Quality Improvements for Trace Links between Source Code and Requirements

Paul Hübner

Institute for Computer Science, Heidelberg University,
Im Neuenheimer Feld 326, 69120 Heidelberg, Germany
huebner@informatik.uni-heidelberg.de

Abstract. [**Context and Motivation**] Traceability between source code and requirement artifacts is important for various tasks during software development. However, it is a lot of effort to create and maintain traceability links manually. Therefore, semi-automatic traceability support is developed. [**Question/ problem**] Traceability research has a strong focus on trace link recovery using information retrieval (IR) techniques. These techniques use the textual similarity of documents to create trace links. The quality according to precision and recall of these techniques is still not satisfying. [**Principal ideas/ results**] Precision and recall can be improved by providing more data as used in IR. In this thesis, we evaluate two new data source types to create trace links. To link two specific artifacts, we exploit existing links in the context of these artifacts. Furthermore, we use the interaction logs of developers for trace link creation between the artifacts the developers touched. [**Contribution**] In this paper, we present the research problems as well as the principal solutions to deal with these challenges, our research methodology, and our progress so far.

Keywords: traceability, quality, interaction, requirement, source code

1 Introduction

Traceability between source code and requirements is important for many software engineering tasks e.g., maintenance, program comparison, verification [10, 11, 16, 5, 1] and is a major concern of software engineering research [7, 2, 3, 15]. IR-based trace link recovery is the most common used technique to discover trace links between artifacts [2]. This technique creates links between artifacts with textual similarity as implemented in the used IR model. Thus, IR-based trace link creation is limited with respect to similarity measure and the used IR model [12]. Also precision and recall are not satisfying [13].

To improve the semi-automatic creation of trace links between implementation and requirements artifacts, we investigate different possibilities to create trace links during software development. We build on the approach of Delater et al. [6] which uses the actual working task of developers as an intermediate element to link the requirements and the code involved in this task. Our main

idea is to evaluate two new data sources to create trace links. To link two specific artifacts, we exploit the existing links in the context of these artifacts. Furthermore, we use the interaction logs of developers for trace link creation between the artifacts the developers touched. Therefore, we implement algorithms for these new data sources and compare them with traditional IR methods, based only on the artifacts, as well as with the working task approach of Delater. We also analyze combinations of these methods to improve precision and recall for the recovered trace links. In which precision is the fraction of retrieved trace links that are relevant, while recall is the fraction of relevant trace links that are retrieved.

The remainder of this paper is structured as follows. Section 2 describes the research problem tackled in this thesis. Section 3 presents the proposed solutions and discusses their novelty. Section 4 gives an overview of related work. Section 5 discusses the applied research methods. With our progress, we conclude the paper in Section 6.

2 Problem

The problem of established trace link recovery techniques is their insufficient quality regarding precision and recall [3]. According to Gotel et al. [7] there is a tradeoff between precision and recall: if IR-based methods have a good recall (up to 90%), the generated candidate links include a lot of false positives, i.e. precision is between 10 to 20% [7]. Even with recall values of 90%, important links might be missing. Therefore, we investigate the following general research question: **(RQ) How to improve semi-automatically detected trace link quality according to precision and recall?** One reason for this problem is that textual similarity is not sufficient to detect links. E.g., a use case is relevant for the class implementing system functions required in the use case, but the class text may use different words than the use case. Another reason for this problem is the dependency between recall and precision when using IR-based recovery techniques. To get high recall values, the threshold defining that two artifacts are linked has to be low [13]. This often results in a bad precision, as many unrelated artifacts are included. Thus, we propose to use interaction logs and existing links as additional data sources. These additional data sources can improve recall, since they are likely to yield new links compared with IR. Precision will be improved, since false positive links are more unlikely for interaction logs and existing links as for IR. We investigate the following detailed research questions: (RQ1) Do the additional data sources improve precision and recall? (RQ2) Can the combination improve the precision and recall even more? (RQ 2.1) How can the best combination of trace link recovery techniques be determined? (RQ 2.2) Is there a certain combination of trace link recovery techniques which leads to most improvements in precision and recall in different settings?

3 Proposed Solution

To answer RQ1, our proposed solution is based on the usage of two new data sources for trace link creation. We build on the approach of Delater et al. [6], where trace links between requirements and code are captured semi-automatically by using tasks as intermediate elements. Trace links are created along with the processing of the task and the associated requirement and code elements based on artefact changes. Figure 1 shows our extension of Delater's approach. It can be separated into two parts.

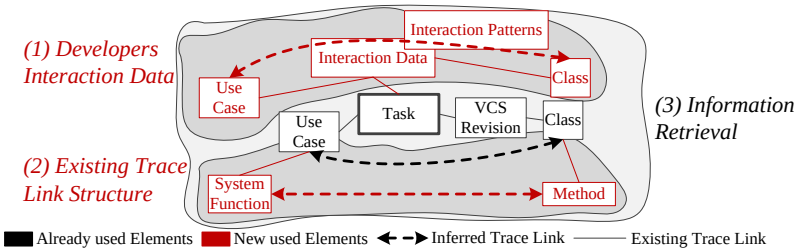

Fig. 1. Already used and new Trace Link Data Source Types

On the one hand, we use developer's interaction data produced while working with an integrated development environment (IDE) (cf. Figure 1 *(1)*). Interaction data consists of *interaction events*. These interaction events have different types e.g., edit, selection, navigation, execution, etc. and are typically related to artifacts like source code or requirements. A set of certain interaction events like *edit* $\rightarrow$ *navigation* $\rightarrow$ *edit* is a *navigation path*. Frequently occurring interaction paths are called *interaction patterns*. The knowledge of interaction patterns will improve precision as frequent navigation indicates a stronger relation between the artifacts.

On the other hand, we exploit the existing trace link structure (cf. Figure 1*(2)*). to improve the trace link precision, since trace links are created based on already verified trace links. When using data created during runtime for trace link recovery, it is natural to also apply this recovery repeatedly during development and not only after development as it is typical in traditional methods [4]. This extends trace link recovery to trace link recommendation to developers directly during development. It makes trace links available early in a project. Furthermore, the assessment of the trace links by developers is facilitated, since link recommendations fit to the actual context of a developer. To investigate RQ2, we compare different combinations of trace link recovery techniques regarding precision and recall. The starting point is the combined usage of IR and interaction data-based recovery to increase trace link recall. To increase precision we combine this with the exploitation of the existing trace link structure or interaction patterns. Both can also be used to focus IR-based recovery on specific artifacts.

In the following we illustrate our ideas by an example. We assume that a developer knows the requirement s/he has to implement. E.g., the developer implements a system function as a set of methods within a class.

First, we outline how to use developers' interaction data to infer trace links. During the implementation of the methods s/he looks up the detailed specification of the system function in the requirements specification. This navigation between system function definition and implemented methods is logged. After s/he has finished the implementation of the methods, trace links between system function specification and methods can be inferred based on the used navigation paths. Using the working task context as in Delater's work [6] is an additional benefit, since navigation interactions can be separated based on their association to a task (cf. Figure 1).

Second, we outline how to exploit the existing trace links structure of implementation and requirements. Both, requirements and code are typically hierarchical. E.g., a use case includes a system function and a class includes a method. Trace links between upper level artifacts can be used to infer trace links between lower level artifacts and vice versa. Assume that the class in which the methods for the system functions should be implemented is already linked with a use case. In the requirements specification the use case is linked with system functions. Thus, the existence of links between the use case and the system functions as well as the use case and the class increases the likelihood that some methods should be linked to the system functions. This can be applied as a filter on previously identified navigation paths or can be used to add additional link candidates. In both cases, precision can be improved by using IR to check the textual similarity between implemented methods and the systems functions specification.

4 Related Work

A recent overview of important IR-based trace link recovery approaches in the last decade and an evaluation of the approaches is given by Borg et al. [2]. We use the evaluation results of the discussed IR approaches as comparison measure for the quality assessment of our approach. Other approaches using additional data sources exist. The following two approaches also exploit the artifacts hierarchy, but not for trace link recovery. De Lucia et al. [5] presented an approach for a fine-grained management of trace links using a software artifacts hierarchy in addition to traditional IR-based recovery methods. In contrast to our approach, the hierarchy relations were not directly used when recovering trace links, instead they were used to navigate between linked artefacts. Wentao et al. [17] use existing trace links for further data processing actions. They compare the evolution of code, requirements, and trace links and use the trace links as a starting point to compensate for the divergence between those elements.

The use of interaction data of developers is widespread in software engineering, especially in the domain of recommendation systems. Maalej et al. [9] describe how to use interaction data, based on examples of existing recommendation systems, to trigger recommendations about used source artifacts. Furthermore, they describe different methods to aggregate this data to sessions, tasks, and activities and how to filter such data for productive use. We build on these methods for the development of our interaction patterns. Cleland-Huang et al.

[4] present a study in which they implemented and evaluated an approach to recommend trace links between system model (UML) and requirements during model changes. They use edit interactions to trigger recommendations of trace links which is similar to our approach, but use IR instead of the interaction data to create links. The use of interaction data to find software artifact relations is also applied in the domain of software architecture. Konopka et al. [8] find relations between source code artifacts and tasks based on IDE interaction data. This is used for grouping code, but not for linking requirements and code as in our approach.

There are further ways to improve the precision of links. Niu and Mahmoud [14] presented a machine learning-based clustering approach to reduce the false positives in a trace link candidate list. This has a different focus and could be applied in addition to our approach.

5 Research Method

The research method used in this thesis is based on an adapted version of Wieringa's Design Science Methodology [18] in which we combine several small studies. It consists of the four phases: (1) problem identification, (2) problem establishment, (3) solution design, and (4) solution validation. In the following we introduce these four phases for our research.

The identified problem (1) is that the quality of IR-based trace link recovery is not sufficient for software engineering tasks. We establish this problem (2) by the analysis of existing studies and related work for trace link recovery and usage. We conduct a literature review to get an overview of existing trace link recovery approaches, to find possibilities to improve precision and recall of trace links, and to discover options to embed trace link creation in the software development process. Thus, the research questions we answer with our literature review are: What are existing trace link recovery approaches, which data sources and which recovery techniques are used? What are the reasons for insufficient precision and recall of existing approaches? Which options for improvements are discussed and how are they applied? Which kind of experimental setup and which kind of data sets are used to evaluate trace link recovery approaches?

Our solution design (3) is described in Section 3. To validate the solution (4), we perform studies with two open source projects. The **Existing Link Structure Study** will use the iTrust project data[1] which consist of a use case-based requirement specification, Java and JSP-based source code, existing trace link definitions, and a project history with multiple versions of the previously mentioned resources available in a version control system (VCS). A history version of the trace link specification can be used to check the recall and precision of our existing link structure algorithm. The **Developers' Interaction Data Study** will use Mylyn project data[2] which consists of a Bugzilla project consisting of

[1] `http://agile.csc.ncsu.edu/iTrust`, last checked February 25, 2016
[2] `http://www.eclipse.org/mylyn/developers/`, last checked February 25, 2016

tasks (bugs, issues, etc.) including attached interaction data. The task descriptions are used to specify requirements (as features). Due to the attached Mylyn interaction data, it is possible to see the interaction associated with a specific requirement and the affected source code elements.

To compare our results with established IR-based trace link recovery approaches we will apply a set of such IR-based recovery approaches to both projects. Moreover, for the developer's interaction data study we also apply Delater's working task-based recovery approach [6]. This is possible since in the used project data development tasks are explicitly specified.

We are still looking for project data which give us the possibility to apply both of our new recovery techniques in combination to completely investigate RQ2. Another way to get data to evaluate all our recovery techniques in combination are practical student courses. This requires the integration of our algorithms in the eclipse[3] IDE and their usage to recommend trace links during development.

6 Progress

We perform the literature review in parallel to the other task of this thesis. We plan to finalize the research questions and the execution of the search process by mid of 2016, followed by the result evaluation and documentation which we plan to finish by the end of 2016.

We have already implemented the basic algorithms for the interaction path detection and the use of existing trace link structure. We already performed a statistical analysis on the Mylyn interaction data and extracted basic interaction patterns which we use in the current implementation of our algorithm. For using existing trace link structures we transferred the iTrust data into a local data base implementing a schema for the structured, hierarchical specification of requirements and source code elements as this is necessary for using trace links between those elements.

We plan to finish the existing link structure analysis with iTrust data in June 2016 and the developers' interaction data study with Mylyn project data in September 2016, including the comparison to only using IR-based trace link recovery. Until then, we also hope to find project data which allows the application and evaluation of both interaction data and existing trace link structure based trace link creation. We plan to finish the research for this thesis by the end of 2017.

If time permits, we will integrate these algorithms in the eclipse IDE and use them to recommend trace links during development. However, for the evaluation of trace link recommendation a further study with developers using the tool in a real project or at least a student course will be necessary. This is difficult to achieve and therefore not planned as an integral part of the thesis.

Acknowledgment I thank my advisor Barbara Paech for her excellent support.

[3] `http://www.eclipse.org`, last checked February 25, 2016

References

1. Bavota, G., Colangelo, L., De Lucia, A., Fusco, S., Oliveto, R., Panichella, A.: TraceME: Traceability Management in Eclipse. In: ICSM 2012. pp. 642–645. IEEE (Sep 2012)
2. Borg, M., Runeson, P., Ardö, A.: Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. Empirical Software Engineering 19(6), 1–52 (May 2013)
3. Cleland-Huang, J., Gotel, O.C.Z., Huffman Hayes, J., Mäder, P., Zisman, A.: Software traceability: trends and future directions. In: FOSE 2014. pp. 55–69. ACM Press, New York, New York, USA (May 2014)
4. Cleland-Huang, J., Mäder, P., Mirakhorli, M., Amornborvornwong, S.: Breaking the big-bang practice of traceability: Pushing timely trace recommendations to project stakeholders. In: RE 2012. pp. 231–240. IEEE (Sep 2012)
5. De Lucia, A., Fasano, F., Oliveto, R., Tortora, G.: Fine-grained management of software artefacts: the ADAMS system. Software: Practice and Experience 40(11), 1007–1034 (Oct 2010)
6. Delater, A., Paech, B.: Tracing Requirements and Source Code during Software Development: An Empirical Study. In: ESEM 2013. pp. 25–34. IEEE, Baltimore, MD, USA (Oct 2013)
7. Gotel, O., Cleland-Huang, J., Hayes, J.H., Zisman, A., Egyed, A., Grunbacher, P., Antoniol, G.: The quest for Ubiquity: A roadmap for software and systems traceability research. In: RE 2012. pp. 71–80. IEEE (Sep 2012)
8. Konopka, M., Navrat, P., Bielikova, M.: Poster: Discovering Code Dependencies by Harnessing Developer's Activity. In: ICSE 2015. pp. 801–802. IEEE (May 2015)
9. Maalej, W., Fritz, T., Robbes, R.: Collecting and Processing Interaction Data for Recommendation Systems. In: Recommendation Systems in Software Engineering, pp. 173–197. Springer Berlin, Heidelberg (2014)
10. Mäder, P., Egyed, A.: Do software engineers benefit from source code navigation with traceability? An experiment in software change management. In: ASE 2011. pp. 444–447. IEEE (Nov 2011)
11. Mäder, P., Egyed, A.: Do developers benefit from requirements traceability when evolving and maintaining a software system? Empirical Software Engineering 20(2), 413–441 (Apr 2015)
12. McMillan, C., Poshyvanyk, D., Revelle, M.: Combining textual and structural analysis of software artifacts for traceability link recovery. In: TEFSE 2009. pp. 41–48. IEEE (2009)
13. Niu, N., Bhowmik, T., Liu, H., Niu, Z.: Traceability-enabled refactoring for managing just-in-time requirements. In: RE 2014. pp. 133–142. IEEE (Aug 2014)
14. Niu, N., Mahmoud, A.: Enhancing candidate link generation for requirements tracing: The cluster hypothesis revisited. In: RE 2012. pp. 81–90. IEEE (Sep 2012)
15. Panichella, A., De Lucia, A., Zaidman, A.: Adaptive User Feedback for IR-Based Traceability Recovery. In: SST 2015. pp. 15–21. IEEE (May 2015)
16. Rempel, P., Mäder, P., Kuschke, T., Cleland-Huang, J.: Mind the gap: assessing the conformance of software traceability to relevant guidelines. In: ICSE 2014. pp. 943–954. ACM Press, New York, New York, USA (May 2014)
17. Wentao Wang, Gupta, A., Yingbo Wu: Continuously delivered? Periodically updated? Never changed? Studying an open source project's releases of code, requirements, and trace matrix. In: JITRE 2015. pp. 13–16 (2015)
18. Wieringa, R.J.: Design Science Methodology for Information Systems and Software Engineering. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)