Traceability between System Model, Project Model and Source Code

Alexander Delater, Barbara Paech

Institute of Computer Science
University of Heidelberg
Im Neuenheimer Feld 326, 69120 Heidelberg, Germany
{delater,paech}@informatik.uni-heidelberg.de

Abstract. Traceability from source code to system model elements like requirements has been extensively researched. Even though existing approaches use various heuristics and methods to compute traceability links automatically, they do not return very satisfying and dependable results. In contrast to these approaches, we not only consider the system model, but also the project model, which is used for planning and organization in software development projects. In this thesis, we plan to create and utilize traceability links between elements from system model, project model and source code. We believe that by using elements from the project model as mediator connectors, links between elements from the system model and source code can be easily created. In this paper, we present the research problems that need to be solved as well as our principal solution ideas to tackle these problems.

Keywords: traceability, system model, project model, source code

1 Introduction

The software development process relies on traceability information captured throughout the evolution of a software product. Traceability supports, amongst others, program comprehension, change management, software maintenance, software reuse and prevention of misunderstandings [10]. Traceability between requirements and source code has been extensively researched in the past years and much progress has been made in this field. Because the manual creation of traceability links between requirements and source code is cumbersome, error-prone, time consuming and complex [22], a major focus in research is on (semi-) automatic approaches. Existing (semi-) automatic approaches use various techniques, e.g. information retrieval, execution traces, static/dynamic analysis, subscription-based or rule-based link maintenance or combinations of them. Even though these approaches use different heuristics and methods to compute traceability links between requirements and source code, they do not return very satisfying and dependable results [22].

In software development projects, two different types of models are used for abstraction: the *system model* and *project model* [16]. Model elements from the

system model describe the system under construction, such as requirements, use cases, components or design documents. Model elements from the project model describe the on-going project, such as work items, the organizational structure, iterations or meetings (we use the term work item instead of task to avoid misunderstandings with the term task used in requirements engineering). These two models have already been integrated within a model called MUSE: Management-based Unified Software Engineering [16]. The MUSE model is implemented in the model-based CASE tool UNICASE [4].

While the MUSE model describes the system to be developed and its project management, it does not provide traceability to the source code. Furthermore, the MUSE model supports the manual creation of traceability links, but it does not support the automatic creation of traceability links.

In this thesis, we want to extend the MUSE model by a new *code model* to support traceability to the source code. We want to study the usage of traceability links between these three models, namely system model, project model and code model. Moreover, we want to present a (semi-) automatic approach for creating traceability links between these three models. These traceability links are expected to support various development activities, such as program comprehension, change management and software maintenance. We want to implement the extended MUSE model and the proposed approach for (semi-) automatic traceability link creation in UNICASE and evaluate it in various case studies.

This paper is structured as follows: Section 2 describes the research problems concerning this thesis. Section 3 presents the proposed solutions and discusses their novelty. Section 4 gives an overview about related work. Section 5 discusses the applied research methods. Our progress concludes the paper in Section 6.

2 Problems

There are various problems that need to be solved in order to link source code with elements from system model and project model.

P1-Representations of Source Code: A problem is to define the representations of source code that make up the elements of the code model.

P2-Capturing & Inferring Traceability Links: The manual creation of traceability links is cumbersome, error-prone, time consuming and complex. Thus, a (semi-) automatic approach for capturing traceability links between the three models is necessary. Support for direct navigation between elements of all three models is also required.

P3-Identifying Relevant Traceability Links: The approach for solving P2 might create a lot of links. Support for the derivation of the most relevant links is necessary.

P4-Supporting Change Impact Analysis: With the different elements from the code model from P1 and approaches for capturing and identifying relevant traceability links from P2 and P3, several development activities can be supported. In this thesis, we want to focus on supporting change impact analysis and present an algorithm using the newly created traceability links.

3 Proposed Solutions

3.1 P1-Representations of Source Code

For the elements of the code model, we want to focus on file-based and change-based representations, because they are widely used in software development projects. For example, file-based representations are file resources containing source code or line(s) of code in these resources. Change-based representations are supported by a version control system (VCS), e.g. patch or revision/branch.

UNICASE is a plugin for the Eclipse integrated development environment (IDE). The Eclipse IDE supports various programming languages through additional plugins, e.g. Java, C++, Python etc. By integrating UNICASE and Eclipse with plugins for VCSs like Subversion [23] or Git [12], we can provide a comprehensive tool environment supporting the developers while they perform various development activities. By using these plugins, we can access file-based as well as change-based representations of source code.

3.2 P2-Capturing & Inferring Traceability Links

Work items represent a unit of work which describe changes to be performed to the code as well as new developments. They are the task descriptions used in many software development projects. As they are the basis of the daily work, they are regularly kept up-to-date [14]. Furthermore, as work items are used to describe pending work, they can also implicitly mention the relationships between system elements relevant to the current work item within its textual description, e.g. the requirement that needs to be implemented or a related design element.

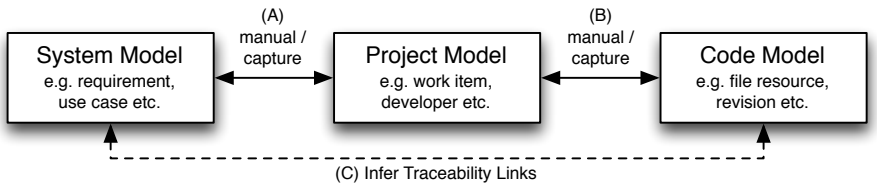

Fig. 1. Traceability between system model, project model and code model

We believe that by using project model elements as mediator connectors, traceability links between system model elements and code model elements can be easily created (see Fig. 1). The core idea of creating traceability links between elements of system-, project- and code model is letting the developers create these links themselves. First, the developer selects a work item and starts implementation. While working on the work item, all system elements (e.g. requirements, design documents) the developer looks at during implementation are automatically captured (see A in Fig. 1). After finishing the implementation of a

work item, the developer does not immediately commit the changes to the VCS. Instead, before the commit, s/he has to verify the list of captured traceability links. This means that the developer has to accept all or reject some traceability links that were captured. It is an open question whether traceability links could be suggested as likely to be relevant. This additional work results in very little overhead for the developers. After this verification, the newly created revision in the VCS is linked to the work item (see B in Fig. 1). It must be studied whether the set of links of one work item can be used efficiently to navigate between the elements linked to that work item, e.g. the requirements and the code related through that work item (see C in Fig. 1). Note that this approach also implicitly alleviates the problem of link maintenance, if it is assumed that any relevant change to a system element is performed only in context of a work item. The links of the most recent work items always provide the most up-to-date links between elements of the system model and code model.

3.3 P3-Identifying Relevant Traceability Links

The approach for capturing and inferring traceability links might create a lot of links. Support for the derivation of the most relevant links is necessary. We plan to implement an algorithm that provides a relevance ranking for each link based on the change history of the elements connected by the link. The change impact analysis can focus on the most relevant traceability links.

3.4 P4-Supporting Change Impact Analysis

We plan to implement an algorithm for change impact analysis using the most relevant captured and inferred traceability links. This algorithm bridges the gap between requirements and source code to answer questions as: What parts of the source code need to be changed based on a change in a requirement? We want to classify our new algorithm using the taxonomy presented by Lehnert [17] and compare it to existing algorithms. We expect that this algorithm is able to provide more detailed results during change management than existing algorithms.

4 Related Work

Maintaining traceability links between source code and other artifacts is a challenging task and therefore a field of intense research.

To the best of our knowledge, no approach uses work items to create traceability links between requirements and code. Either they create links between requirements and code using mostly (semi-) automatic approaches (e.g. information retrieval [1, 19, 20, 13, 6], execution-trace analysis [8, 11, 5], static/dynamic analysis [2], subscription-based or rule-based link maintenance [18] or combinations of them [7]) or only create links between work items and code [3].

Furthermore, other approaches only relate structures in the source code like classes, methods, lines of code or modules, files and resources to other artifacts like requirements [24]. This is also supported by our approach. However, our approach is also able to track exact changes in the source code.

An approach similar to ours for the automatic capturing of links was presented by Omoronyia et al. [21]. They have achieved traceability between use cases and source code. Their approach is based on tracing the operations carried out by a developer called navigation trails. However, this approach requires an elaborate model with rankings of navigation trails to derive the most relevant links. It is an open question whether the availability of work items can alleviate this ranking and how to define rankings for other elements, e.g. design documents touched while implementing a use case. Their approach is also able to identify which developer is involved in the realization of a specific use case. The contribution of Omoronyia et al. shows that tracking changes displays some advantages over the other approaches. For example, relating a developer to the source code and use cases is almost impossible with the other approaches, but very easy if changes/operations are tracked, like in our approach.

Except Omoronyia et al., all other approaches mentioned above try to create traceability links after the implementation of the source code. In comparison, our approach creates traceability links while the system is implemented. We track the changes made to the source code and link them to the work items they belong to. The work items themselves are linked to elements of the system model and new traceability links can be captured between them during development. Based on the intermediate work items, we expect to be able to infer reliable traceability links between system model elements and source code.

5 Research Methods

The overall goal is to validate our proposed solutions. To reach this goal, we apply a tool prototype driven approach where each conceptual research result is developed in parallel with a tool prototype based on the model-based CASE tool UNICASE. Thus, we are able to validate our results early by applying them in academic projects (e.g. bachelor/master theses), in practical courses as well as in the open source project UNICASE itself.

First case studies showed that links between system elements and project elements provide useful information for the work (by shortening the navigation paths of the developers) and that based on such links system elements are kept more up-to-date [15]. We want to conduct more case studies using our presented approach and developed tool support based on UNICASE.

For change impact analysis, traceability links can be evaluated by calculating two metrics: the percentage of actual matches that are found (recall) and the percentage of correct matches as a ratio to the total number of candidate links returned (precision). We want to apply these metrics to our algorithm for change impact analysis and compare the results to existing approaches, e.g. [1, 19, 20, 13, 6]. We want to compare the effort and quality of capturing traceability links

between requirements and source code of our presented approach to the results of other conducted exploratory experiments, e.g. by Egyed et al. [9].

6 Progress

In 2011, defined the representations of source code that we want to focus on (P1). Furthermore, we provided (semi-) automatic support for capturing traceability links between project model and code model. We used patches and revisions in a version control system as two possible types of representation of source code. Changes to the source code are tracked and when the developer commits some code changes, links between the code changes and the work item are captured (P2).

In 2012, we plan to provide (semi-) automatic support for capturing traceability links between system model and project model in UNICASE (P2). We will implement an algorithm that provides a relevance ranking for each link based on the change history of the elements connected by the link (P3). Furthermore, we plan to implement an algorithm for change impact analysis using the most relevant captured and inferred traceability links (P4). We will evaluate the algorithm using data from the open source project UNICASE. We expect to finish this thesis by mid 2013.

References

1. Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., Merlo, E.: Recovering traceability links between code and documentation, IEEE Transactions on Software Engineering, pp. 970-983 (2002)
2. Antoniol, G., Gueheneuc, Y.G.: Feature identification: A novel approach and a case study, In ICSM '05: Proceedings of the 21st IEEE International Conference on Software Maintenance, pp. 357-366 (2005)
3. Anvik, J., Storey, M.A.: Task articulation in software maintenance: Integrating source code annotations with an issue tracking system, In ICSM '08: IEEE International Conference on Software Maintenance, pp. 460-461 (2008)
4. Bruegge, B., Creighton, O., Helming, J., Koegel, M.: Unicase - an Ecosystem for Unified Software, In ICGSE '08: Distributed software development: methods and tools for risk management, ICGSE Workshop 2008 (Bangalore, India, 2008)
5. Burgstaller, B., Egyed, A.: Understanding where requirements are implemented, In 2010 IEEE International Conference on Software Maintenance, pp. 1-5 (2010)
6. De Lucia, A., Fasano, F., Oliveto, R., Tortora, G.: Recovering traceability links in software artifact management systems using information retrieval methods, Transactions on Software Engineering Methodology, vol. 16, no. 4, art. 13, ACM (2007)
7. Eaddy, M., Aho, A.V., Antoniol G., et al.: CERBERUS: Tracing requirements to source code using information retrieval, dynamic analysis, and program analysis, In the 16th IEEE International Conference on Program Comprehension (ICPC), pp. 53-62 (2008)
8. Egyed, A.: A Scenario-Driven Approach to Trace Dependency Analysis, Transactions on Software Engineering, vol. 29, no. 2, pp. 116-132, IEEE (2003)

9. Egyed, A., Graf, F., Grünbacher, P.: Effort and quality of recovering requirements-to-code traces: Two exploratory experiments, In RE '10: Proceedings of the 18th International IEEE Requirements Engineering Conference (RE) (2010)
10. Egyed, A., Grünbacher, P.: Supporting software understanding with automated requirements traceability, International Journal of Software Engineering and Knowledge Engineering, vol. 15, no. 5, pp. 783-810 (2005)
11. Eisenberg, A.D., De Volder, K.: Dynamic feature traces: Finding features in unfamiliar code (2005)
12. Git - Fast Version Control System. http://git-scm.com.
13. Hayes, J.H., Dekhtyar, A., Osborne, J.: Improving requirements tracing via information retrieval, International Conference on Requirements Engineering (2003)
14. Helming, J., Arndt, H., Hodaie, Z., Koegel, M., Narayan, N.: Semi-automatic assignment of work items, In ENASE '10, pp.149-158 (2010)
15. Helming, J., David, J., Koegel, M., Naughton, H.: Integrating system modeling with project management - a case study, In COMPSAC '09: Proceedings of the 2009 33rd Annual IEEE International Computer Software and Applications Conference (Washington, DC, USA, 2009), IEEE Computer Society, pp. 571-578 (2009)
16. Helming, J., Koegel, M., Naughton, H.: Towards traceability from project management to system models, In TEFSE '09: Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering, pp.11-15. IEEE Computer Society (2009)
17. Lehnert, S.: A taxonomy for software change impact analysis, In Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution (New York, NY, USA, 2011), IWPSE-EVOL '11, ACM, pp. 41-50 (2011)
18. Maeder, P., Gotel, O.: Towards Automated Traceability Maintenance, Journal of Systems and Software (2011)
19. Marcus, A., Maletic, J.I.: Recovering documentation-to-source-code traceability links using latent semantic indexing, In Proceedings of the 25th International Conference on Software Engineering, pp. 125-135. IEEE Computer Society (2003)
20. Marcus, A., Maletic, J.I., Sergeyev, A.: Recovery of traceability links between software documentation and source code, International Journal of Software Engineering and Knowledge Engineering, vol. 15, no. 5, pp. 811-836 (2005)
21. Omoronyia, I., Sindre, G., Roper M., Ferguson J., Wood, M.: Use case to source code traceability: The developer navigation viewpoint, In 2009 17th IEEE International Requirements Engineering Conference, pp. 237-242 (2009)
22. Spanoudakis, G., Zisman, A.: Software traceability: A roadmap, In Handbook of Software Engineering and Knowledge Engineering, World Scientific Publishing, pp. 395-428 (2004)
23. Apache Subversion. http://subversion.apache.org.
24. Treude, C., Storey, M.A.: How tagging helps bridge the gap between social and technical aspects in software development, In Proceedings of the 31st International Conference on Software Engineering (ICSE), pp. 12-22 (2009)