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 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

Traceability in the context of software engineering is the ability to describe and follow relationships within requirements, design, and implementation artifacts. These relationships can be used for top-down and bottom-up program comprehension, impact analysis, and reuse of existing software [1,2,3]. 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 the 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 and we will therefore call them the project model.

Artifacts of the project model, e.g. planning and management artifacts or the organizational structure, are usually not traceable to their according parts of the system model, e.g. requirements and design artifacts. That is surprising as there are a lot of important relations between project management and the system model, which have directly influence 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, a developer working on a task could use information on how to directly navigate from the task to the corresponding part of the specification. Starting from the system model it is valuable information to know how much work remains to be done in order to complete a specific part of the system model. For example the project manager could be interested in implementation tasks that remain to be done in order to complete a certain requirement. That requires traceability from the system model to the corresponding part of the project model. Status aggregation over artifacts of the system model is especially useful in lifecycle models where system model elements are used for planning. An example is the Scrum [15] 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 the unified model against criteria such as consistency or integrity. To evaluate these concepts, we used a unified repository. 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 Unicase, the platform our approach is based on, Section 4 describes the set-up of the ongoing case study and Section 5 describes the key concepts which are currently being evaluated.

2. Related Work

Several academic tools such as ADAMS [1] provide traceability between artifacts, usually of the system model. ADAMS could be extended to capture the project model just like Unicase (see Section 3). In 2002 Gelbard et al. [4] showed that there is hardly any integration between Project Management Tools and Computer Aided Software Engineering (CASE) Tools, although they claim this would 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*" [5] the authors state that individual case tools provide 'islands of automation', so they propose among others to integrate a bug-tracking tool, which represents artifacts from the project model. In [6] 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 [7] 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 [8] 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 [9]. The bug-tracking system JIRA [10] by Atlassian offers planning support for agile development using tasks, but traceability for instance to requirements is not yet supported [11], making the integration of project and system model incomplete. Microsoft Team Foundation Server [12] offers some integration between the project and system model, but in form of links to entire documents, not on the basis of single artifacts. This makes analyses on artifact level much more costly. Sparx Enterprise Architect [13] in addition to its UML and modeling capabilities offers support for requirements management and some parts of project management. But it lacks tailored support for various software engineering methodologies, leaving requirement as the only basis for such planning. ReqAnalyst by Lormans et al. [14] uses requirements as basis for coverage and status views and provides traceability based on link recovery using latent semantic indexing. They show how much information can be obtained by exploiting the recovered links and how useful views based on traceability can be.

3. Unicase

Our goal is to research the benefits of integrating project and system model. To focus on this issue and to avoid additionally having to deal with integration issues we chose Unicase as evaluation platform. Unicase [16], the successor system of Sysiphus [17], is a CASE (Computer Aided Software Engineering) tool for a unified model. Unicase provides a unified and flexible repository, able to store 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 from the system model as well as the project model and can be linked to each other (see Figure 1).

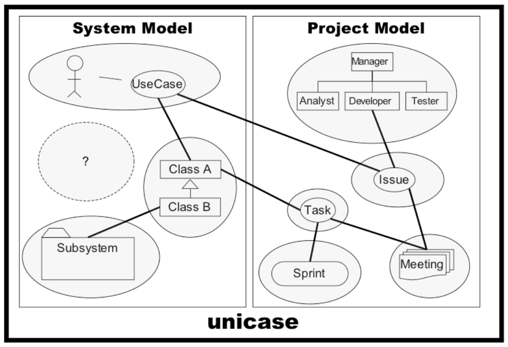

Figure 1: Unicase integrates project model and system model.

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, for example a task can be linked to a use case. 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.
Instances of the unified model can be collaboratively modified using a changed-based versioning system [18]. Unicase provides several generic views (graphical and textual) to browse and modify model elements. It is easily extensible by 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 Unicase as basis of our implementation.

4. Case-study set-up

In this section we shortly describe the ongoing case study as well as the parts of the unified model relevant to the study. The project examined in the case study is a one semester practical course in software engineering. The project participants are about 20 students (low personal fluctuations) with a low to medium 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 participants use Unicase as 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. We will briefly describe a small part of the tailored system and project model in Unicase. The concepts in section 5 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 be beyond the scope of this paper.

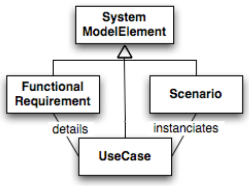

Figure 2: All parts of the system model are subtypes of SystemModelElement. The can be linked to each other by typed associations like "details"

As shown in Figure 2 all parts of the system model are subtypes of SystemModelElement. 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 Functional Requirements 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 aggregation of 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. For 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. Work items can be assigned to OrgUnits, either users or whole teams.

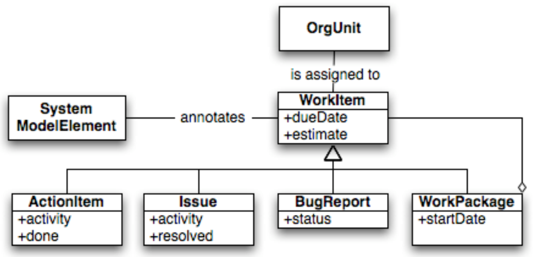

Figure 3: The common super type of project model elements defining work is WorkItem. A WorkItem can be assigned to one or more project participants and can be annotated to a system model element.

5. Key Concepts

In this section we introduce the four key concepts, which have been implemented in Unicase and are currently evaluated in the ongoing case study. Some of these concepts were implemented in Sysiphus before, but were improved based on our experiences from previous evaluations.

5.1 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. As we described in Section 4 work items are directly annotated to their corresponding part of the specification, i.e. one or more system model elements. On the one hand this can signify that the work item is to be directly 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 imply 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 task's assignee needs to be informed about the requirement and will most likely navigate to it. In Unicase links such as annotations are displayed as hyperlinks and are therefore easy 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 Sysiphus, the predecessor system of Unicase, this approach had basically two impediments: (1) Links were not entered correctly by the users, i.e. many tasks were annotated not at all. This is especially true for implementation tasks. (2) The tasks themselves were often not the entry point for users intending to solve a task, because the system was not integrated into their work environment. In Unicase we tried to solve these impediments as follows: First we improved the feature allowing to link tasks to the corresponding model elements. Second we validate tasks, i.e. we check if they are linked correctly (see section 5.4). And finally we integrated Unicase in Eclipse, which automatically integrates the task list into the daily workbench of developers. We therefore expect tasks to be used as entry points more frequently and therefore links to be navigated more often.

5.2 Status aggregation

For project participants it is often valuable information, which work remains to be done on a specific part of the system model. A manager may want to know, which implementation tasks prevent the system under construction from fulfilling a functional requirement. In Unicase such 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 another model element to be completed. For example (see Figure 4) for the completion of a certain scenario a number of use cases also need to be completed. In turn for the completion of use cases a certain number of functional requirements needs to be completed. Unicase defines links, which represent such a dependencies as so-called OpeningLinks [19]. 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.

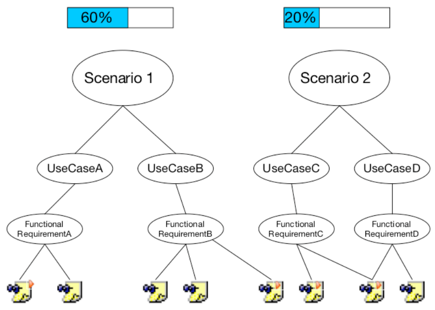

Figure 4: The status of system model elements can be aggregated along so-called Opening Links. In this example the status of the two scenarios is aggregated.

Furthermore we can show a list of required work items. In Unicase these recursively collected work items is presented to the user in four different ways: (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 work items by the users they are assigned to, (4) Activity View: This view groups work items by the corresponding activity, e.g. implementation.

5.3 Planning using system models

The integration of project model and system model enables us to use system model elements for planning. For instance we can use Functional Requirements as so-called "Backlog Items" [15]. These Backlog Items are used to plan iterations. Assigning a Backlog Item to an iteration signifies that all corresponding work items have to be completed during this iteration.

To archive 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 (cf.

Section 4.2.) Unicase can show the list of work items in different representations then just a flat list.

5.4 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 warnings, which are 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 the following we will describe three exemplary rules, which explicitly make use of the traceability from system to project model. (1) The first rule checks if implementation tasks are annotated to functional requirements and therefore assures a complete integration between project model and system model. (2) The 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 rule checks if there is any analysis or design task related to a part of the system model, which has a later due date then an 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. Conclusion

In this paper we described an approach explicitly combining project model and system model. We introduced four concepts based on this integration, which are currently evaluated in a case study. So far the results of the case study look promising. Validation rules and the integration in Eclipse lead to highly maintained links. Concept two and three will be evaluated during planning of two iterations at the end of the case study. After the case study we plan experiments, which compare the benefit of the key concepts with the effort for maintaining the corresponding links.

7. References

[1] A.D. Lucia, F. Fasano, R. Oliveto, und 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] R. Gelbard, N. Pliskin, I. Spiegler, “Integrating system analysis and project management tools”, International Journal of Project Management, vol. 20, 2002, 461-468

[5] A.W. Brown, D.J. Carney, E.J. Morris, D.B. Smith, and P.F. Zarrella, Principles of CASE Tool Integration, 1994.

[6] P. Mi and W. Scacchi, “Process integration in CASE environments,” Software, IEEE, vol. 9, 1992, pp. 45-53.

[7] S. Jarzabek and R. Huang, “The case for user-centered CASE tools,” Commun. ACM, vol. 41, 1998, pp. 93-99.

[8] L. Gong, T. Klinger, P. Matchen, P. Tarr, R. Uceda-Sosa, A. Ying, J. Xu, and X. Zhou, “Integrated solution engineering,” 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, Portland, Oregon, USA: ACM, 2006, pp. 726-727.

[9] “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.

[10] http://www.atlassian.com/software/jira/

[11] http://jira.atlassian.com/browse/JRA-10425

[12] http://msdn.microsoft.com/en-us/teamsystem/

[13] http://www.sparxsystems.com.au/

[14] Marco Lormans, Hans-Gerhard Gross, Arie van Deursen, Rini van Solingen, and André Stehouwer, “Monitoring Requirements Coverage using Reconstructed Views: An Industrial Case Study,”, 13th Working Conference on Reverse Engineering, 2006, pp. 275-284.

[15] K. Schwaber. Agile Project Management with Scrum. Microsoft Press, 2004.

[16] 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).

[17] T. Wolf. Rationale-based unified software engineering model. In Dissertation, Technische Universitaet Muenchen, 2007.

[18] M. Kögel, Towards software configuration management for unified models. In ICSE ’08, CVSM ’08: Proceedings of the international workshop on Comparison and versioning of software models (New York, 2008), ACM, pp. 19–24.

[19] J. Helming, Integrating Software Lifecycle Models into a uniform Software Engineering Model. In Workshop Proceedings of Software Engineering Conference 2008, Munich, Germany