2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement

Tracing Requirements and Source Code During Software Development: An Empirical Study

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]* In practice, traceability links between requirements and code are often not created, because this would require increased development effort. To address this weakness, we developed in previous work an approach that semi-automatically captures traceability links between requirements and code during development. We do this by using work items from project management that typically are stored in issue trackers. *[Question/problem]* Practitioners and researchers have discussed the practice of using work items to capture links between requirements and code, but there has been no systematic study of this practice. *[Principal ideas/results]* In this paper, we present such an empirical study based on the application of our approach. We applied our approach in three different software development projects conducted with undergraduate students. We evaluated the feasibility and practicability of our approach and its tool support. The feasibility results indicate that our approach creates correct traceability links between all artifacts with high precision and recall during development. At the same time the practicability results indicate that the subjects found our approach and its tool support easy to use. *[Contribution]* The empirical evaluation in this paper contributes valuable insights into the tracing of requirements and code during software development.**

Keywords*—*traceability; requirements; work items; code; software development; evaluation.

I. Introduction

Requirements-to-code traceability reflects the knowledge where requirements are implemented in the code. This knowledge is very important for participants within a software development project [1]. However, these traceability links are often not created as this would lead to higher development effort. Therefore, these traceability links cannot be used during software development. However, researchers have shown that such links would be useful during development. For example, developers could use these links for comprehension support while working on an implementation task, e.g., navigating from a code file to its corresponding requirement [2].

In previous work, we presented an approach that semi-automatically captures traceability links between requirements and code during software development by using work items from project management. Our approach consists of three parts. The first part [3] comprises a Traceability Information Model (TIM) consisting of artifacts from three different areas, namely requirements engineering, project management, and code. The TIM also includes the traceability links between them. We also presented an algorithm for automatically inferring direct traceability links between requirements and code based on the interlinked work items. The second part [4] comprises three processes for capturing traceability links between requirements, work items, and code. The third part [5] is an implementation of our approach as an extension to the model-based CASE tool UNICASE [6], which is called UNICASE Trace Client (UTC) [7]. UNICASE is a plug-in for the Eclipse integrated development environment (IDE) and is developed in an open source project [8]. UTC integrates itself seamlessly into Eclipse and its supporting plug-ins, e.g., Subversion (a commonly used version control system). UTC implements the TIM and all its artifacts and traceability links as well as all three traceability link creation processes.

The capture of traceability links during development is the focus of recent research [9]. Asuncion & Taylor [10] presented an approach for capturing links between heterogeneous artifacts, including requirements and code, by analyzing interactions of users while they create/generate or modify artifacts. Omoronyia et al. [11] capture links between requirements and code based on the operations carried out by developers creating code artifacts to realize requirements. Requirements, work items and code can already be stored together in various tools, e.g., in IBM Rational Team Concert or Polarion Requirements. Practitioners and researchers have discussed the practice of using work items to capture links between requirements and code, but there has been no systematic study of this practice so far [12]. Such an empirical study should analyze the feasibility and practicability of an approach. Feasibility evaluates the accuracy of an approach, while practicability evaluates the ease of use of applying an approach in practice and whether the subjects have an intention to use the approach. Asuncion & Taylor and Omoronyia et al. did not provide empirical work to show the feasibility and practicability of their approaches.

In this paper, we present such an empirical study based on the application of our own approach in three different software development projects conducted with undergraduate students. We evaluate the feasibility and practicability of our approach and its tool support. For feasibility, we are investigating the precision and recall of traceability links between requirements and work items, work items and code, and requirements and code. For practicability, we use questionnaires to ask students about the ease of use and their intention to use our approach.

The paper is structured as follows: Section II provides background information on our approach. Section III describes the case study design and research method. Section IV presents the results and Section V the threats to validity. Section VI discusses related work and Section VII provides a discussion. Section VIII concludes the paper and discusses future work.

978-0-7695-5056-5/13 $26.00 © 2013 IEEE
DOI 10.1109/ESEM.2013.16

II. Background

This section provides background information about our approach to improve the general understanding.

A. Traceability Information Model

In [3], we defined a TIM (see Fig. 1) 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 between them. An extended UML notation was used to represent these three models with their artifacts.

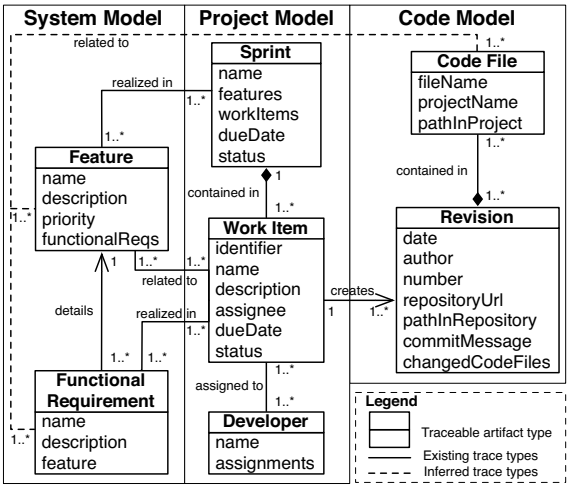

Fig. 1. Traceability Information Model (cf. [3])

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 have already been assigned to developers. Below, the term *requirement* refers to both: features and functional requirements.

B. Traceability Link Creation Processes

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. In [4], we identified three possibilities for 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 has already been stored in a VCS (process C). All three processes are depicted in Fig. 2 and explained in the following.

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 find out 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.

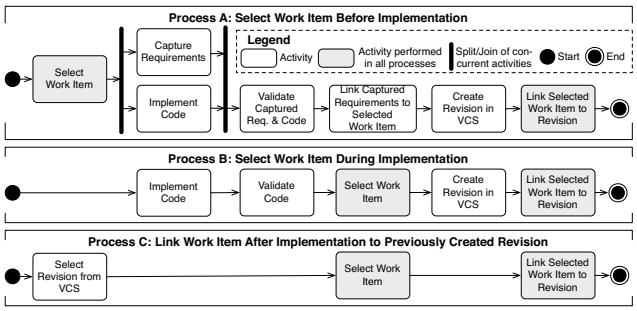

Fig. 2. Traceability Link Creation Processes A, B and C (cf. [4])

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 with the implementation directly. After implementing 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 the history of all previously created revisions with the 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 developer can perform a mixture of all three processes during the course of the project. However, each process 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.

III. Case Study Design & Research Method

In the following, the study context and the research questions and hypotheses are described.

A. Study Context

We conducted three different development projects with undergraduate students of different duration and number of participating students. In the following, we describe the development projects and provide information about the participants and the development process used. Table I provides an overview of the key metrics of all three projects. For describing features and functional requirements, we used the user task descriptions by Lauesen [13] and use cases, respectively.

TABLE I. Development Projects

Metric	Project 1	Project 2	Project 3
Participants	6	3	3
Sprints	6	3	3
Programming Lang.	JavaScript, Java	Java	Java
Duration	5 months	3 weeks	3 weeks
Features	2	1	1
Functional Reqs.	6	4	5
Work Items	395	51	20
Code Files	183	25	23
Lines of Code	5528	3827	9527
Revisions	694	80	165

For project 1, we were working together with a company from industry specialized in mobile business applications. The company integrates existing business applications such as ERP systems with mobile applications for smart phones and tablet computers. For this company, a system was developed retrieving data from various Internet data sources (e.g., Wikipedia, Google News). The system is capable of answering recurring questions based on input data, e.g., a company name, and the retrieved data, for example: "Who is the CEO?" or "What are recent news?". The company had a great interest in full traceability between requirements and code, because they wanted to maintain the developed application later on. The company did not provide a list of requirements before the project. Therefore, the students had to elicit the requirements themselves. The requirements did change during development to reflect the changed demands of the company. The functionality could be described in only a small number of requirements, but these requirements were very complex. JavaScript was used as the main programming language, with only a small subset of code programmed in Java. Because JavaScript was used, the functionality could be realized with a small amount of lines of code. The same functionality would have required notably more lines of code if programmed in another programming language. Therefore, the lines of code are not comparable across the projects. The project lasted five months from Oct. 2012 until Feb. 2013 and was divided into six sprints.

The project descriptions for projects 2 and 3 were identical. In both projects, an extension to UNICASE was developed that identifies missing links between all elements of a project. For example, a work item is missing an assigned developer. We acted as the "customer" for both projects, because we wanted to use this extension in our future development projects and we provided the subjects with a list of requirements before the project. The requirements did not change considerably during development. Compared to project 1, the requirements were less complex. Java was used as the programming language. The team of project 2 implemented more efficient code than the team of project 3, thus requiring less lines of code. Both projects lasted three weeks from mid Feb. 2013 until the beginning of Mar. 2013 and were divided into three sprints.

We recruited a total of 12 undergraduate students for our development teams, all having basic knowledge in software engineering. However, the team from project 1 was more advanced in their studies and had a more extensive knowledge in software engineering. To decrease variability in knowledge across students regarding UTC, we provided an introductory tutorial of UTC [7]. All teams were required to use UTC to store and link all artifacts. All teams applied agile software development techniques, e.g., they held regular stand-up meetings discussing completed work, planned work and any problems preventing them to continue work. The development process was as follows: in the beginning, the team elicited and/or specified a first draft of the requirements. In each sprint, the team detailed the requirements (if necessary) and broke them down into work items describing their realization. They assigned each work item to a developer and included it in a sprint. The team realized the requirements as described in the work items, which means they implemented the code. Thus, the situation we presume was ensured (see Section II-A).

B. Research Questions & Hypotheses

The goal of this case study is to get an understanding of the feasibility and practicability of our approach in practice. According to the Goal Question Metric (GQM) template by Basili et al. [14], these goals can be reformulated as:

- *Goal 1*: Analyze the approach for the purpose of understanding with respect to *feasibility* from the viewpoint of *approach developers*.
- *Goal 2*: Analyze the approach for the purpose of understanding with respect to *practicability* from the viewpoint of *approach users*.

1) Feasibility: Feasibility studies evaluate the accuracy of the results achieved by the approach [15]. In our study, this means the precision and recall of the created traceability links between requirements and work items, work items and code, and requirements and code. Precision and recall are two standard metrics used in information retrieval [16]. 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*. We distinguish three types of correct traceability links:

1) *Requirement and Work Item*: a link between a requirement and a work item where the work item describes (in part or whole) the realization of the requirement.
2) *Work Item and Code*: a link between a work item and a revision where the revision contains code that realizes the work described in the work item.
3) *Requirement and Code*: "a link between a requirement and its code where the code is necessary to realize the requirement" [4].

For comparing precision and recall across experiments, another metric known as F-Measure is used. F_2-Measure is a variant of F-Measure, which weights recall values more highly than precision [17]. All metrics are computed as follows:

$$Precision = \frac{|RelevantLinks \cap RetrievedLinks|}{|RetrievedLinks|} \quad (1)$$

$$Recall = \frac{|RelevantLinks \cap RetrievedLinks|}{|RelevantLinks|} \quad (2)$$

$$F_2Measure = \frac{3 * Precision * Recall}{(2 * Precision) + Recall} \quad (3)$$

While the links between requirements and work items and work items and code are created by the developers themselves, the links between requirements and code are automatically created by an inference algorithm presented in [3]. UTC does not provide the functionality to manually create links between requirements and code. As our approach creates links during development at the end of each sprint, we are interested in the precision and recall per sprint (links only created during the sprint), aggregated from sprint to sprint (all links created until a particular sprint) and at the end of the project. UNICASE uses the EMFStore [18] framework for storing and versioning all artifacts and their changes. We use EMFStore to access all artifacts per sprint and at the end of the project to calculate precision and recall. We manually identified all correct links and calculated precision and recall with the given equations. During calculation, we also considered special cases. For example, a correct link between requirement A and code file A is created in sprint 1. If requirement A changes during sprint 2, code file A could be no longer related to the requirement, but a new code file B was created and linked to requirement A. Thus, the link between requirement A and code file A would be incorrect in sprint 2, but remains correct in sprint 1.

TABLE II. FEASIBILITY RESEARCH QUESTIONS & HYPOTHESES

Research Question	Hypothesis
F-RQ1: What is the precision and recall of the created links between *requirements and work items "per sprint"* and at the *"end of the project"*?	The hypothesis **F-H1** is that high values for precision and recall will be achieved, which means 80% or more.
F-RQ2: How does the precision and recall of the links between *requirements and work items* develop *from sprint to sprint* in the project?	The hypothesis **F-H2** is that precision and recall will not fluctuate considerably from sprint to sprint in the project.
F-RQ3: What is the precision and recall of the created links between *work items and code "per sprint"* and at the *"end of the project"*?	The hypothesis **F-H3** is that high values for precision and recall will be achieved, which means 80% or more.
F-RQ4: How does the precision and recall of the links between *work items and code* develop *from sprint to sprint* in the project?	The hypothesis **F-H4** is that precision and recall will not fluctuate considerably from sprint to sprint in the project.
F-RQ5: What is the precision and recall of the created links between *requirements and code "per sprint"* and at the *"end of the project"*?	The hypothesis **F-H5** is that high values for precision and recall will be achieved, which means 80% or more.
F-RQ6: How does the precision and recall of the links between *requirements and code* develop *from sprint to sprint* in the project?	The hypothesis **F-H6** is that precision and recall will not fluctuate considerably from sprint to sprint in the project.
F-RQ7: How often is each traceability link creation process executed?	The hypothesis **F-H7** is that all processes are executed equally frequently.
F-RQ8: What is the precision and recall for all links per sprint created by process A?	The hypothesis **F-H8** is that high values for precision and recall will be achieved, meaning 80% or more.

Table II shows the resulting research questions (F-RQ1 to F-RQ6) and hypotheses (F-H1 to F-H6), which are quite similar to each other as they each cover one of the three types of correct traceability links. We used the scale of 80% or higher for precision and recall in our hypotheses, because results on this scale indicate that an approach delivers high quality links, which is comparable to manually performed linkage [19]. As all traceability links are created by either one of the processes, we measure how often each traceability link creation process is executed (see F-RQ7 and F-H7). Processes B and C only create links between work items and code. Since process A creates traceability links between requirements and work items as well as between work items and code, and traceability links are inferred between requirements and code, we analyzed this process in more detail (see F-RQ8 and F-H8).

2) Practicability: Practicability studies evaluate the practicability of a method when it is applied by the *approach users* instead of the *approach developers* [15]. In our case, the *approach users* are the undergraduate students in the three software development projects. To evaluate the practicability of our approach, we build upon the Technology Acceptance Model (TAM) [20]. TAM is modeling the user acceptance of information technology. In TAM, the actual use of a technology is determined by the subjects' intension to use it. The intention to use a technology is determined by its perceived usefulness and its perceived ease of use. The variables of TAM are:

- *Ease of use* refers to the degree to which a person expects the target system to be effortless.
- A person's *intention to use* determines whether s/he can imagine using the technology in the future or not.
- *Usefulness* is defined as a person's subjective probability that using a specific system will increase her/his job performance within an organizational context.

We use questionnaires to ask the subjects about the practicability of UTC. In the questionnaire, we focus on the variables "ease of use" and "intention to use". The variable "usefulness" cannot be considered because perceived usefulness can only be investigated when subjects work in an organizational context [20], which is not the case with the subjects of our case study.

TABLE III. PRACTICABILITY RESEARCH QUESTIONS & HYPOTHESES

Research Question	Hypothesis
1. Questions in the Questionnaire regarding Ease of Use	
P-RQ1: Is it easy to create traceability links between requirements. and work items?	The hypothesis **P-H1** is that UTC makes it easy to create traceability links between requirements and work items.
P-RQ2: Is it easy to create traceability links between work items and code?	The hypothesis **P-H2** is that UTC makes it easy to create traceability links between work items and code.
P-RQ3: Is it easy to infer traceability links between requirements and code?	The hypothesis **P-H3** is that UTC makes it easy to infer traceability links between requirements and code.
2. Questions in the Questionnaire regarding Intention to Use	
P-RQ4: Is it easy to use the inferred traceability links between requirements and code?	The hypothesis **P-H4** is that UTC makes it easy to use the inferred traceability links between requirements and code.
P-RQ5: Do the subjects have a concrete intension to use UTC?	The hypothesis **P-H5** is that the subjects have a concrete intension to use UTC.
3. Analyses of the Project Data	
P-RQ6: How often do developers use the inferred links between requirements and code for direct navigation?	The hypothesis **P-H6** is that developers use at least 20% if the inferred traceability links for direct navigation.

Table III shows the resulting research questions P-RQ1 to P-RQ5 and corresponding hypotheses P-H1 to P-H5. The sub-

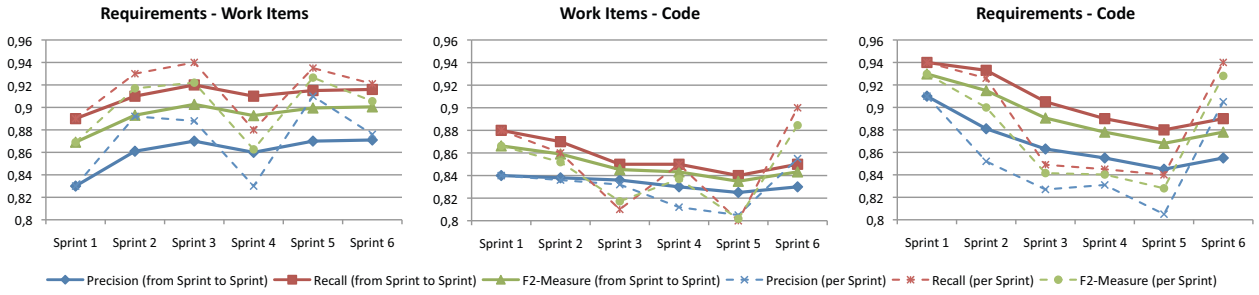

Fig. 3. Project 1: Precision, Recall, and F_2-Measure for Links Between Requirements and Work Items, Work Items and Code, Requirements and Code

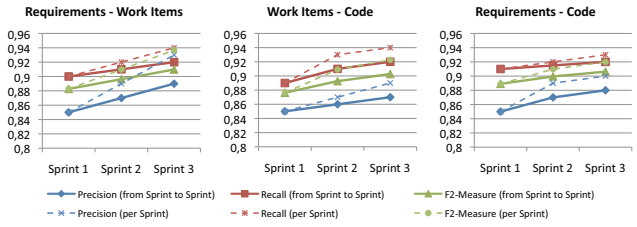

Fig. 4. Project 2: Precision, Recall, and F_2-Measure for Links Between Requirements and Work Items, Work Items and Code, Requirements and Code

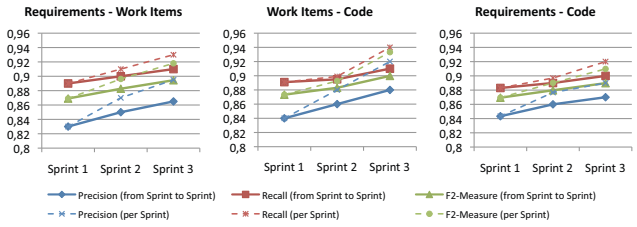

Fig. 5. Project 3: Precision, Recall, and F_2-Measure for Links Between Requirements and Work Items, Work Items and Code, Requirements and Code

jects have to assess predefined statements in the questionnaire. We use predefined statements to ensure the comparability of the subjects' responses. Furthermore, we ask the subjects to provide responses in free text form to get individual feedback. As the number of available subjects is too small to achieve statistical evidence, we wanted to collect as much individual feedback as possible. Therefore, we ask the subjects to provide a rationale for each assessment. The subjects score each statement on a six point Likert scale [21]. The Likert scale is an established approach in survey research for scaling the subjects' responses. If the majority of subjects tick 4 or higher on the Likert scale, we consider the statement as confirmed. If less than the majority of subjects tick 4 or higher on the Likert scale, we consider the statement as rejected.

In addition to the results from the questionnaire, we analyzed the gathered project data regarding one further aspect that is related to practicability. In P-RQ4, we asked the subjects how easy it is to use the links between requirements and code. Furthermore, UTC measures how often each link between requirements and code is used for direct navigation. For each link we logged how often it is used. Each link has a "click counter" that increases each time a developer uses it for direct navigation (see P-RQ6 and P-H6 in Table III).

IV. Results

In the following, we report on the results from the analyses of the feasibility and practicability of our approach. As stated in Section III-B, we manually identified all correct traceability links between all artifacts. Wrong traceability links are created, e.g., when developers change code files that are not particularly related to a work item. Missing correct links are created when work items are linked to the wrong requirement, e.g., when several requirements have similar names like the use cases "Manage Request" and "Manage Result" in projects 2 and 3, or when these links are not created at all. Both situations are potential causes of errors decreasing precision and recall in our approach. However, it has been shown that such "linkage bias is more likely due to the development process rather than being a side effect of the linking heuristics" [22], which means it is not uncommon to have linkage bias in development projects.

A. Feasibility

Figs. 3-5 show the precision, recall, and F_2-Measure for the traceability links between requirements and work items (left), work items and code (middle), and requirements and code (right) for the three development projects 1, 2, and 3. Straight lines represent aggregated values from sprint to sprint, while dashed lines represent values per sprint. In the following, the research questions and hypotheses are discussed together for all three relationships. Each project is discussed one after another. Finally, a conclusion is drawn for each specific type of relation and the hypotheses are confirmed or rejected.

Requirements and Work Items (F-RQ1, F-RQ2): In the graphs, the trend for aggregated precision and recall from sprint to sprint between requirements and work items is increasing in all three projects. All graphs for F_2-Measures from sprint to sprint have an upward trend. At the beginning of all projects, the precision and recall for links between requirements and work items were low. All teams improved from sprint to sprint and achieved good results at the end of the project. Especially for project 1, the values for precision and recall were particularly high for sprint 3, because the project manager of this sprint did a good job and looked after the work items and the requirements very thoroughly. After sprint 3, precision and recall declined slightly per sprint as new work items were created, but not all of them were linked correctly to requirements. The aggregated values for precision and recall recovered and reached a peak in sprint 6 with a precision of

0.871 and a recall of 0.916. Since the values for precision and recall are higher than 80% in all projects per sprint and at the end of the project, our hypothesis F-H1 is confirmed. As precision and recall do not fluctuate considerably from sprint to sprint, hypothesis F-H2 is also confirmed.

Work Items and Code (F-RQ3, F-RQ4): For project 1, precision and recall per sprint and from sprint to sprint decreased over time up to sprint 5 to a minimum of an aggregated precision of 0.825 and a recall of 0.84. The reason was that one project member constantly kept implementing code that was not particularly related to the work described in the work items. As the project was about to finish in sprint 6, the remaining five team members tried to improve their implementation of the remaining work items. The said project member stopped the unhelpful behavior of implementing unnecessary code, which resulted in a considerable increase in precision and recall per sprint between the work described in the work items and the actual implementation in the code, resulting in an aggregated precision of 0.83 and a recall of 0.85. In contrast to project 1, precision, recall and F_2-Measures kept fairly stable during projects 2 and 3 with a slight increase at the end. Since the values for precision and recall are higher than 80% in all projects per sprint and at the end of the project, our hypothesis F-H3 is confirmed. As precision and recall do not fluctuate considerably from sprint to sprint, hypothesis F-H4 is confirmed as well.

Requirements and Code (F-RQ5, F-RQ6): As in project 1 one project member kept implementing non-relevant code, the precision and recall for traceability links between requirements and code decreased per sprint as well as aggregated from sprint to sprint. In sprint 6, the team refactored the entire code base and removed unnecessary code introduced by one project member. The code was so unnecessary that the developed software was still compilable and runnable without any noteworthy missing features or necessary adjustments to the code base after the code was removed. This refactoring was very successful and greatly increased precision and recall between requirements and code at the end of the project, reaching an aggregated precision of 0.835 and a recall of 0.89. However, both values never reached the peak of the beginning of the project, because at the beginning only few requirements and code were available and precision and recall were particularly high. For projects 2 and 3, precision, recall and F_2-Measures kept increasing steadily from sprint to sprint. As both teams had only little experience in using the frameworks and technologies for implementing an extension for UNICASE (although they were introduced to these frameworks and technologies in practical courses at our university before), they tried different implementations that were not particularly relevant for the realization of the requirements. Therefore, the values for precision and recall were low in the beginning of the projects, but steadily increased as unnecessary code was removed from sprint to sprint and the team learned better how to implement the required functionality. At the end, this resulted in an aggregated precision of 0.88 and a recall of 0.92 for project 2 and an aggregated precision of 0.87 and a recall of 0.90 for project 3. Therefore, we can confirm our hypothesis F-H5, as all values for precision and recall are higher than 80% in all projects per sprint and at the end of each project. As precision and recall do not fluctuate considerably from sprint to sprint, hypothesis F-H6 is confirmed as well.

How often is each of the three traceability link creation processes executed? (F-RQ7) As described in Section II-B, each revision is created or linked by one of the three traceability link creation processes. This means that the number of revisions equals the number of executed processes. UTC logged the used process for each revision. Fig. 6 shows how often each process was executed for each project. The majority of executed processes were process B and C with 66%-72%. Process A was only used between 5%-10% in all three projects. This rejects our hypothesis F-H7 that all processes are executed equally frequently.

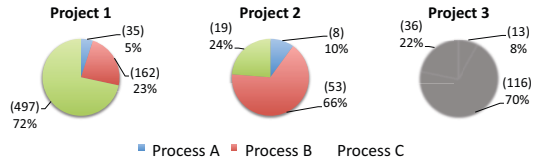

Fig. 6. Execution of Traceability Link Creation Processes by Project

Process A would require the subjects to select a work item before development. As the selection of a work item also occurs during process B, the subjects did not execute process A often, because they knew the other selection possibility would come in process B. In process B the developer is reminded to select a work item before committing the changes in the code to the VCS to create a new revision. Furthermore, we think that because the requirements did not change considerably during development in projects 2 and 3, the subjects could better link work items during development (process B) as the work described in the work item was more precise. In project 1, the subjects first implemented code and committed a new revision to the VCS and then linked a matching work item to the revision (process C).

As all three processes are executed before, during, or after implementation, we analyzed more deeply how the subjects used the processes. To accomplish this, UTC logged in detail how the subjects used the processes. As process A includes a validation step for the captured requirements, UTC logged whether the subjects confirmed or rejected the captured requirements. In all the cases it was used, the subjects confirmed the captured requirements and did not reject any of them. The confirmed requirements were the ones that were linked to the previously selected work item. Here, the subjects read the work described in the work item and looked at the linked requirement(s) to get an even better understanding. Therefore, this link was valid and did not have to be rejected. We analyzed whether the subjects linked new requirements to the selected work item after the execution of process A, but no subject linked new requirements afterwards. Both processes A and B include a validation step for the changed code files. Again, in all cases the subjects confirmed all changed code files and did not reject any code files. During the execution of process C, the subjects had to select a work item to be linked to a previously created revision. We analyzed whether the subjects linked new requirements to the selected work item after the execution of process C, but no subject linked new requirements afterwards.

Process A (F-RQ8): Figs. 7-9 (see next page) show the precision, recall, and F_2-Measure for the traceability links created by process A in each particular sprint in all projects. Although process A was only executed between 5%-10% in

all projects, it can be seen that if it was executed, it achieved high results. In project 1, precision, recall, and F_2-Measure kept fairly stable with a slight upward trend for all traceability links between requirements and work items, work items and code, and requirements and code, reaching a precision between 0.90 and 0.92 and a recall between 0.92 and 0.94.

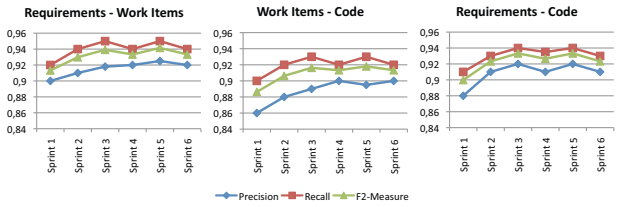

Fig. 7. Process A in Project 1: Precision, Recall, and F_2-Measure for Links between Requirements and Work Items, Work Items and Code, and Requirements and Code

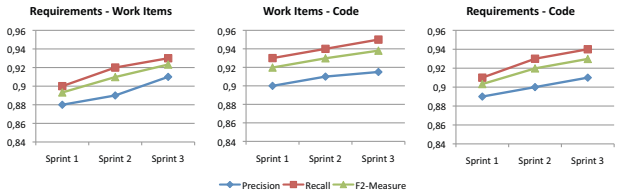

Fig. 8. Process A in Project 2: Precision, Recall, and F_2-Measure for Links between Requirements and Work Items, Work Items and Code, and Requirements and Code

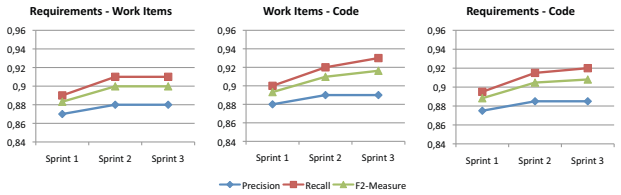

Fig. 9. Process A in Project 3: Precision, Recall, and F_2-Measure for Links between Requirements and Work Items, Work Items and Code, and Requirements and Code

In projects 2 and 3, the same trend can be recognized with slightly better values for precision, recall, and F_2-Measure. This means, if subjects used process A and implemented code, they looked at the requirements and knew exactly what to implement and only implemented/changed relevant code files. However, sometimes the subjects changed a small amount of lines of code, e.g., Java documentation, in another code file that was not related to the work described in the work item. Thus, 100% of precision and recall were not achieved.

B. Practicability

To assess the practicability of our approach, we used a questionnaire to ask the subjects whether it was easy to create links (see Table IV), easy to use links for direct navigation in UTC (see Table V on next page), and whether they have an intention to use UTC (see Table VI on next page).

P-RQ1: Is it easy to create traceability links between requirements and work items? These traceability links can either be created manually in UTC (1) or during the execution of process A (2) (see Table IV). The majority of subjects

TABLE IV. EASE OF USE - CREATION

It was easy to ...	Strongly Disagree	Disagree	Rather Disagree	Rather Agree	Agree	Strongly Agree
(1) create links between requirements and work items				5	7	
(2) confirm requirements captured during the work on a work item (Process A)					8	4
(3) link a work item to a revision *before* or *during* development (Process A, B)				2	7	3
(4) link a work item to a previously created revision *after* development (Process C)			2	3	5	2
(5) infer links between requirements and code				2	6	4

ticked 5 on the Likert scale, which confirms hypothesis P-H1. For question (1), the subjects provided the feedback that while it was easy to link a requirement to a work item, it was harder to link a work item to a requirement, because there was a large amount of work items in the project and finding the right one required knowing the name of the work item. Therefore, they did not "strongly agree" to question (1). For question (2), the subjects assessed that it was easy to confirm the captured requirements, because they were pre-selected in the confirm dialog. This required only one click to proceed to the next dialog for validating the code files.

P-RQ2: Is it easy to create traceability links between work items and code? These traceability links can either be created before or during implementation by executing process A or B (3) as well as after implementation by executing process C (4) (see Table IV). Subjects ticked between 4 and 6 on the Likert scale, which confirms hypothesis P-H2. For question (3), subjects provided the feedback that they liked that UTC reminded them to link a work item. However, again they had to search for the right work item which required knowing the name. The subjects liked the ability to filter the dialog for selecting a work item that was only assigned to themselves. For question (4), subjects "rather agreed" by responding that they would like to be able to select more than one revision at a time to be linked to a work item. Two subjects "rather disagreed", because currently it is not possible to see which revision is already linked to a work item in the history of the VCS. We will consider this feedback for improving UTC.

P-RQ3: Is it easy to infer traceability links between requirements and code? Question (5) in Table IV was concerned with the ease of inferring traceability links between requirements and code. All the subjects ticked 4 or higher on the Likert scale. The subjects confirmed the ease of use for inferring traceability links. Thus, hypothesis P-H3 is confirmed. The subjects especially liked that the inference process is initiated with a single push of a button in UTC and all links are created automatically by the inference algorithm. The subjects particularly liked the performance of the inference process. We measured the performance and achieved on average about 80 milliseconds for 600 revisions with linked work items and requirements for project 1. Thus, the subjects were able to instantly create the traceability links and use them for direct navigation. However, we did not conduct a comprehensive investigation of the performance of the inference algorithm. Results can differ depending on the project data and the computer hardware used.

P-RQ4: Is it easy to use the inferred traceability links between requirements and code? We asked the subjects whether it was easy to use the inferred traceability links between requirements and code for direct navigation (see Table V).

TABLE V. EASE OF USE - USAGE

It was easy to ...	Strongly Disagree	Disagree	Rather Disagree	Rather Agree	Agree	Strongly Agree
navigate between requirements and code using the inferred traceability links				2	6	4

As the majority of subjects ticked 4 or higher on the Likert scale, P-H4 is confirmed. The subjects liked that by opening a requirement in UTC, a list of linked code files was automatically presented. With a single click on such a code file, the subjects could navigate directly from the requirement to the code file. The subjects also liked that when they opened a code file in Eclipse, UTC showed all linked requirements of this code file in a separate list. However, two subjects responded that this list is not shown automatically by UTC and had to be enabled manually, thus they ticked only "rather agree".

P-RQ5: Do the subjects have a concrete intension to use UTC? In order to determine the subjects' intension to use our approach, we asked them whether they are motivated to use UTC in the future. Table VI shows their assessments.

TABLE VI. INTENSION TO USE

I'm motivated to use UTC ...	Strongly Disagree	Disagree	Rather Disagree	Rather Agree	Agree	Strongly Agree
in the future for storing all artifacts (requirements, work items, and code) in a development project			2	1	7	2
in the future for creating traceability links between all artifacts (requirements, work items, and code) in a development project				2	8	2
as it is currently integrated in UNICASE				1	9	2

The subjects mostly ticked 5 on the Likert scale. Thus, hypothesis P-H5 is confirmed. The subjects justified their statements by stating that the direct assignment of the code changes in form of a revision to work items is facilitating teamwork and clarity in the long run. Two subjects "rather disagreed" with storing all artifacts in one single environment, as they would have preferred to have quick web-based access to the artifacts. Currently, the change of a single piece of information always requires to open Eclipse with integrated UTC. The majority of subjects stated that they especially liked the seamless integration of UTC in UNICASE and Eclipse.

P-RQ6: How often do developers use the inferred traceability links between requirements and code for direct navigation? The subjects used a total of 25 of 387 links (6.5%) for project 1, 7 of 32 (21.9%) links for project 2, and 8 of 37 (21.6%) links for project 3. Thus, our hypothesis P-H6 holds for projects 2 and 3, but needs to be rejected for project 1. However, we noticed that particular types of links to code were used more often than others, especially the important code parts that comprise the core functionality. In project 1, code files containing the retrieval mechanisms for accessing the various Internet data sources were used often. In project 2 and 3, code files containing the search procedures for finding missing links as well as the main code files for creating the user interface were used often.

V. THREATS TO VALIDITY

Runeson et al. [23] distinguish four different threats to validity in case study research, which are discussed below.

Internal Validity is concerned with the correlation between the investigated factors and other factors [23]. The students knew that we had developed UTC and thus might have been biased towards UTC. Therefore, we explicitly advised the students to assess UTC objectively and that both, positive and negative feedback, are desired. To decrease the variability of knowledge across students regarding the tracing of requirements and code in UTC, we provided an introductory tutorial of UTC [7]. We had no influence on *how* the students created these links, we only ensured that they created links. Therefore, we had no direct influence on the results. The assessment of each created link is a manual task and cannot be automated. A potential bias is that the assessment was performed by the first author. However, due to the manageable scale of the projects, it was obvious whether a link was right or wrong.

External Validity is concerned with the extent to which the findings of a specific study can be generalized [23]. Due to temporal restrictions, the sizes of the development projects were limited, e.g., number of requirements and developed code. This does not allow us to draw conclusions on larger projects. In the development projects, all undergraduate students had basic knowledge in software engineering. However, no undergraduate student had industrial experience. This does not allow us to draw conclusions on more experienced developers from industry. However, case studies in an academic environment are common practice in empirical software engineering [23]. Studying approaches in practice is also rather difficult, as the industry is rarely willing to use research prototypes. Nevertheless, our projects contained situations common to industrial projects, e.g., the elicitation of requirements by the participants (project 1) vs. a provided list with requirements (projects 2-3), changing requirements due to changed customer demands, as well as communication problems with certain developers regarding their task responsibility. 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. During the projects we gave advice to the students and made sure that they used UTC. They may have behaved differently if they had not to use UTC.

Construct Validity is concerned with the intended observations of the researchers and their actual observations [23]. A possible threat to validity is the inadequate usage of the variables of TAM in our questionnaire. The questionnaire could have measured something different than TAM, because it was not evaluated under realistic conditions prior to the study.

Reliability Validity is concerned with the extent to which the data and the analyses are dependent on the specific researchers [23]. As we wanted to evaluate the feasibility of our approach, we had a great interest in the traceability links between requirements, work items, and code. The students knew that we would look at those links at the end of the project and our behavior could have influenced the students.

VI. Related Work

Empirical studies related to our work can be divided into two groups: studies evaluating the creation and studies evaluating the usage of links between requirements and code.

A. Empirical Studies on the Creation of Traceability Links

In [24] and the book of Cleland-Huang et al. [25], a general overview of requirements traceability is provided. As the manual creation of traceability links between requirements and code is error-prone, time consuming, and complex [26], research focuses on (semi-) automatic approaches. Existing approaches with empirical evaluations use various techniques, e.g., information retrieval [17] [27] [28] [29] [30] [31], execution-trace analysis [32] [33], static/dynamic analysis [34], subscription-based or rule-based link maintenance [35], or combinations of them [36]. However, in their evaluations all these approaches only focussed on the feasibility and not on the practicability, as we did with our approach in this work.

Egyed et al. [37] investigated the effort of recovering traceability links between requirements and code after development. In general, these traceability links were recovered by project members who were not directly involved in the realization of a particular requirement, but knew the code base. Our approach distributes the effort of creating traceability links to all developers actively participating in the project.

B. Empirical Studies on the Usage of Traceability Links

Maeder & Egyed [2] conducted a controlled experiment with 52 subjects (students of computer science) performing 315 maintenance tasks on two third-party development projects: half of the tasks with and the other half without traceability navigation. Their findings show that subjects with traceability performed on average 21% faster and created on average 60% more correct solutions, suggesting that traceability not only saves time but can profoundly improve software maintenance quality. As our approach creates traceability links between requirements and code during development, the traceability links are readily available for software maintenance. However, in all projects software maintenance tasks did not have to be performed as development ended when all requirements were realized and the project durations were fixed.

VII. Discussion

As shown above, the hypotheses F-H1 to F-H6 for feasibility were confirmed. This means that our approach creates correct traceability links between requirements and work items, work items and code, and requirements and code with high precision and recall during development. The processes were not equally frequently executed, which rejected hypothesis F-H7. Process A was only used in 5%-10% of all three projects. However, if process A was executed, it achieved high values for precision and recall, which confirmed hypothesis F-H8. The results for practicability from the questionnaires confirmed our hypotheses that our approach is easy to use (P-H1 to P-H4) and the subjects have a concrete intention to use our approach (P-H5). We had to partially reject our hypothesis P-H6, as in project 1 only 6.5% of the links were used for direct navigation between requirements and code. We believe that the direct navigation will be used more often in projects where new developers join the team who need to understand how the existing code files relate to the requirements. We did not have this situation in our projects. We think that process A will be used more often in larger projects, as there was only a small amount of requirements in our projects. Here, the subjects were very familiar with the requirements and did not need to look at them often during development. We think that in larger projects with more requirements it is more likely that the subjects will use process A, because the subjects cannot keep every requirement in mind. Furthermore, we think that for larger projects, the subjects need training courses to get familiar with the three traceability link creation processes.

We argue that the additional manual work required of the subjects for applying the traceability link creation processes is justified by the benefits of early availability of high-quality traceability links. We showed that our approach achieves good results with respect to the quality of the links and the subjects rated our approach as easy to use in practice. We consider the additional work required for our approach as minimal, as our approach seamlessly integrates into the regular development workflow that the subjects had to perform anyway, which is the implementation and check-in of code into the VCS.

In the current approach, developers might make mistakes when adding non-related requirements to a work item or implementing/changing code that is not described in the work item. This would create incorrect traceability between these artifacts. However, this risk is reduced since we let the developers validate all traceability links before they are created. It has been shown that humans were better at validating links as opposed to searching for missing links [38]. This strengthens our approach of letting the developers validate the links to be created instead of recovering links or searching for missing links. During development, links between requirements and code can become irrelevant when a requirement has changed considerably so that the linked code is no longer relevant for the realization of the particular requirement. It is an open research issue for us whether we can improve our inference algorithm so that it can detect these non-relevant links automatically and discard them. This would make our inference algorithm more ”robust” against this cause of error.

Before we conducted the three development projects, we investigated whether instead we could use data from open source projects or mining challenges from the Mining Software Repositories [39] community for evaluation. However, we found out that these datasets usually only consist of work items and code managed in issue trackers and VCSs respectively. Therefore, we could not use these datasets, because no explicit requirements were available and connected to the work items. Thus, we had to carry out our own development projects comprising requirements, work items, and code.

VIII. Conclusion and Future Work

In this paper, we presented an empirical study based on our approach for tracing requirements and code during development using work items from project management. We applied our approach in three development projects conducted with undergraduate students. Based on the data gathered in the projects, we have shown the feasibility and practicability of our approach in practice. The major finding of our evaluation is that our approach creates correct traceability links of high quality during development. Another finding is that developers

mainly used process B and C during development. The subjects rated our approach and its tool support as easy to use and used the links between requirements and code for direct navigation.

In future work, we want to apply existing automated approaches for creating traceability links between requirements and code on the data we gathered in our three projects and compare these results to the results achieved by our approach. Furthermore, we want to improve our inference algorithm to discard non-relevant links between requirements and code to improve precision and recall of our approach. Finally, we hope that our study can be seen as a first step in researching the tracing of requirements and code during development and that it will be replicated in an industrial setting in the future.

ACKNOWLEDGMENT

The authors would like to thank the company for providing the opportunity to realize one of the projects, all students for their participation, as well as Ulrike Abelein, Florian Flatow and Robert Heinrich for their help in organizing the projects.

REFERENCES

[1] Gotel, O. et al. Traceability Fundamentals. In Software and Systems Traceability, J. Cleland-Huang, O. Gotel, and A. Zisman (Eds.), Springer, pp. 3-22 (2012)

[2] Maeder, P. and Egyed, A. Do software engineers benefit from source code navigation with traceability? - An experiment in software change management. In Proc. of the 26th Int. Conf. on Automated Software Engineering, pp. 444-447 (2011)

[3] Delater, A., Narayan, N., and Paech, B. Tracing Requirements and Source Code during Software Development. In Proc. of the 7th Int. Conf. on Software Engineering Advances, pp. 274-282 (2012)

[4] Delater, A. and Paech, B. Analyzing the Tracing of Requirements and Source Code during Software Development: A Research Preview. In Proc. of the 19th Int. Working Conf. on Requirements Engineering: Foundation for Software Quality, pp. 308-314 (2013)

[5] Delater, A. and Paech, B. UNICASE Trace Client: A CASE Tool Integrating Requirements Engineering, Project Management and Code Implementation. Workshop Nutzung und Nutzen von Traceability, In: Wagner S, Lichter H (Eds.): Software Engineering 2013 Workshopband, Lecture Notes in Informatics, vol. 215, pp. 459-463 (2013)

[6] Bruegge, B., Creighton, O., Helming, J., and Koegel, M. Unicase - an Ecosystem for Unified Software, In ICGSE'08: Distributed software development: methods and tools for risk management, pp. 12-17 (2008)

[7] UNICASE Trace Client, http://code.google.com/p/unicase/wiki/TraceClient [retrieved: June, 2013]

[8] UNICASE, http://www.unicase.org/ [retrieved: June, 2013]

[9] Maeder, P. and Egyed, A. Assessing the effect of requirements traceability for software maintenance. In Proc. of the 28th Int. Conf. on Software Maintenance, pp. 171-180 (2012)

[10] Asuncion, H. and Taylor, R. Automated techniques for capturing custom traceability links across heterogeneous artifacts. In Software and Systems Traceability, J. Cleland-Huang, O. Gotel, and A. Zisman (Eds.), Springer, pp. 129-146 (2012)

[11] Omoronyia, I., Sindre, G., Roper, M., Ferguson, J., and Wood, M. Use case to source code traceability: The developer navigation viewpoint. In Proc. of the 17th Int. Requirements Engineering Conf., pp. 237-242 (2009)

[12] Cleland-Huang, J. Traceability in agile projects. In: Cleland-Huang, J., Gotel, O., Zisman, A. (eds.) Software and Systems Traceability, pp. 265-275. Springer (2012)

[13] Lauesen, S. Task Descriptions as Functional Requirements. IEEE Software, vol. 20, no. 2, pp. 58-65 (2003)

[14] Basili, V.R., Caldiera, G., and Rombach, H.D. The Goal Question Metric Approach, Encyclopedia of Software Engineering, pp 528-532, Wiley and Son (1994)

[15] Eusgeld, I., Freiling, F.C., and Reussner, R. Dependability Metrics. Springer (2008)

[16] Frakes, W.B. and Baeze-Yates, R. (Eds.) Information Retrieval: Data Structures and Algorithms. Prentice-Hall (1992)

[17] Cleland-Huang, J., Czauderna, A., Gibiec, M., and Emenecker, J. A machine learning approach for tracing regulatory codes to product specific requirements. In Proc. of the 32nd ACM/IEEE Int. Conf. on Software Engineering, pp. 155-164 (2010)

[18] EMFStore, A model repository for EMF-based models, http://eclipse.org/emfstore/ [retrieved: June, 2013]

[19] Maeder, P. and Gotel, O. Ready-to-use Traceability on Evolving Projects. In Software and Systems Traceability, J. Cleland-Huang, O. Gotel, and A. Zisman (Eds.), Springer, pp. 173-194 (2012)

[20] Davis, F.D., Bagozzi, R.P., and Warshaw, P.R. User Acceptance of Computer Technology: A Comparison of two Theoretical Models, Manage. Sci. 35, pp. 982-1003 (1989)

[21] Likert, R. A Technique for the Measurement of Attitudes. Archives of Psychology, 140, pp. 1-55 (1932)

[22] Nguyen THD., Adams, B., and Hassan, AE. A Case Study of Bias in Bug-Fix Datasets. 7th Working Conf. on Reverse Engineering (WCRE), pp. 259-268 (2010)

[23] Runeson, P., Host, M., Rainer, A., and Regnell, B. Case Study Research in Software Engineering: Guidelines and Examples, Wiley&Sons (2012)

[24] Dahlstedt, A. and Persson, A. Requirements interdependencies: State of the art and future challenges. In Engineering and Managing Software Requirements, Aurum and Wohlin (eds.) Springer, pp. 95-116 (2005)

[25] Cleland-Huang, J., Gotel, O. and Zisman, A. (Eds.) Software and Systems Traceability, Springer (2012)

[26] Spanoudakis, G. and Zisman, A. Software traceability: A roadmap. Handbook of Software Engineering and Knowledge Engineering, World Scientific Publishing, pp. 395-428 (2004)

[27] Hayes, J.H., Dekhtyar, A., and Osborne, J. Improving requirements tracing via information retrieval. In Proc. of the 11th Int. Requirements Engineering Conf., pp. 138-147 (2003)

[28] De Lucia, A., Fasano, F., Oliveto, R., and 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)

[29] Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., and Merlo, E. Recovering traceability links between code and documentation. Transactions on Software Engineering, pp. 970-983, IEEE (2002)

[30] Marcus, A. and Maletic, J.I. Recovering documentation-to-source-code traceability links using latent semantic indexing. In Proc. of the 25th Int. Conf. on Software Engineering, pp. 125-135 (2003)

[31] Marcus, A., Maletic, J.I., and Sergeyev, A. Recovery of traceability links between software documentation and source code. Int. Journal of Software Engineering and Knowledge Engineering, vol. 15, no. 5, pp. 811-836 (2005)

[32] Egyed, A. A Scenario-Driven Approach to Trace Dependency Analysis. Transactions on Software Engineering, vol.29, no.2, pp. 116-132 (2003)

[33] Eisenberg, A.D. and De Volder, K. Dynamic feature traces: Finding features in unfamiliar code. In Proc. of the 21st IEEE Int. Conf. on Software Maintenance, pp. 337-346 (2005)

[34] Antoniol, G. and Gueheneuc, Y.G. Feature identification: A novel approach and a case study. In Proc. of the 21st IEEE Int. Conf. on Software Maintenance, pp. 357-366 (2005)

[35] Maeder, P. and Gotel, O. Towards Automated Traceability Maintenance. Journal of Systems and Software, vol. 85, no. 10, pp. 2205-2227 (2011)

[36] Eaddy, M., Aho, A.V., Antoniol G., et al. CERBERUS: Tracing requirements to source code using information retrieval, dynamic analysis, and program analysis. In Proc. of the 16th IEEE Int. Conf. on Program Comprehension, pp. 53-62 (2008)

[37] Egyed, A., Graf, F., and Gruenbacher, P. Effort and quality of recovering requirements-to-code traces: Two exploratory experiments. In Proc. of the 18th Int. Requirements Engineering Conf., pp. 221-230 (2010)

[38] Kong, W.-K., Huffman Hayes, J., Dekhtyar, A., and Holden, J. How do we trace requirements: an initial study of analyst behavior in trace validation tasks. In Proc. of the 4th Int. Workshop on Cooperative and Human Aspects of Software Engineering, pp. 32-39 (2011)

[39] MSR Conference, http://www.msrconf.org [retrieved: June, 2013]