Tool Support for Traceability-Adaptation

Marcus Seiler

Institute of Computer Science, University of Heidelberg
Im Neuenheimer Feld 326, 69120 Heidelberg, Germany
`seiler@informatik.uni-heidelberg.de`

Abstract. [Context & motivation] Traceability of software engineering artifacts is important in software development. Tools are used to establish traceability between software engineering artifacts. **[Problem]** One of the open traceability challenges is the traceability-configuration during usage. At any time within a project, the traceability information model (TIM) should be adaptable to changing contexts and needs. **[Principal idea]** Our approach for adaptable traceability is based on two main ideas. First, the TIM can be configured based on a comprehensive meta-model. Second, a TIM-repository enables consistent management of adapted TIMs and artifacts and links instantiated from it. **[Contribution]** This paper describes the problem, related work, main solution ideas, research methodology and progress so far.

Keywords: traceability, adaptation, information model, tool support

1 Introduction

Traceability is used to follow and understand relationships between various software engineering artifacts such as requirements, design artifacts or source code. Comprehensive traceability of software engineering artifacts is important to ensure the quality of a software being developed or being maintained. A recent study shows that traceability has a positive impact on software quality in terms of time to implement a solution and the correctness of the solution [14]. Although traceability has a positive effect, it is still far from satisfying in current development practices. One of the open challenges is traceability-configuration during usage [8]. Traceability tool support should be adaptable to changing contexts and needs. To provide adaptable traceability tool support, the project's traceability should be defined upfront. A traceability information model (TIM) can be used to define the traceability within a project [5]. According to [13], a basic TIM consists of at least two types of artifacts and a traceability relation between these artifacts. However, adaptable traceability provided by tools is insufficient in practice. Current software engineering tools such as DOORS[1], Polarion[2] or Jira[3] are e.g. limited by a fixed number of traceable artifact and

[1] *https://www.doorsng.com/*, last checked Feb 18, 2016
[2] *https://www.polarion.com/*, last checked Feb 18, 2016
[3] *https://www.atlassian.com/software/jira*, last checked Feb 18, 2016

link types as well as by a pre-defined TIM. Thus tools are inflexible in case of changing project needs. Therefore, this thesis aims to address this challenge by providing tool support for traceability-adaptation.

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

2 Problem

In the following, requirements on adaptable tool support are presented which were collected from literature and practice (cf. Sec. 5). The research question to answer is: How can adaptable traceability tool support be provided satisfying the following requirements?

R1-Comprehensive meta-model: Traceability tool meta-models have been already proposed [1, 2]. In industry, various software engineering artifacts exist and tools need to provide a wide spectrum of artifact types including e.g. requirements, architecture and code. However, tools are mostly restricted to traceability between two particular types of artifacts [19]. Many different link types with customizable semantics between artifacts are needed [7]. Link types describe the meaning of relations between two or more artifacts such as *requires*, *implemented in* or *verified by*. It must be possible to use these link types between all artifacts. Thus, due to the many artifact types and many link types a comprehensive meta-model is needed.

R2-Definition and application of a project-specific TIM: Traceability usage is project-specific and thus usage varies from project to project [4]. Based on the meta-model for different projects different TIMs that reflects different types of projects and usage scenarios need to be definable. The artifacts and links must be instantiated and validated according to the defined TIM.

R3-Incremental TIM-adaptation: Software evolves over time and traceability cannot be planned upfront [4]. To reflect changing contexts and needs, the TIM must be adaptable [5]. Being able to only adapt the TIM once is not sufficient. Our interview results show that an incremental TIM-adaptation is needed [16]. At the beginning, only few changes have been made on the traceability. Later, some changes wrt traceability were reverted or existing changes were refined.

R4-Consistent management of TIM and artifacts and links: Artifacts and links must comply to the TIM. After TIM-adaptation, existing artifacts and links must also be adapted. For example, if an artifact type is removed from the TIM, all corresponding artifacts must be deleted without losing traceability information. Furthermore, in case of artifact type addition, new artifacts can be stored. Similarly, for link type changes. If a TIM-adaptation is not helpful for a project, it should be possible to revert to prior consistent states of TIM and corresponding artifacts and links. Therefore, TIM version management is needed.

R5-Visualization of TIM and the corresponding artifacts and links: Our interviews showed that TIM-adaptations were documented explicitly to be able to discuss different TIM-versions [16]. Thus, a requirement is to provide visualization of different TIM-versions including the corresponding artifacts and links, e.g. display two different versions of a TIM.

3 Related Work

Providing traceability tool support is a challenging task and therefore a field of intense research. The search for related work (cf. Sec. 5 and 6) has so far revealed the following approaches. The approaches are assessed against the stated requirements.

Table 1: Requirements satisfaction of approaches

	Boronat [3]	De Lucia [6]	Kelleher [9]	Lindsay [11]	Macfarlane [12]	Maletic [15]	Taromirad [17]
R1 Comprehensive meta-model	√	√	√	√	○	○	√
R2 TIM-Definition and -application	√	○	○	√	-	√	√
R3 Incremental TIM-adaptation	√	√	-	-	-	√	√
R4 Consistent management	-	-	-	-	-	-	-
R5 TIM-Visualization	-	-	-	-	-	-	-

Tab. 1 shows the requirements satisfaction of the approaches whereby √ means full-, ○ means partial- and − means no satisfaction. In the following, the approaches from Tab. 1 are discussed wrt requirements satisfaction. Approaches to support model-to-model traceability where artifacts such as requirements or source code are defined as models are presented by [15, 3, 17]. All three approaches allow users to define their own meta-model to fit into a specific working context and allow TIM-adaptation during the project. However, the approach by Maletic et al. [15] only provides three different types of traceability links between the models. Furthermore, none of these approaches satisfies R4 or R5. De Lucia et al. [6] present an approach for fine-grained management of traceability links based on a hierarchy of software artifact documents. Traceability links between artifacts can be inserted manually and for each traceability link a stereotype describes the link semantics can be defined and also adapted. However, no restrictions of the link types between artifacts are made upfront. Thus, the approach lacks in application of a project-specific TIM. In addition, the approach does not satisfy R4 and R5. Kelleher [9] presents a framework for reusable traceability practices. The framework consists of a traceability meta-model and a traceability process. From the meta-model traceable artifacts are defined for the process providing a project-specific TIM. However, the TIM is only published on a website to guide users on traceability. The approach does not satisfy R3-R5.

Lindsay et al. [11] use a generic document model to represent and link artifacts. The approach allows definition and application of a project-specific TIM, but does not support R3-R5. Macfarlane et al. [12] describes the fine-grained requirements traceability based on files supporting traceability through all phases of the software life cycle. However, no traceability link types are defined by this approach, and lacks in satisfaction for R2-R5.

4 Solution Ideas

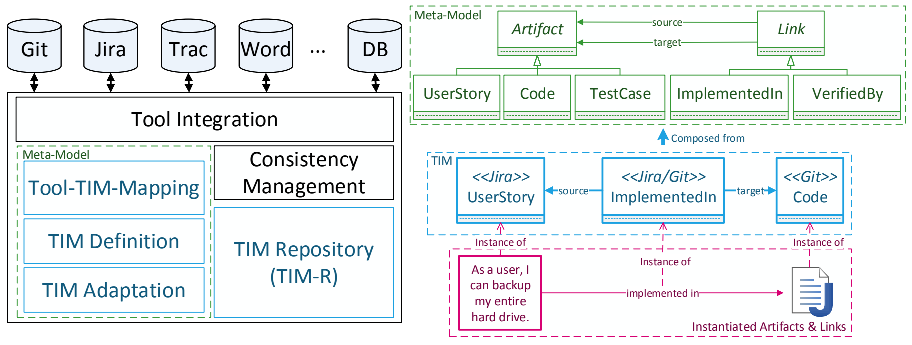

Fig. 1: Approach for adaptable traceability tool support

Our approach for adaptable traceability tool support shows in Fig. 1 on the left side the components involved and on the right side an example. The tool integration component is required for two reasons. (i) Artifacts and links can be stored in different tools, e.g. source code is stored in a version control system such as git[4] and requirements are stored in an issue tracking system such as Jira or Trac[5]. (ii) On TIM-adaptations, changes of the TIM must be populated to the tools, e.g. if an artifact type is added to a TIM, the artifact type is also added to the tools. The component for tool integration can be implemented using the Open Services for Lifecycle Collaboration[6]-specification. The meta-model component is used as a basis for TIM-definition and -adaptation (shown as green dashed box in the left part of Fig. 1). The green top right part shows the proposed comprehensive meta-model. Due to readability reasons, only parts of the meta-model are shown, e.g. artifact attributes are hidden and in the full version more artifact types and more link types are available. Traceability links between artifacts are not pre-defined and thus artifacts can be linked to each other by any kind of traceability link. This satisfies R1. In the TIM-definition component, the TIM is derived from the meta-model by selecting the artifacts and the links. For example, we can compose the TIM shown in the blue middle right part of Fig. 1. Artifacts and links can be only instantiated if they are defined within the TIM and have a tool mapping. The TIM-Tool-Mapping component

[4] *https://git-scm.com/*, last checked Jan 13, 2016
[5] *https://trac.edgewall.org/*, last checked Feb 23, 2016
[6] *http://open-services.net/*, last checked Feb 23, 2016

is used to define a mapping between tools, artifacts and links in the TIM. In our example TIM, the tool mapping is shown as stereotypes, e.g. Jira is used to manage *UserStories*, git is used for *Code* and a combination of Jira and git is used to link both artifacts. The instantiated artifacts and links (shown in the red right lower part of Fig. 1) are versioned within the tools. This satisfies R2. The TIM-adaptation component allows editing the defined TIM. The initial TIM and all versions of it which arise due to adaptations are versioned within a TIM-repository (TIM-R). Model repositories (MR) such as EMFStore[7] or AMOR[8] can be used to versioned TIMs, because MR are designed to handle model changes appropriately. Thus, R3 is satisfied.

In the following, possible changes resulting from a TIM-adaptation are discussed. If instantiated artifacts or links change, e.g. another link is added between *UserStory* and *Code*, the artifacts and links are updated in their respective tools. If a new artifact type is added to the TIM, e.g. add *TestCase* to example TIM, a new TIM-version is stored in the TIM-R and the new artifact type is populated to the tool. Similarly, for adding link types e.g. add link type *VerifiedBy* between *UserStory* and *TestCase*. If a link type is removed from the TIM, e.g. remove the *VerifiedBy* link, all existing links of that type should be also removed from the respective tools. The link type is kept in the tool to be able to revert to prior consistent states of artifacts and links. If an artifact type without link types is removed from the TIM, e.g. remove the *TestCase* artifact, all artifacts of this type are removed. As in the case before, the artifact type remains in the tool. If an artifact type with a link type is removed from the TIM, e.g. remove the *UserStory* artifact having the link type *ImplementedIn*, then all artifacts and corresponding links are removed, but the types are kept in the tools. In our meta-model, and thus, in a TIM in principle the source or the target artifact type of link types could be changed. This is handled by removing the link type and adding a new link type. Due to the applied TIM-versioning, no link-information is lost.

On TIM-adaptations affected artifacts and links must be retrieved and updated. For this, two solutions are possible. (i) The TIM-R could hold references between TIM-versions and the artifacts and links. In a TIM-version, each artifact type and each link type keep references to the corresponding artifacts and links. The references can be created during TIM-application, i.e. on instantiation and validation of artifacts and links according to the TIM. On TIM-adaptations, the references are used to update affected artifacts and links. (ii) Affected artifacts and links could be retrieved from their respective tools when needed. For each changing artifact type or link type all corresponding artifacts and links are retrieved. Thus, only artifacts and links of interest are retrieved and updated on TIM-adaptation. The TIM, artifacts and links are managed consistently for both solutions and thus R4 is satisfied. From the TIM-R, a graph can be generated containing e.g. two TIM-versions and corresponding artifacts for each version. This finally satisfies R5.

[7] *http://www.eclipse.org/emfstore/*, last checked Jan 13, 2016
[8] *http://www.modelversioning.org/index.php*, last checked Feb 23, 2016

5 Research Method

Design science is used as research method [18]. First, an analysis of the as-is-state in research and practice is done. Based on the as-is-state, theories for solution ideas can be implemented in a software prototype. To prove that the solution ideas and the prototype solve the problem, evaluation in practice is performed. Our as-is-study comprises a literature review and a series of interviews with experts from practice. The literature review is conducted as a systematic mapping study according to Kitchenham and Charters [10]. The research question to answer is: What approaches exist to adapt the traceability during software development projects (RQ1)? For the series of expert interviews, we use approximately 8 to 10 semi-structured interviews. The research question to answer is: What is the current state of practice wrt traceability-adaptation (RQ2)? During the interviews, open questions are used to elicit as much information as possible from the experts minimizing prior bias. We use an interview guideline to ensure that all relevant aspects are covered during the interviews. Within the evaluation, the overall goal is to validate the feasibility of the proposed solutions. For this, we build a prototypical tool realizing the solution ideas. The tool is validated by application in student theses and in practical courses. In addition, experts from practice evaluate our software prototype, in particular evaluation is planned with the experts we have interviewed to derive the requirements.

6 Progress

In 2015, we implemented the comprehensive meta-model in a prototypical tool based on UNICASE[9]. In its early state, our tool provides the ability to use a flexible TIM. We have started with the systematic mapping study in order to answer RQ1 using the search string: $traceability \wedge (adaptation \vee configuration \vee adjustment \vee modification \vee customization \vee tailoring \vee evolution)$. The search string has been used in 5 different sources. Overall 913 hits have been identified, the initial selection based on publication title and abstract lead to currently 30 relevant publications. This selection has then been reviewed with defined exclusion criteria. Moreover, we conducted two interviews with two experts settled in different domains to answer RQ2 [16]. In 2016, we first plan to finish the systematic mapping study. We also plan to conduct more expert interviews on the as-is-state in practice. From the interview results, we will be able to sharpen existing requirements. We will adapt and extend our implementation of the TIM-repository to manage different TIM-versions. In 2017, we want to conduct the case study for validation and expect to finish this thesis by December.

Acknowledgement. I would like to thank my supervisor Barbara Paech for support of this research.

[9] *http://unicase-ls1.github.io/unicase/*, last checked Jan 13, 2016

References

1. Aizenbud-Reshef, N., Nolan, B.T., Rubin, J., Shaham-Gafni, Y.: Model traceability. IBM Syst. J. 45(3), 515–526 (2006)
2. Badreddin, O., Sturm, A., Lethbridge, T.C.: Requirement Traceability: A Model-Based Approach. In: MoDRE14. pp. 87–91. IEEE, Karlskrona, Sweden (2014)
3. Boronat, A., Carsí, J., Ramos, I.: Automatic support for traceability in a generic model management framework. In: Model Driven Architecture – Foundations and Applications, LNCS, vol. 3748, pp. 316–330. Springer Berlin Heidelberg (2005)
4. Cleland-Huang, J., Gotel, O., Hayes, J.H., Mäder, P., Zisman, A.: Software Traceability: Trends and Future Directions. In: FOSE 2014 (ICSE 2014). pp. 55–69. ACM, Hyderabad, India (2014)
5. Cleland-Huang, J., Gotel, O., Zisman, A. (eds.): Software and Systems Traceability. Springer London, London, United Kingdom (2012)
6. De Lucia, A., Fasano, F., Oliveto, R., Tortora, G.: Fine-grained management of software artefacts: the adams system. Software: Practice and Experience 40(11), 1007–1034 (2010)
7. Espinoza, A., Garbajosa, J.: Tackling traceability challenges through modeling principles in methodologies underpinned by metamodels. CEE-SET WiP pp. 41–54 (2008)
8. Gotel, O., Cleland-Huang, J., Hayes, J., Zisman, A., Egyed, A., Grünbacher, P., Dekhtyar, A., Antoniol, G., Maletic, J.: The grand challenge of traceability (v1.0). In: Software and Systems Traceability, pp. 343–409. Springer London (2012)
9. Kelleher, J.: A reusable traceability framework using patterns. In: TEFSE 2005. pp. 50–55. ACM, New York, NY, USA (2005)
10. Kitchenham, B.A., Charters, S.: Guidelines for Performing Systematic Literature Reviews in Software Engineering (Version 2.3). Tech. Rep. EBSE 2007-001, Keele University; University of Durham, Keele, Staffs, UK; Durham, UK (2007)
11. Lindsay, P., Liu, Y., Traynor, O.: A generic model for fine grained configuration management including version control and traceability. In: ASWEC 1997. pp. 27–36 (1997)
12. Macfarlane, I., Reilly, I.: Requirements traceability in an integrated development environment. In: RE 1995. pp. 116–123 (1995)
13. Mäder, P., Gotel, O., Philippow, I.: Getting back to basics: Promoting the use of a traceability information model in practice. In: TEFSE 2009. pp. 21–25 (2009)
14. Mäder, P., Egyed, A.: Do developers benefit from requirements traceability when evolving and maintaining a software system? Empirical Software Engineering 20(2), 413–441 (2014)
15. Maletic, J.I., Collard, M.L., Simoes, B.: An xml based approach to support the evolution of model-to-model traceability links. In: TEFSE 2005. pp. 67–72. ACM, New York, NY, USA (2005)
16. Seiler, M., Kücherer, C., Paech, B.: Traceability Usage and Adaptation in Practice. Softwaretechnik-Trends (2016), (accepted to appear)
17. Taromirad, M., Paige, R.F.: Agile requirements traceability using domain-specific modelling languages. In: XM 2012. pp. 45–50. ACM, New York, NY, USA (2012)
18. Wieringa, R.J.: Design science methodology for information systems and software engineering. Springer Verlag, London (2014)
19. Winkler, S., Pilgrim, J.: A survey of traceability in requirements engineering and model-driven development. Software & Systems Modeling 9(4), 529–565 (2009)