Analyzing the Tracing of Requirements and Source Code during Software Development: A Research Preview

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. [**Context and motivation**] Traceability links between requirements and code are often created after development, which can, for example, lead to higher development effort. To address this weakness, we developed in previous work an approach that captures traceability links between requirements and code as the development progresses by using artifacts from project management called work items. [**Question/problem**] It is important to investigate empirically what is the best way to capture such links and how these links are used during development. [**Principal ideas/results**] In order to link requirements, work items and code during development, we extended our approach from previous work by defining three traceability link creation processes. We are applying these processes in practice in a software development project conducted with undergraduate students. The results indicate that our approach creates correct traceability links between requirements and code with high precision/recall during development, while developers mainly used the third process to link work items after implementation. Furthermore, the students used a subset of the created traceability links for navigating between requirements and code during the early phase of the development project. [**Contribution**] In this paper, we report on preliminary empirical results from applying our approach in practice.

Keywords: trace, requirement, work item, code, software development

1 Introduction

Requirements-to-code traceability reflects the knowledge where requirements are implemented in the code. The capture of such links during development is the focus of recent research [6]. Asuncion & Taylor [1] presented an approach for capturing links between heterogenous artifacts, including requirements and code, by analyzing interactions of users while they create/generate or modify artifacts. Omoronyia et al. [8] capture links between requirements and code based on the operations carried out by developers creating code artifacts to realize requirements. In contrast to these approaches, we present an approach that captures traceability links between requirements and code using artifacts from project management called work items [3]. Practitioners have discussed the practice of

using work items to capture links between requirements and code, but to the best of our knowledge there has been no systematic study of this practice [2].

However, surprising little is known about the quality (precision/recall) and usage of requirements-to-code links during software development [6]. Asuncion & Taylor and Omoronyia et al. did not provide empirical work in which they showed the feasibility of their approaches in practice. Maeder & Egyed [5, 6] report on the usage of such links during software maintenance. However, the authors did not investigate how these links are used during development, as software maintenance happens after development.

In this paper, we present preliminary research results from applying our approach [3] in practice. In order to do this, we extended our approach by defining three traceability link creation processes linking requirements, work items and code. We are applying these processes in practice in a software development project conducted with undergraduate students. Based on the gathered data, we are inferring direct traceability links between requirements and code using work items. We are investigating the precision/recall of the inferred traceability links between requirements and code as well as how often each process was executed by the students. Furthermore, we are analyzing whether the students actually use these direct links during the early phase of the development project for navigating between requirements and code.

The remainder of this paper is structured as follows: Section 2 presents the approach. Section 3 introduces the project context, while Section 4 reports on our research questions and preliminary results. Section 5 discusses threats to validity and Section 6 concludes the paper and discusses future work.

2 Approach

In [3], we defined a Traceability Information Model (TIM) consisting of artifacts from requirements engineering (features, functional requirements), project management (work items, sprints, developers) and code (code files, revisions) as well as the traceability links in between. A feature is realized in a sprint and is detailed in one or more functional requirements. Work items describe work to be done to realize functional requirements, are assigned to developers, have a completion status and a due date. A work item must have one or more linked functional requirements and is contained in a sprint. A feature can be related to a work item, e.g. during bug fixing. One work item can create one or more revisions. A revision contains one or more changed code files and is stored in a version control system (VCS).

We presume the following situation in a development project. First, a list of features and functional requirements exists. Second, a project manager has planned the implementation of the features in sprints and s/he has broken down the implementation schedule of the functional requirements into work items for the developers. Third, all work items are already assigned to developers. Below we use the term *requirement* to refer commonly to features and functional requirements.

Our approach uses work items to link requirements and code during development. As we presume that the implementation of the requirements is planned in work items, we need to capture links between the work item and the code that is created by its assigned developer. We identified three possibilities of developers to select a work item that is related to their implemented code. Developers can select a work item *before* they start the implementation of code (Process A), *during* implementation when they have created code but have not yet stored it in a VCS (Process B), or *after* implementation when they have created code that is already stored in a VCS (Process C). All three processes are depicted in Figure 1 and explained in the following.

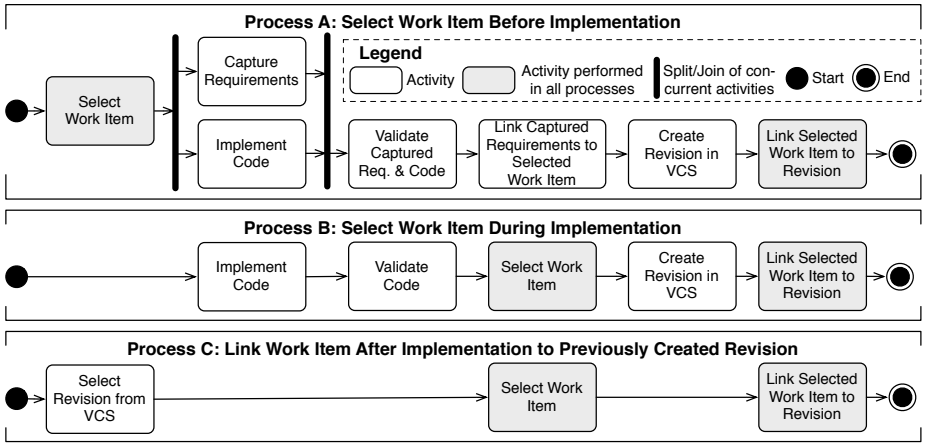

Fig. 1. Traceability Link Creation Processes A, B and C

Process A) Select Work Item Before Implementation: First, the developer selects a work item from his/her list of assigned work items. While working on the work item and implementing new code or changing existing code, all requirements the developer looks at during implementation are automatically captured. For example, s/he may look at requirements to know what to implement. When finishing the implementation of the work item, the developer is asked to validate all captured requirements and new/changed code, which means s/he confirms all related and removes all non-related requirements or code files. The validated requirements are linked to the work item and the validated code is stored in a new revision in the VCS, which is also linked to the work item.

Process B) Select Work Item During Implementation: In contrast to Process A, in Process B a developer does not need to select a work item before implementation. Instead, s/he starts directly with implementation. After the implementation of code and before creating a new revision stored in the VCS, the developer validates the new/changed code files and selects a work item from his/her list of assigned work items. A new revision with the validated code files is stored in the VCS and is automatically linked to the selected work item. In this process, no requirements are captured and need to be validated.

It is important to note that Processes A and B do not force developers to select a work item related to the current implementation. In case the developer implemented code that s/he does not want to be linked to a work item, s/he can omit the linking of a work item, which ends Processes A and B.

Process C) Link Work Item After Implementation to Previously Created Revision: In contrast to Processes A and B, Process C occurs after implementation and it represents an alternative way for the developer to link code to a work item. A VCS stores a history of all previously created revisions with information by whom and when each revision was created, as well as all changed code files. In case a developer has implemented code without selecting a work item before implementation (see Process A) or without selecting a work item during implementation (see Process B), s/he can manually select to link a previously created revision to a work item from his/her assigned work items list. Similar to Process B, no requirements are captured and validated.

A practitioner can perform a mixture of all three processes during the course of the project. However, one of the processes can only be applied once per revision. This means each revision in the VCS is either created (Process A,B) or linked (Process C) by only one of the three processes.

Inferring Traceability Links Between Requirements and Code: The created traceability links of Processes A, B and C are used to infer direct links between requirements and code based on the corresponding work items. In [3], we presented an algorithm for inferring links that is executed when the developer changes the completion status of a work item from *assigned* to *done*. The algorithm connects in a brute force manner all linked requirements of a work item with all the code files in the linked revisions of a work item. An in-depth description of the algorithm can be found in [3].

UNICASE Trace Client: We implemented our approach in the tool UNICASE Trace Client (UTC) [9]. It is an extension to the model-based CASE tool UNICASE [10], which is a plug-in for Eclipse developed in an open-source project. UTC integrates itself seamlessly in Eclipse and its supporting plug-ins, e.g. Subversion (a commonly used VCS). UTC implements the TIM and all its artifacts and traceability links as well as all three link creation processes.

3 Project Context

To evaluate our approach, we conduct a development project with undergraduate students. In the following, we describe the development project and provide information about the participants and the used development process.

Project Description: We are working together with a company from industry specialized on mobile business applications. The company integrates existing business applications into mobile applications for smartphones and tablet computers. For the company, a knowledge database is developed containing user-generated content as well as content retrieved from various Internet data sources (e.g., Google Maps, Wikipedia). The people of the company have a great interest

in full traceability between requirements and code, because they want to maintain the developed application later on. Java and JavaScript were used as main programming languages. The entire project will last for five months from October 2012 until February 2013. In this paper, we report on preliminary results from the first phase (October 1st - November 8th, 2012) divided in two sprints.

Participants & Development Process: We recruited six undergraduate students for our development team, all having basic knowledge in software engineering. The team is applying agile software development techniques, e.g., they hold regular stand-up meetings discussing completed work, planned work and any problems preventing them to continue work. The development process is as follows: in the beginning, the team elicits and specifies a first draft of the requirements together with the company. In each sprint, the team details the requirements and breaks them down into work items describing their realization. They assign each work item to a developer and include it in a sprint. Thus, the situation we presume is present in the project (see Section 2). In the current state of the project, the team specified 3 features, 8 functional requirements and implemented 32 code files with 1.573 lines of code in 81 revisions.

4 Research Questions & Preliminary Results

Based on the gathered data in the project consisting of requirements, work items and code files stored as revisions in VCS, we are applying different analyses. Our research is driven by three research questions (RQ) and the preliminary results are presented in the following.

RQ1. What is the precision and recall of the inferred links? Precision and recall are two standard metrics used in information retrieval [4]. Precision is the fraction of retrieved instances that are relevant, while recall is the fraction of relevant instances that are retrieved. In our case, 'relevant' refers to a *correct traceability link*, which is as a link between a requirement and its code where the code is necessary to realize the requirement. The metrics are computed as:

$$P = \frac{RelevantLinks \cap RetrievedLinks}{RetrievedLinks} \quad (1) \qquad R = \frac{RelevantLinks \cap RetrievedLinks}{RelevantLinks} \quad (2)$$

We manually identified all correct traceability links based on the requirements and code. A total of 42 traceability links between requirements and code were created, while 37 were correct, 5 were wrong and 8 correct links were missing. Wrong traceability links are created when developers change and link code files that are not particularly related to a work item. Missing correct links are created where work items have been linked to the wrong requirement. Both situations are potential causes of errors in our approach. Our approach achieved precision of 0.881 and recall of 0.933. Results of this scale mean that our approach delivers high quality links, which is comparable to manually performed linkage [7].

RQ2. What traceability link creation process do developers use? This RQ is focusing on how often each process was used to link requirements, work

items and code in the first place. The team executed all three processes a total of 81 times (Process A = 5, Process B = 23, Process C = 53). Thus, in 65% of the cases, the team implemented the code first and then linked it to work items (Process C). However, the students behaved differently during the development project. While some students mainly used Process B, others mostly used Process C. Since the requirements are not final and detailed in each sprint, the students only used Process A a few times to look at requirements during development.

RQ3. Do developers use the created links to navigate between requirements and code in the early phases of software development? Each inferred link between a requirement and a code file had a boolean attribute *used* (default value = false). When a developer clicked on the link and "used" it for navigation, the value was set to true. Although the project is still in an early phase, developers already used 9 links of the 42 created links for navigation between requirements and code, or vice versa. We think that the usage will increase in the later phases of the project.

5 Threats to Validity

From the very beginning of the project, we are striving to avoid external and internal threats to validity.

External Validity: In the development project, all undergraduate students had basic knowledge in software engineering. However, no undergraduate student had industrial experience. This does not allow us to draw conclusions for more experienced developers. In addition, Java and JavaScript were used as programming languages. Even though we do not expect this, effects might be different for other programming languages. The authors of this paper gave advice to the students during the project and made sure that they used UTC. The students might behaved differently if they did not have to use UTC.

Internal Validity: To decrease variability in knowledge across students regarding the tracing of requirements and code in UTC, we provided an introductory tutorial of UTC [9]. This ensured that all students knew how to use UTC.

6 Conclusion & Future Work

In this paper, we presented three traceability link creation processes linking requirements, work items and code. We presented preliminary results from applying our approach in practice in a development project conducted with undergraduate students. A finding of our results is that developers mainly link work items after implementation to previously created revisions. Our approach creates correct traceability links with high quality during development. In the current early project state, developers already used 9 of 42 created links to navigate between requirements and code.

Once the state of the project has further progressed, we want to deepen our analyses for the research questions in future work. Regarding RQ1, we will study the evolution of precision and recall of the traceability links, whether they increase or decrease over time. Furthermore, we want to apply existing approaches for automatically linking requirements and code and compare the results with respect to precision and recall to the links created by our approach in the project. To support comparison between the approaches and their results, we will consider the metrics F_2-Measure and MAP (Mean Average Precision), combining precision and recall into a single score. For RQ2, we will investigate how the usage of the three traceability link creation processes changes over the course of the project. With regard to RQ3, we will investigate whether developers use the traceability links in later phases of the project for navigation between requirements and code. Additionally, we want to perform more analyses on the gathered data, e.g., how many work items were assigned to each developer, or the minimum/maximum and average amount of lines of code traced per process.

Acknowledgment. The authors would like to thank the company for providing the opportunity for this project, the students for their participation as well as Ulrike Abelein and Florian Flatow for their help in organizing the project.

References

1. Asuncion, H. and Taylor, R.: Automated techniques for capturing custom traceability links across heterogeneous artifacts. In Software and Systems Traceability, Cleland-Huang, J., Gotel, O., and Zisman, A. (Eds.), Springer, pp. 129-146 (2012)
2. Cleland-Huang, J.: Traceability in agile projects. In Software and Systems Traceability, Cleland-Huang, J., Gotel, O., and Zisman, A. (Eds.), Springer, pp. 265-275 (2012)
3. Delater, A., Narayan, N., and Paech, B.: Tracing Requirements and Source Code during Software Development. In ICSEA'12: Proceedings of the 7th International Conference on Software Engineering Advances, pp. 274-282 (2012)
4. Frakes, W.B. and Baeze-Yates, R. (Eds.): Information Retrieval: Data Structures and Algorithms. Prentice-Hall (1992)
5. Maeder, P. and Egyed, A.: Do software engineers benefit from source code navigation with traceability? - An experiment in software change management. In ASE'11: Proceedings of the 26th International Conference on Automated Software Engineering, pp. 444-447 (2011)
6. Maeder, P. and Egyed, A.: Assessing the effect of requirements traceability for software maintenance. In ICSM'12: Proceedings of the 28th International Conference on Software Maintenance, pp. 171-180 (2012)
7. Maeder, P. and Gotel, O.: Ready-to-use Traceability on Evolving Projects. In Software and Systems Traceability, Cleland-Huang, J., Gotel, O., and Zisman, A. (Eds.), Springer, pp. 173-194 (2012)
8. Omoronyia, I., Sindre, G., Roper, M., Ferguson, J., and Wood, M.: Use case to source code traceability: The developer navigation viewpoint. In RE'09: Proceedings of the 17th International Requirements Engineering Conference, pp. 237-242 (2009)
9. UNICASE Trace Client, http://code.google.com/p/unicase/wiki/TraceClient
10. UNICASE, http://www.unicase.org/