relations (links) between the process model and the knowledge resources (e.g., domain knowledge, documents, user expertise, software).

Presentation 4: Text Mining for Traceability Link Recovery

René Witte, witte@ipd.uka.de

Documents written in natural languages constitute a major part of the artifacts produced during the software engineering lifecycle. Examples are software source code documentations, requirements specifications, design description, and user's guides, as well as natural language texts contained in other artifacts, like bug reports, repository check-in messages, or emails. Especially during software maintenance or reverse engineering, semantic information conveyed in these documents can provide important knowledge for the software engineer. An automated analysis of these artifacts requires the application of technologies from natural language processing (NLP) to documents within the software engineering domain to extract the semantic knowledge. Traceability between code and document artifacts can then be established through executing additional linking algorithms.
In this talk, we first introduced technologies for processing unstructured (natural language) data in the software domain, including information retrieval [9] and text mining [10]. Text mining is commonly known as a knowledge discovery process that aims to extract non-trivial information or knowledge from unstructured text. Unlike information retrieval systems, text mining does not simply return documents pertaining to a query, but rather attempts to obtain semantic information from the documents themselves, using techniques from NLP and artificial intelligence. In the software domain for example, a text mining system can be employed to obtain information about individual software entities mentioned in documents, like the system architecture, its components, and their relationships with packages or classes. These mentions, called named entities, can then be exported in a structured format for further (automated) analyses or browsing by a user. We discussed typical steps performed in the analysis of natural language documents, such as tokenization, part-of-speech tagging and noun phrase chunking, and illustrated particular challenges for NLP within the software domain.
A particular focus was placed on automatically establishing traceability links between source code and its corresponding documentation. An approach based on the automatic population of an ontology (in OWL-DL format) was presented, which allows to cross-link software artifacts represented in source code and natural language on a semantic level [10], by using a common, formal representation for both types of artifacts.

Summary of the Interactive Discussion Section

It seems that from its roots of manual link editing in traceability matrices to modern ontology approaches, the basic idea of a more maintainable, easier understandable and therefore, better software product through links between software artifacts has remained the same. The ongoing research shows that the problem is still far from being solved and both maintainers and researchers face more difficulties nowadays.
During the interactive part of the workshop, participants were asked to form groups to identify, discuss and present potential future directions related to software traceability. In what follows, we summarize briefly

[1] www.google.com

[2] www.amazon.com

the outcome of this interactive session by introducing the topics and some of the discussion points raised in the related discussions.

Processes and their Support for Enforcing Traceability. The discussion here focused on making traceability a mandatory part of any software development and maintenance process. The discussion focused on the impact on traceability caused by the paradigm shift in development methods from pure waterfall to agile development models. In agile development, requirements analysis and coding are improved due to iterative strategies and best practices. Nevertheless, agile modeling suffers from two major drawbacks as far as traceability is concerned: (1) The incompleteness of models. Larman [11] states: "Know that all models will be inaccurate, and the final code or design different - sometimes dramatically different - than the model. Only tested code demonstrates the true design; all prior diagrams are incomplete hints, best treated lightly as throw-away explorations." (2) The lack of models. For example, in XP only 10-20 minutes of fast sketching on a white board is common.

Evolution of Links. This group discussion mainly focused on the trustworthiness of links, including the quality evaluation of these links. Another interesting part of the discussion was related to the fact that traceability and the quality of links are only valid at a given time. It is not guaranteed that throughout the evolution of a software system, these links will still be valid or complete and therefore become less and less trustworthy over time. There is a need for traceability link configuration management and validation, to guarantee a certain level of trustworthiness.

Traceability in Model-driven Software Development. One of the major challenges discussed here was the mapping and therefore, traceability, among the different models used during model-driven software development. Furthermore, the group discussed the need for roundtrip engineering support, to ensure the consistency and evolution of traceability links among often very large models.

Security and Privacy: This discussion focused mainly on the following two topics: (1) How to deal with secured components, where no or only limited information is available? What type of traceability can and should be established between these secured components and the remaining system parts? (2) How to restrict traceability based on privacy and security policies in organizations? It was pointed out that these issues add an additional level of complexity to the traceability problem.

Conclusions

Even after 30 years of research in software traceability, it remains a research domain with many open-ended research challenges. Instead of resolving some of these challenges, the advent of new technologies, programming paradigms and process models have even created new challenges. Some of these new challenges are: the ability to link the semantic knowledge found in the various artifacts; the evolution of existing traceability links in a consistent and accurately way; the ability to ensure that traceability becomes an essential part of any software development process. Future work on traceability will have to focus more on these challenges.

Author Biographies

Philippe Charland, Defence Research and Development Canada - Valcartier

Philippe Charland is a Defence Scientist at Defence Research and Development Canada - Valcartier, in the System of Systems Section. He received a master's degree in computer science from Concordia University in 2004. His research interests are software architecture recovery, program comprehension, and software maintenance.

Juergen Rilling, Concordia University, Canada

Juergen Rilling is an Associate Professor at Concordia University in Montreal, Canada, His research focus is in the areas of software evolution, source code analysis and software traceability. He leads the software maintenance and program comprehension research group at Concordia University, and is a program committee member of several international workshops and conferences related to his research area. His current main research interest focuses on modeling and supporting different aspects of software traceability. He also has several years of industry experience as a software engineer. Dr. Rilling holds a Ph.D. from the

Illinois Institute of Technology in Chicago, a M.Sc. in Computer Science from the University of East Anglia, UK, a Diploma in Business Information Management (Dipl.-Inform. (FH)) from the School of Informatics, Hochschule Reutlingen, Germany.

René Witte, University of Karlsruhe, Germany

René Witte is currently a research associate at the Institute for Program Structures and Data Organization (IPD), at University of Karlsruhe, Germany, where he heads the Text Mining group. His work encompasses foundations in computational linguistics, technical aspects of natural language processing systems including ontologies and semantic desktops, as well as the application of text mining in diverse areas such as news analysis, building architecture, biomedical research and discovery, and software engineering. He also has several years of industry experience as an information systems consultant. Dr. Witte holds a Doctorate of Engineering (Dr.-Ing.) and a Diploma in computer science (Dipl.-Inform.) from the Faculty of Informatics, University of Karlsruhe, Germany.

References

[1] Pierce R., "A Requirements Tracing Tool," *Proc. Software Quality Assurance Workshop Functional and Performance Issues*, 1978.

[2] D.Y. Tamanaha, W.C Wenjen, and B.K. Patel, "The Application of CASE in Large Aerospace Projects," *Aerospace Applications Conference*, IEEE, 12-17 Feb. 1989.

[3] C.R. Spitzer, "Avionics: Elements, Software and Functions", *CRC*, 2006.

[4] RTCA DO-178B, "Software Considerations in Airborne Systems and Equipment Certification", *RTCA Inc.*, Washington, D.C., 1992.

[5] ALT Software Inc., "DO-178B and DO-178B A Certification and Safety Critical Testing - ALT Software," Dec. 2007; http://www.altsoftware.com/products/do-178b/safety/.

[6] Egyed, A. and Grünbacher, P." Supporting Software Understanding with Automated Requirements Traceability". International *Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, Vol. 15, Nr. 5, World Scientific Publishing Company, 783-810, 2005

[7] IEEE, Std. 830: Guide to Software Requirements Specification, 1998.

[8] J. Rilling, et al., "A Unified Ontology-Based Process Model for Software Maintenance and Comprehension", *Proc. IEEE/ACM MoDELS'06 Satellite Events*, Genoa, Italy 2006.

[9] G. Antoniol, et al., "Recovering Traceability Links between Code and Documentation", *IEEE Transactions on Software Engineering*, vol. 28, no. 10, Oct. 2002, pp. 970-983.

[10] R. Witte, et al., "Ontological Text Mining of Software Documents", *Proc. 12th International Conference on Applications of Natural Language to Information Systems*, Jun. 27-29, 2007, CNAM, Paris, France, Springer LNCS 4592, pp.168-180.

[11] C. Larman, "Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development", 3rd Edition, Prentice Hall/PTR, 2004.