

Towards Traceability from Project Management to System Models

Jonas Helming, Maximilian Koegel, Helmut Naughton
*Technische Universität München, Institut für Informatik
Chair for Applied Software Engineering
Boltzmannstrasse 3, D-85748 Garching, Germany
{helming, koegel, naughton}@in.tum.de*

Abstract

Traceability is commonly known as the ability to describe and follow links between artifacts, e.g. between requirements and their corresponding part of the system design. These links are typically inside of the system specification. Very few approaches consider traceability between the system specification and project management artifacts. By introducing links between these two models a task can be traced to the system model elements it is related to.

This paper proposes a unified model, which explicitly combines project management models and the system specification models to enable traceability. We introduce and discuss the following key concepts, which are currently evaluated in a case study: (1) Ability to navigate between tasks and the according part of the specification, (2) Project Management Status aggregation by system specification artifacts, (3) Use of entities from the system model for project planning, (4) Unified model validation.

1. Introduction

“Software artifact traceability is the ability to describe and follow the life of an artifact (requirements, code, tests, models, reports, plans, etc.) developed during the software lifecycle in both forward and backward directions (e.g. from requirements to the modules of the software architecture and the code components implementing them and vice-versa)” [1,2]. “Traceability can provide important insights into system development and evolution assisting in both top-down and bottom-up program comprehension, impact analysis, and reuse of existing software, thus giving essential support in understanding the relationships existing within and across software requirements, design, and implementation” [1,3]. This definition restricts the

term traceability to system model artifacts. These artifacts describe the system under construction on different levels of abstraction. Therefore we will call the entirety of such models system model. On the other hand, there are a lot of artifacts in a software project like tasks, schedules or the organizational structure, which do not describe the system. These artifacts describe the project itself, we will therefore call them project model.

Artifacts of the project model, i.e. planning and management artifacts or the organizational structure, are usually not traceable to their according part of the system model, i.e. the requirements and design artifacts. That is surprising as there are a lot of important relations between project management and the system model, which have a direct influence on the project. For example many analysis tasks are related to analysis artifacts of the system model, implementation task usually realize requirements, which are also system model elements. Starting from the project model, e.g. in case you work on a task, it would be useful to directly navigate from the task artifact to the corresponding part of the specification. Starting from the system model it is valuable information to know which work is yet to be done corresponding to a specific part of the system model. As an example the project manager could be interested in implementation tasks that are yet to be done to complete a certain requirement. That requires traceability from the system model to the corresponding part of the project model. Status aggregations over artifacts of the system model are especially useful in lifecycle models where you use system model elements for planning. An example is the Scrum [11] methodology, which uses “Backlog Items” to plan iterations. Backlog Items are usually requirements or scenarios (“User Stories”) and are part of the system model. Iterations are management artifacts and are part of the project model. Therefore integration between system model and project model can support the activity of planning itself. Finally by

integrating the project model with the system model we get to apply model validation techniques, which are usually only used in the system model to the entire unified model. That means we can check instances of our unified model against criteria such as consistency or integrity. To evaluate these concepts, we used a unified repository. The implementation is based on our experience with predecessor implementations. We are currently running a case study with 20 students to prove the feasibility and find possible impediments.

The paper is organized as follows: section 2 gives an overview over related work, section 3 describes the set-up of the ongoing case study and section 4 describes the key concepts which are currently evaluated.

2. Related Work

Several academic tools such as ADAMS [4] provide traceability between artifacts of the system model but not to the project model, since it is not captured in the system itself. Gelbard et al. showed 2002 [5] that there is hardly any integration between Project Management Tools and Computer Aided Software Engineering (CASE) Tools, although they claim this to be desirable. This missing integration between CASE tools and project management tools has been noted some time ago. In the book *“Principles of CASE Tool Integration”* [6] the authors state “Individual case tools that provide ‘islands of automation’, so they propose among others to integrate a bug tracking tool, which represents artifacts from the project model. In [7] Mi and Scacchi propose a “process driven CASE environment“ with “a higher level of integration, process integration, which represents development activities explicitly in a software process model to guide and coordinate development and to integrate tools and objects.” This provides the user of the system with a context for each task, providing links to resources like specifications and such. Jarzabek and Huang [8] claim that “future CASE tools should be based on sound models of a software process”, since their study showed that the issue of a developer having to perform a certain task, make a decision or meet a goal” and finding some information such as application domain model, program requirements, program design etc. missing should be addressed by the CASE tool itself. An IBM Research project called Integrated Solution Engineering [9] also acknowledges the problem and tries to help developers by offering semi-automated support for “capturing and mining relationships among artifacts and/or developer tasks”. They also do not model tasks explicitly. In the

industry, there exist solutions to bridge the gap somewhat, but they lack refinement. For example the integration between IBM Rational RequisitePro and Microsoft Project is difficult since the user has to sync the tools using a wizard and even has to map IDs manually by entering them with a text editor [10].

2. Unicase

Our goal is to research the benefits of integrating the project with the system model. To focus on this issue and to avoid additionally dealing with integration issues we have chosen Unicase, a unified platform, as an evaluation platform. Unicase [12] is a "CASE" (Computer Aided Software Engineering) tool for a "uni"-fied model, the successor system of Sysiphus [13]. Unicase provides a unified and flexible repository, which stores arbitrary types of artifacts. As these artifacts are always part of an entire unified model we will call them model elements. These model elements can be either from the system model of a project, i.e. the requirements, the specification in artifacts like UML [14], or from the project model, i.e. the organization, the project management or the rationale management. Model elements can be linked to each other (see Figure 1). These links can be inside the project model, e.g. a project participant can be linked to a task he is assigned to, or inside the system model, e.g. a functional requirement is linked to a detailing use case. But links can also be between system and project model, as an example a task from the project model can be linked to a part from the system model, e.g. a use case, it is referring to. The interrelations and traceability between these two models are the topic of this paper. In Unicase links and model elements can be flexibly defined in a meta-model, which can be modified, even during project run-time, without adapting the tool itself.

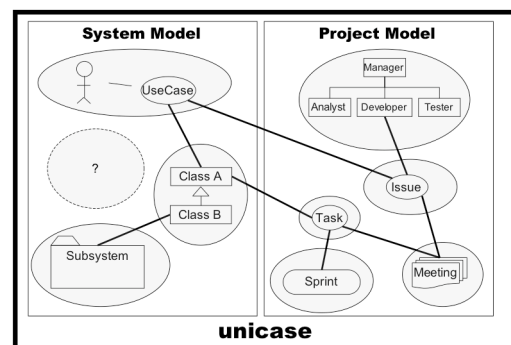


Figure 1: Unicase integrates the project model and the system model.

Instances of the unified model are collaboratively modifiable enabled by a changed-based versioning system [15]. Unibase provides several generic views (graphical and textual) to browse and modify model elements. Further it is easily extensible with new views, e.g. a Gantt-view.

The integrated model as well as the possibility to enhance the model and the corresponding views flexibly without caring about infrastructural parts of the system was the reason to choose Unibase as the platform to implement the concept of traceability between project and system model.

3. Case-study set-up

In this section we shortly describe the ongoing case study as well as the relevant part of the unified model, which is used in Unibase for the case study. The project used for the case study is a practical training of one semester duration. The project participants are about 20 students (low personal fluctuations) with a low to middle level of experience in the field of software engineering, organized in 6 teams. The scope is the development of a software system with several partly distributed components. The project involves the following software engineering activities: requirements elicitation, analysis, system design, object design, implementation, testing and project management. The project participants use Unibase as a central repository for system models and project models. Both, the system model and the project model were tailored to the specific requirements of the project. In this paper we will briefly describe a small part of the tailored system and project model in Unibase. The concepts in section 4 are applicable to the entire model without any modification. The complete model also contains elements such as subsystems or classes (see Figure 1), but a description would go beyond the scope of this paper. As shown in Figure 2 all parts of the system model are a subtype of SystemModelElement.

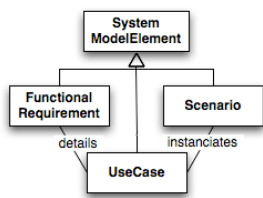


Figure 2: All parts of the system model are subtypes of SystemModelElement. They can be linked to each other by typed associations like "details"

System model elements can be linked to each other. In Figure 2 there are two examples for such links. First UseCases can be instantiated in Scenarios; second

FunctionalRequirements can be detailed in use cases. As shown in Figure 3 all system model elements can be annotated by WorkItems. Work items represent some kind of work, which needs to be done. There are several subtypes of work items. WorkPackage implements a composite pattern, which allows to aggregate work items, e.g. to model iterations. ActionItems, Issues and BugReports are basically different representations of tasks. An annotation to a system model element means, that the associated work is related to this element. As an example a work item, which is annotated to a functional requirement can describe an analysis issue, which refers to the description of the functional requirement, but also an implementation action item, which extends the system to fulfill this functional requirement. The activity of action items and Issues is determined by the corresponding attribute, bug reports always represent an implementation task. Work items can be assigned to OrgUnits, either users or whole teams.

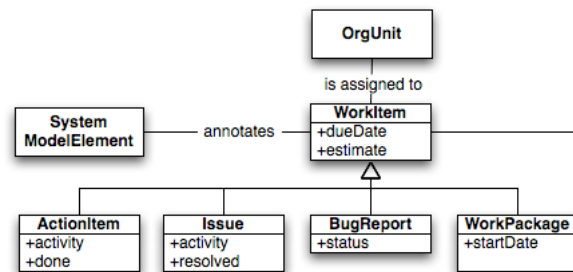


Figure 3: The common super type of project model elements defining work is WorkItem. A WorkItem can be assigned to one or more Project participant and annotated to a system model element.

4. Key Concepts

In this section we introduce the four key concepts, which have been implemented in Unibase and are currently evaluated in the ongoing case study. Some of these concepts were implemented in Sysiphus before, the Predecessor system of Unibase. We improved the implementation based on our experiences from the evaluation of Sysiphus.

4.1 Ability to navigate

The most evident benefit of links between project and system model elements is the ability to navigate between related artifacts from both models. The most common example is the navigation from a task to the corresponding part of the specification. For instance a developer uses this navigation to gather detailed information about his implementation task. As we described in section 3, work items are directly annotated to their corresponding part of the

specification, i.e. one or more system model elements. On one hand this can mean, that the work item is directly to be applied to the system model element, e.g. a task “refine use case” is annotated to the related use case. As the assignee of the task will have to open the system model element to complete the task, the annotation link is obviously a shortcut. On the other hand an annotation can mean the work item is part of the implementation of this part of the model. For instance an implementation task is annotated on a requirement to express that the task extends the system under construction to fulfill the requirement. In this case the developer, the task is assigned to, needs to be informed about the requirement and most likely navigate to it. In Unicase links such as annotations are displayed as hyperlinks and therefore easily to navigate. We implemented a usage data collection for the activation of these hyperlinks to evaluate how frequently they are executed. Further we will complement this result by asking the user about the benefit of the ability to navigate. As we know from our experiences with the predecessor system of Unicase, Sysiphus, this approach had basically two impediments: (1) Links were not entered correctly by the users, means many tasks were annotated not at all. This is especially true for implementation tasks (2) The tasks were often not the entry point for users to solve a task, cause the system was not integrated into their work environment. In Unicase we tried to solve these impediments as follows: First we improved the feature to link tasks to the corresponding model elements. Second we validate tasks, if they are linked correctly (see section 4.4). Thirdly we determined implementation task to be annotated to functional requirements. This constrains the number of system model elements, which can be annotated and therefore makes it easier for the users to find the right one. And finally we integrated Unicase in Eclipse, which automatically integrates the task list into the daily workbench of developers. As an effect we expect, that task are more often the entry point for project participants to solve tasks and therefore links more often to be navigated.

4.2 Status aggregation

For project participants it is often valuable information to gather which work is yet to be done on a specific part of the system model. For instance a manager wants to know, which implementation tasks are yet to be done, that the system under construction fulfills a functional requirement. In Unicase these tasks are linked to the corresponding part of the system model and therefore it is possible to automatically gather this information. In the system model, there are dependencies where system model elements require the

completion of an other model element to be completed. As an example (see Figure 4) for the completion of a certain scenario a number of use cases has to be also completed. In turn for the completion of use cases a certain number of functional requirements needs to be completed. In Unicase you can define links, which represent such a dependencies as so-called OpeningLinks [16]. Along these OpeningLinks Unicase can recursively collect all work items, which are required to be done to complete a certain system model element. Based on the status, the estimates and the due dates of these work items, we can calculate the status as well as an estimated completion date of a system model element.

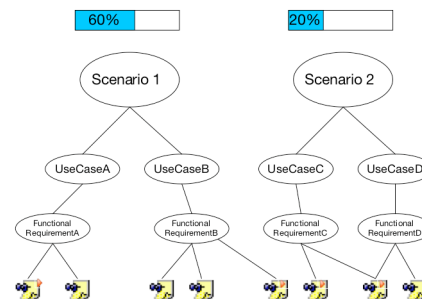


Figure 4: The status of system model elements can be aggregated along so-called Opening Links. In this example the status of the two scenarios is aggregated.

Furthermore we can show a list of required work items. In Unicase these recursively collected work items can be shown to the user in four different representations: (1) Flat View: this view shows a tabular list of all collected work items, (2) Hierarchical View: This view hierarchically groups the work items by the annotated system model elements. (3) User View: This view groups the work item by the users the work items are assigned to, (4) Activity View: This view groups the work items by the corresponding activity, e.g. implementation.

4.3 Planning using system models

The integration of project model and system model enables us to use system model elements for planning. As an example we can use Functional Requirements as so-called “Backlog Items” [4]. These Backlog Items are used to plan iterations. Assigning a Backlog Item to an iteration means that all corresponding work items have to be completed during this iteration.

To achieve this, Unicase provides a planning view, which shows all planned iterations. As a project manager you can assign work items to these iterations. But furthermore you can also assign parts of the system model, e.g. a Functional Requirement to an

iteration. In this case, all corresponding work items are also related to the iteration. As introduced in section 4.2 you can even recursively collect required work items using dependencies from the system model, e.g. if you relate a Scenario to a iteration. Similar to the different views provided for the status aggregation (see section 4.2.) Unicase can show the list of work items in different representation then just a flat list. Especially two of them are worth mentioning: (1) The Hierarchical View shows the system model elements in a hierarchical tree, where the work items are the leaf nodes (2) The User Views shows who is working on which tasks in the according iteration.

4.4 Unified model validation

As simple examples we introduced validation rules which check the tasks themselves, e.g. if they are annotated to model elements from the system model. Violations of these validation rules lead to a warning, which is visible to the user. Unicase tracks if a user follows a validation warning to solve the corresponding problem in the model. Thereby we evaluate the usefulness of specific validation rules. In this paper we will describe three example rules, which explicitly use the traceability from system to project model. (1) The first example rule checks if implementation tasks are annotated to functional requirements and therefore assures a complete integration between project model and system model. (2) The second example rule compares the priorities of Functional Requirements to the due dates of the annotated work items. Work items annotated to functional requirements with a higher priority must have earlier due dates. (3) The third example rule checks, if there is any analysis or design task related to a part of the system model, which have a later due date then a implementation task related to the same system model elements. Usually at least the analysis and design of the relevant part the system model should be completed before it is implemented, even in agile methodologies.

6. References

- [1] A.D. Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Recovering traceability links in software artifact management systems using information retrieval methods," *ACM Trans. Softw. Eng. Methodol.*, vol. 16, 2007, S. 13.
- [2] O. Gotel und C. Finkelstein, "An analysis of the requirements traceability problem," *Requirements Engineering*, 1994, Proceedings of the First International Conference on, 1994, S. 94-101.
- [3] J. D. PALMER, 2000, "Traceability", In *Software Requirements Engineering*, Second Edition, R. H. Thayer and M. Dorfman, Eds. IEEE Computer Society Press, 2000, Los Alamitos, CA, 412-422.
- [4] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Recovering traceability links in software artifact management systems using information retrieval methods," *ACM Trans. Softw. Eng. Methodol.*, vol. 16, 2007
- [5] R. Gelbard, N. Pliskin, I. Spiegler, "Integrating system analysis and project management tools", *International Journal of Project Management*, vol. 20, 2002, 461-468
- [6] S. Jarzabek and R. Huang, "The case for user-centered CASE tools," *Commun. ACM*, vol. 41, 1998, pp. 93-99.
- [7] P. Mi and W. Scacchi, "Process integration in CASE environments," *Software, IEEE*, vol. 9, 1992, pp. 45-53.
- [8] S. Jarzabek and R. Huang, "The case for user-centered CASE tools," *Commun. ACM*, vol. 41, 1998, pp. 93-99.
- [9] L. Gong, T. Klinger, P. Matchen, P. Tarr, R. Uceda-Sosa, A. Ying, J. Xu, and X. Zhou, "Integrated solution engineering," *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, Portland, Oregon, USA: ACM, 2006, pp. 726-727.
- [10] "IBM Rational RequisitePro Help," Integrating with Microsoft project, http://publib.boulder.ibm.com/infocenter/reqpro/v7r1m0/index.jsp?topic=/com.ibm.reqpro.help/integ/ms_project_integ/t_integ_ms_proj.html, 2009.
- [11] K. Schwaber. *Agile Project Management with Scrum*. Microsoft Press, 2004.
- [12] B. Bruegge, O. Creighton, Jonas Helming, Maximilian Koegel, Unicase - an Ecosystem for Unified Software, In *ICGSE '08: Distributed software development [Electronic Resource] : methods and tools for risk management ; ICGSE Workshop 2008 (Bangalore, India, 2008)*.
- [13] T. Wolf. *Rationale-based unified software engineering model*. In *Dissertation, Technische Universitaet Muenchen*, 2007.
- [14] Omg unified modeling language specification version 2.0., 2004.
- [15] M. Kögel, Towards software configuration management for unified models. In *ICSE '08, CVSM '08: Proceedings of the 2008 international workshop on Comparison and versioning of software models (New York, NY, USA, 2008)*, Technische Universität München, ACM, pp. 19-24.
- [16] J. Helming, Integrating Software Lifecycle Models into a uniform Software Engineering Model. In *Workshop Proceedings of Software Engineering Conference 2008, Munich, Germany*