UNICASE Trace Client: A CASE Tool Integrating Requirements Engineering, Project Management and Code Implementation

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: Artifacts for requirements engineering, project management and code implementation are usually stored in separate tools, which makes traceability between these artifacts difficult. We developed the tool UNICASE Trace Client, which stores the aforementioned artifacts in a single environment with full traceability between all artifacts. In this paper, we describe the three traceability link creation process supported by our tool as well as its advanced features for traceability link usage.

1 Introduction

Requirements-to-code traceability reflects the knowledge where requirements are implemented in the code. The simple creation of such links is very important for a development project, as the manual creation can, for example, lead to higher development effort. In previous work [DNP12], we presented an approach that captures traceability links between requirements and code as the development progresses by using artifacts from project management called work items. Based on this approach, we developed the tool UNICASE Trace Client [UTC]. It is an extension to the model-based CASE tool UNICASE [UNI], which is an Eclipse plug-in developed in an open-source project. UTC integrates itself seamlessly in Eclipse and supporting plug-ins, e.g. Subversion, and supports three processes for traceability link creation between requirements, work items and code. Based on these links, it supports various features for traceability link usage.

The remainder of this paper is structured as follows: Section 2 describes the three traceability link creation processes and features supported by UTC. Section 3 concludes the paper and discusses future work.

2 UNICASE Trace Client

UTC incorporates artifacts from *requirements engineering* (features, functional requirements), *project management* (work items, sprints, developers) and *code* (code files, revi-

sions). 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, they 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).

For using UTC, we presume the following situation in a development project and the used development process. 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.

2.1 Traceability Link Creation Processes

UTC uses work items to link requirements and code during development. As we presume that the implementation of the requirements is planned in work items, UTC captures 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 as a new revision in a VCS (Process B), or *after* implementation when they have created code that is already stored as a revision in a VCS (Process C). These three processes are depicted in Figure 1. In general, developers should not perform any change in the code without a work item describing the realization of a requirement.

Requirements, work items and revisions are stored as artifacts in UTC. However, revisions only contain a subset of information (revision number, author, creation date and list of changed code files) that is stored in the VCS. More detailed information, e.g. what lines of code were changed in the revision, can be found in the VCS.

2.1.1 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 files, which means s/he confirms all related and removes all non-related requirements or code files. The validated requirements are automatically linked to the work item and the validated code files are stored in a new revision in the VCS. A new revision artifact is automatically created and stored in UTC and linked to the work item.

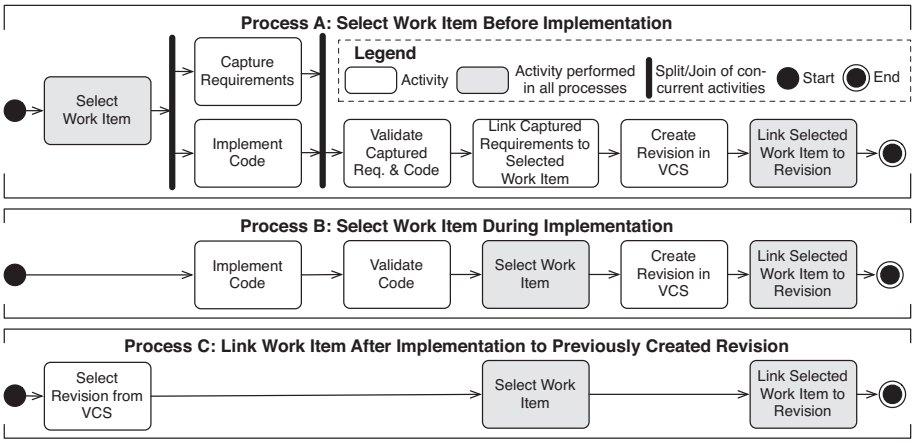

Figure 1: Traceability Link Creation Processes

2.1.2 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. A new revision artifact is automatically created and stored in UTC and linked to the selected work item. In this process, no requirements are captured and 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.

2.1.3 Process C) Link Work Item After Implementation

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. A new revision artifact is created and stored in UTC and linked the selected work item. Like in Process B, no requirements are captured and validated.

Issue tracking systems (e.g. Trac) and project management applications (e.g. Redmine) also support VCS integration like UTC, which means linking work items to revisions. However, these tools do not support requirements as discrete artifacts. Tools supporting the same artifacts as UTC are, for example, IBM Rational Team Concert or Polarion Requirements. However, unlike these tools, UTC can capture links between these artifacts during development. Moreover, UTC can automatically infer direct traceability links between requirements and code using work items, which is explained in the following.

2.1.4 Inferring Traceability Links Between Requirements and Code

The created traceability links of Processes A, B and C are used by UTC to infer direct links between requirements and code based on the corresponding work items. In [DNP12], 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.

Changes in the code do not have a direct impact on the artifacts and links stored in UTC. However, changes in the code can lead to new (inferred) links to requirements. Over time, inferred links might become obsolete due to work on other work items. Thus, we are currently working on intelligent algorithms to discard links not relevant anymore.

Summing up, the only manual work in UTC is to establish initial links between work items and requirements (which is typical for issue management) and to validate the captured links (which should be easy as the links refer to the work just finished). Besides this, there is no other additional work required to achieve traceability between requirements and code.

2.2 Features for Traceability Link Usage

Versioning: The EMFStore [EMF] is a repository and VCS for the Eclipse Modeling Framework designed for collaborative editing and versioning of models. The EMFStore is the foundation for UNICASE, and thus UTC. All artifacts in UTC are part of a model that is versioned with EMFStore. This allows versioning all artifacts and the traceability links between them and as a result, supports merging and conflict detection. For example, one can follow all changes of a requirement and its traceability links over time as well as revert to a previous version.

Graph Visualization: UTC supports graph visualization of all artifacts and the traceability links between them. Advanced layout algorithms can be applied to the graph and one can search within the graph.

Traceability between Requirements & Code: Using inferred links between requirements and code, UTC helps to analyze which code contributes to the realization of which requirement.

Requirements Context: During implementation, a developer can look at the requirements context that shows all requirements linked to the currently open code file. Due to agile software development techniques, development teams can change quickly. This feature supports new developers joining the project trying to understand the purpose of the implemented code.

Progress of Implementation: Work items have a completion status and are linked to requirements. Thus, work items enable to identify not implemented requirements as well as the progress of their implementation. UTC enables to see how far all requirements are already implemented, as well as identifying not implemented requirements requiring increased attention.

Requirements Impact Analysis: If a requirement needs to be changed to reflect changed customer demands, all related artifacts potentially affected by this change can be identified. Affected requirements and work items can be identified, e.g. if a change in a requirement is comprehensive, related requirements and their planning of realization described in work items needs to be adapted. An initial set of code files can be identified, which can be a starting point for detailed impact analysis. The changes in the code files can result in additional changes in other code files.

3 Conclusion

We developed UTC and it integrates requirements engineering, project management and code implementation in a single environment with full traceability between all artifacts. Currently UTC is used in an academic case study. We want to compare the effort and quality of the created traceability links to the results of other conducted exploratory case studies. Furthermore, we will work on intelligent algorithms to discard links between requirements and code not relevant anymore due to work on other work items.

References

[DNP12] Delater, A.; Narayan, N.; Paech, B.: Tracing Requirements and Source Code During Software Development. In ICSEA 12: Proc. 7th Int. Conf. of Software Engineering Advances, Lissabon, 2012; pp. 274-282

[EMF] EMFStore, A model repository for EMF-based models, http://eclipse.org/emfstore/ (Last Access: January 30, 2013)

[UNI] UNICASE, Technical University Munich, Chair for Applied Software Engineering, http://www.unicase.org/ (Last Access: January 30, 2013)

[UTC] UNICASE Trace Client at Google Code, http://code.google.com/p/unicase/wiki/ TraceClient (Last Access: January 30, 2013)