Software Traceability – A Key to Improve Software Evolution

Alexey Ponomarev[1,a], Hitesh S. Nalamwar[1,b]

[1]National Research Tomsk Polytechnic University, 30 Lenin Avenue, Tomsk, 634050, Russia

[a]aaponomarev@tpu.ru, [b]Hitesh@tpu.ru

Keywords: IT evolution, software traceability, maintenance, software evolution, agile methodologies.

Abstract. Software traceability is an important part in software development that is getting more and more attention nowadays from organizations and researchers. The paper outlines the importance, different methods and techniques of software traceability. It also explains the need of automating traceability, problems and drawbacks of existing traceability tools, the ongoing challenges facing implementation of traceability in software development life cycle, and finally the paper discusses whether software traceability should be mandated as a key to improve software evolution.

Introduction

Software evolution (maintenance) plays a very important role in software development industry since it is responsible for 40% of software development costs and is also one the most challenging, time consuming and frustrating phases as well. The main challenge facing the software maintainers is the ability to locate the part relevant to the required change or bug and other system's part affected by implementing the change also known as software traceability. [1] To maintain the value of a software system it needs to continuously evolve that involves the need for new functionalities, modifying existing functionalities, or fixing existing defects in the system.

After a software project has been released, the developers move into newer projects and therefore when maintenance is required in the system it would be very hard to allocate the same resources to do the maintenance task. As a result the developers responsible for current maintenance tasks lack knowledge about the software system. Since there is a general lack of understanding the system either because the expert developers have left the project/company or no longer remember how the system was implemented; and there also remains the possibility for them to have missing or not up-to-date software documentation.

Impact analysis, risk assessment and change management

Changes to a software system need to be very carefully analyzed as any change to a part of a software system may affect other parts, which causes it to be unstable, and would lead to waste of time, project overrun, inefficient low quality end product and high cost. Therefore, the change should be controlled, managed and analyzed carefully. Cost and impact analysis, dependency analysis and traceability analysis should be performed, the change should be estimated and any risk should be assessed. The change impact and risk analysis should be reviewed and then accepted before being applied. The larger and more complex a software system is the more difficult and costly it is to maintain it. To manage the evolution of a software system suitable mechanisms have to be designed and implemented.

Traceability

"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." [2] Due to the lack of connectivity between the software artifacts, large amounts of effort and time are spent comprehending the software system and linking various artifacts. In order to reduce the time spent on software comprehension, it is essential to have connections and links between the various software artifacts that would enable to perform forward and backward traceability easily and efficiently. The various reasons for the lack of traceability include the lack of tool support to create and maintain traceability, software processes adopted by organizations not enforcing the maintenance of existing traceability links.

Traceability becomes challenging specifically in agile development environments due to the lack of documentation. Nonetheless, it is a very important task and should be part of the agile development process due to the fact that agile development involves continuous iterative changes to the source code. While making these continuous changes, developers need to carefully preserve the original design decision, architecture and structure of the source code. The need for effective tools to support program comprehension in agile particularly is high, where the software written today is next week's legacy code. [3]

Traceability tools and techniques

The traceability tools and techniques that help to achieve manual and/or automated traceability are listed below.

Requirements traceability matrix. Requirements Traceability Matrix (RTM) is the most popular manual traceability technique that is widely used in organizations. RTM is a table that links high level requirements, design, test cases and source code. The purpose of the RTM is to make sure that all the requirements are met and addressed efficiently in all the development phases and to provide traceability between all software development. The drawback is that it is time consuming; maintaining it becomes more challenging for the large-scale system. Since it is done manually it is more error prone.

Zelda tool for agile environments development. Zelda is a tool created specifically for managing traceability in agile development environments. It is an Eclipse plug-in that helps developers create links from user stories and tasks to source code, test cases and various text-based files. Zelda can manage information from user stories internally or retrieve them from an external tool, such as XPlanner. The process for creating links requires little effort, and the links are kept up to date automatically, so that developers can remain focused on coding. When the underlying artifacts are changed, links are updated by extracting information from a Revision Control (RC) system. By analyzing the results from differencing subsequent versions of an artifact, we can automatically determine the correct location of a link. [3]

Information retrieval. Information retrieval method is used based on similarity. The concept of similarity threshold is used to choose threshold value between correct and false values. The fault with this technique is some of correct links are missed and the tool retrieves some incorrect link as correct one. The performance of this technique is measured on recall and precision. The recall is percentage of correct link retrieved & precision means percentage of actually correct link. An optimal threshold value is used to make good balancing between recall and precision. In this tool, software engineer selects source and target artifact, which he/she wants to work for traceability recovery and similarity threshold. The system returns all pairs of source and target artifact that have not been traced and have similarity below or above threshold value. The retrieval performance needs to be improved in Information Retrieval technique and integrated with structured or syntactic techniques as it is only based on text analysis. [4]

Envision. Envision is a visualization technique that helps in understanding software traceability by using tools such as viewing, navigating, searching, focusing, filtering. It helps transform net structured data into tree structure data. The Envision technique also has features like dual visualization, historical navigating and round-trip visualization. [5]

Case study. For case study, Eclipse based prototype for software traceability is built with Envision technique on Automatic Teller Machine (ATM) simulation system. The team of 15 software engineers selected out of which 6 are professional engineers and 9 are computer science graduates. The case study is divided into 4 tasks: change method for cash dispenser and out UML method transaction, identify java method that causes failure and identify to-be modified java method.

After the case study there is encouraging reaction and positive response from Engineers to perform task using Envision from Engineers. The software offers all kinds of traceability in centralized and hyperbolic tree view allow them to have global view of traceability link. Further improvements suggested by Engineers: 1. Tool interface is verbose to understand 2. It needs some pre-defined search pattern. [5]

Related works. Marcus has developed same software traceability prototype for recovering, maintaining and browse traceability links, but it lacks additional functions like historical navigation path. [5]

Approach based on string edit distance. The technique is used to find out traceability links and attributes from different version of Object Oriented software. The activity of compliance checking of two software versions is greatly assisted by automatic checking that assists user to find out unmatched code. [6]

Case study. The case study considers open source software, as source code is easily available. The approach based on string edit has been experimented on 9 releases of LEDA (Library efficient data type) as it evolves considerably from 35 KLOC to 153 KLOC and from 69 to 410 classes from the first to ninth releases. For discussing finding pair difference coloring technique is used. The error rate of this technique is 1 % in normal case and in worse not more than 4 %. In result Visualization, the 3 ideas namely Abstraction, Navigation & Automation are used. [6]

Related works. Murphy, Notkin, and Sulivan designed an approach that are much closer to this technique but differs between the uses of mapping method used in models and lacks partial matching technique between entities. Sefika developed hybrid approach and powerful tool for checking compliance in code but it does not handle approximate matches. [6]

Benefits, challenges and future work of traceability

Benefits. Regarding the semantics of the traceability relations, they can deliver critical information that can be implemented in various software development and maintenance activities. These relations can be implemented to verify whether a system satisfies its requirements or whether these requirements are indeed relevant for system analysis. [9] Also during maintenance and evolution of software system traceability can be used to assess the implications of a given change or changes to a software system and also the effectiveness of these changes. The several different benefits of software traceability relations are listed below:

- System testing: by relating requirements with test models.
- System inception: by supporting inception team to locate alternatives.

- System acceptance: by empowering users to understand and trust the choices made about the design and implementation of the system and it can also make it possible to reuse system components for a new system. [1]

Challenges. Over the past decade, lots of approaches and methods have been implemented toward handling different aspects of issues related to traceability and have led to production of a number of tools to deploy traceability during software maintenance. Despite all the efforts, time and money spent on the mentioned researches, studies indicate that due to the inefficiency of current technology behind the tools and techniques a quiet number of software development companies are still reluctant when it comes to the enforcing of traceability. The main reasons behind this are the lack of sufficient tools and techniques for automatic generation of traceability relations, and traceability relations produced by current tools are usually unclear and do not have a strong semantic meaning, and as the result can not satisfy requirements needed for system analysis. Also existing tools fail to provide support for all of artifacts types produced in the software development life cycle. [1] Another challenge is that traceability still does not provide tangible direct benefits to the software development.

Future work. The current technologies, techniques and tools have limited implementation of software traceability as a part of software development life cycle. These limitations require additional research and investment in Software traceability field in future. Regarding the automatic generation of traceability relation's problem there should be some work done on standardization of vocabularies used in modeling systems and also standardization on the development of ontologies. [1] Researchers are also working on improving techniques as follows:

- In Envision Technique, they are trying to provide more customization on Visualization style and to accept more data sources like JDBC (Java Database Connectivity). [7]
- In Technique based on string edit distance; they are trying to identify split and fused classes between two consecutive versions. [6]

Controversial statements

Does a complete automated traceability relation generator tool solve all the problems? No, because even if we can develop such a tool that can 100% generate all the traceability relations automatically we still need some mechanisms or tools to verify the correctness and completeness of the relations generated by the tools.

Small organizations with smaller projects do not need to apply traceability techniques in their projects. Regardless of the type or size of the project or the organization, any project is apt to evolution and at some point it would be highly effective to have artifacts that are related to each other in a well-organized manner. It is true that it is much easier to maintain and comprehend smaller software systems rather than large complex ones; nonetheless the need for a well traceable environment is a necessity for delivering high quality results.

Automated traceability tools are better than manual techniques like the manual RTM tables. Tools and techniques should be applied depending on the type of the software project and the type of the organization. For smaller organizations it is very hard to afford investing in traceability tools that offer complete integration, therefore it is more efficient for those organizations to use methods like the manual Requirements Traceability Matrix. On the other hand for larger projects, applying manual RTM is not a good approach as it will be ineffective, time consuming and highly error prone, therefore in this case an automated traceability tool is a necessity.

Conclusion: Mandating Traceability to Improve Software Evolution

Although traceability can be time consuming and organizations would not want to invest time, effort and money in it and also most developers do not like having to deal with documents and having to do extra work that they do not think of it as being necessary. It has proved traceability is very effective and important part of a software development process. Software organizations should adopt traceability as part of their processes. It should be taken very seriously since the return on investment is very high as it facilitates software evolution, minimizes its challenges and helps deliver better quality products.

Organizations should always think ahead regarding traceability because any software product is apt for evolution and the need for a well-maintained traceable environment will be highly effective. Traceability mechanisms can be adapted to comply with any type of software process and also according to the nature of the project.

References

[1] George Spanoudakis, Andrea Zisman, Software traceability - a road map, Software Engineering Group, Department of Computing, City University Northampton Square, EC1V 0HB, UK volume 3, 2004(395-428).

[2] Juergen Rilling, Philippe Charland, René Witte, Traceability in software engineering - past, present and future, CASCON Workshop, IBM Technical Report: TR- 74-211, October 25, 2007.

[3] Ratanotayanon, S., Sim, S.E., Gallardo-Valencia, R., Supporting program comprehension in agile with links to user stories, Agile Conference, 2000(26 – 32).

[4] Rocco Oliveto, Traceability management meets information retrieval methods – strengths and limitations, 12th European Conference Software Maintenance and Reengineering 2008(302 – 305).

[5] Xin Zhou, Zhenzhong Huo, Yaowen Huang, Jian Xu, Facilitating software traceability understanding with ENVISION, 32nd Annual IEEE International COMPSAC, 2008(295 – 302).

[6] G. Antoniol, G. Canfora, A. De Lucia, Maintaining traceability during object-oriented software evolution: a case study, Proceedings, IEEE International Conference Software Maintenance ICSM, 1999 (211 – 219).