

# Integrating System Modeling with Project Management – a Case Study

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

## Abstract

*In software engineering projects, there are two different types of modeling tools. On the one hand, there are tools to model the system under construction, i.e. tools for requirements or UML tools for design. On the other hand, there are tools to model the project itself such as project management tools, bug trackers, or tools to model the organizational structure. In typical software engineering projects, the artifacts managed using these two types of tools are not integrated with each other. This makes traceability, e.g. from a task to the according part of the specification, rather difficult. Furthermore, entities belonging to the system model, for instance requirements, have to be modeled redundantly in management tools if they are used for planning purposes. We also claim that specifications are more likely to be kept up-to-date if they are integrated with frequently visited views such as task lists.*

*This paper presents a case study, in which the system under construction as well as the project management was modeled with a single unified tool. The paper focuses on how this approach can solve the above-mentioned issues. We present additional benefits, unsolved issues, and chances for future research using such an approach.*

## 1. Introduction

The lifecycle of a software development project is usually supported by various tools. These tools range from requirements elicitation and analysis tools to system design tools and tools to support the implementation and testing activity. This category of tools is known as Computer Aided Software Engineering (CASE) tools. A lot of work in the field of tool integration has been done in the last years to

produce an integrated and traceable model (e.g. [1]). The models these tools are based on describe the system under construction on different levels of abstraction. Therefore we call the entirety of such models the system model.

Parallel to the system model, various tools are used to model tasks, schedules, and the organizational structure of a project. Examples for such tools are bug trackers, task repositories, or project management tools such as Microsoft Project. Like system modeling tools, a lot of these tools offer the ability to be integrated with each other [10]. All of these tools are used to describe the project itself. Analogously, we call the entirety of the models forming the basis of these tools the project model.

Although tool integration inside the system model or inside the project model is not very common yet, there are at least some approaches available as stated before. In contrast, there is hardly any integration between CASE tools (system model) and project management tools (project model) [2]. This is quite surprising, since there are a lot of meaningful relations between the project model and the system model, which have a direct impact on the project.

In this paper, we propose an approach in which the system model and the project model are jointly modeled by means of a single unified tool. Thus both models are integrated from the beginning of a project. In particular, it is possible to relate artifacts from the project model to artifacts from the system model and vice versa. We conducted a case study based on a project with over 40 participants using the unified tool. The goal was to show the advantages of such an integrated approach, but also to identify obstacles and open questions for future research. In this paper, we focus on the evaluation of the following key concepts:

- (1) Ability to navigate: The most evident concept,

which relies on traceability between project model and system model, is the ability to navigate from a task to the according part of the specification. For instance, a developer should be able to navigate from an assigned implementation task to the according requirement. This allows quick referencing of documentation relevant to the task.

(2) Up-to-date documentation: While out-dated documentation can still be relevant, up-to-dateness is the second most important attribute for documentation besides correctness, as shown by Forward [3]. He suggests to support project participants in keeping documentation up-to-date. On the other hand, artifacts such as tasks, which are considered by the user on a daily basis, are automatically updated to a certain extent. We claim that links between these daily-used artifacts of the project model and artifacts of the system model lead to a higher level of actuality for documentation. Again, a developer who navigates from his tasks to the according requirements may correct out-dated information accordingly.

(3) Planning with system models: Several methodologies, especially agile ones, rely on artifacts of the system model for planning purposes. These system model artifacts are indirectly used to specify tasks for developers. An example is the Scrum [4] methodology, which uses “backlog items” to plan iterations. Backlog items are usually requirements or scenarios (“user stories”) and are part of the system model. By contrast, tasks referencing backlog items represent management artifacts and thus are part of the project model. Therefore, integration between system model and project model can be used to support the activity of planning. Otherwise, backlog items would have to be modeled redundantly in the system model and the project model, which leads to inconsistency in most cases.

Section 2 gives an overview over related approaches that integrate the system model with the project model. Section 3 describes the prerequisites for the case-study, the implementation of the concepts and the set-up of the case-study. In section 4-6 we evaluate the key concepts. Finally, we present a conclusion in Section 7.

## 2. Related Work

The lack of integration between CASE tools and project management tools has been looked into by various researchers. In the book “Principles of CASE Tool Integration” [5], the authors state “Individual case tools that provide ‘islands of automation’ are not sufficient to realize this goal [producing complex

software systems of high quality, on time to budget and with adequate documentation]”. Amongst others, they propose 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 called process integration, which explicitly represents development activities 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, recommending links to resources like specifications and such.

Jarzabek and Huang [7] claim “future CASE tools should be based on sound models of a software process”. They support this with a study showing that the CASE tool itself should help a developer having to perform a certain task, making a decision, or meeting a goal by finding the relevant information. This information should come from diverse sources such as application domain model, program requirements, program design etc.

Ramesh and Jarke [8] demonstrate the need for capture of dependency links. In particular, they identify task dependencies, that is “the dependency between objects created by their common reliance on a task [...]. Design objects (say, the ERD [Entity Relationship Diagram] and DFD [Data Flow Diagram]) often have task dependencies.” In their system though, tasks remain implicit and are only used for linking purposes.

In 2002, Gelbard et al. [2] claimed that there is a low degree of integration between classical CASE tools and project management tools. Information such as specifications is rarely used for project management purposes, even though “this information is valuable for project managers”. They try to bridge the gap between CASE and Project Management tools, but limit themselves to a subset of all available software engineering artifacts. Nevertheless they state that applying such an integrating layer in software development projects enables improved assessment of such essential elements as development duration, development cost, response time, and data traffic volumes.

The IBM Research project Integrated Solution Engineering [9] acknowledges this problem, too, and tries to help developers by offering semi-automated support for “capturing and mining relationships among artifacts and/or developer tasks”. They do not model tasks explicitly, either.

In the industry, there are 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 synchronize the tools using a wizard and even has to map IDs manually by entering them with a text editor [10].

Academic tools such as ADAMS [11] provide traceability between artifacts created during the software engineering process, but does not include the tasks that created them, since they are not captured by the system.

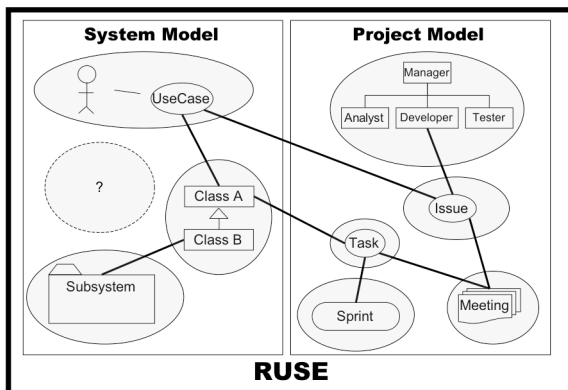
The tool Sysiphus [12], developed at our chair, provided links between tasks and other software engineering artifacts almost from the outset, but its capabilities were heavily extended for this study.

### 3. Prerequisites

Our approach is based on a unified model called Rational-based Unified Software Engineering model (RUSE) [13]. This model allows modeling and linking artifacts from both the project and the system model. It is implemented in a tool-suite called Sysiphus [12]. To evaluate our approach, we used the Project Analyzer for a Unified Software Engineering model (PAUSE) [14]. The following sections describe RUSE, Sysiphus, and the PAUSE approach in greater detail.

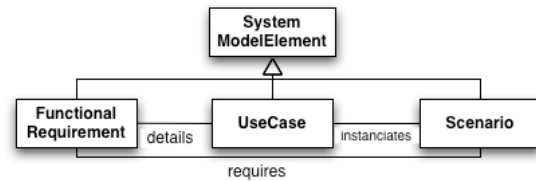
#### 3.1 RUSE and Sysiphus

The Rational-based Unified Software Engineering model (RUSE) [13] supports distributed software engineering projects in system modeling, collaboration, and management. The tool Sysiphus provides RUSE with a unified and flexible repository to store arbitrary types of artifacts. As these artifacts are always part of an entire unified model, we will call them *model elements*.



**Figure 1:** RUSE integrates the project model and the system model.

These model elements can either be part of the system model, i.e. requirements, the specification in artifacts like UML [15], or part of the project model, i.e. the organization, project management, or rationale management. Model elements can be linked to each other (see Figure 1). These links can be within the project model, e.g. a project participant can be linked to a task assigned to him, or within the system model, e.g. a functional requirement linked to a detailing use case. But links may also point from the system model to the project model, for example a task from the project model can be linked to the corresponding part of the system model, e.g. a use case. This paper mainly focuses on these interrelations and traceability between both models. In Sysiphus, links and model elements can be flexibly defined in a meta-model, which can also be adapted, even during project run-time, without modifying the tool itself. Furthermore, Sysiphus provides online and offline collaboration support for distributed teams. Project participants are able to browse and modify all model elements either in generic or in explicitly adapted views. For instance, to modify UML model elements, Sysiphus provides a graphical drawing pane. To browse ones daily tasks, the tool provides a tabular task list. All changes to the model are recorded by a change-based versioning system [16] (cf. Section 3.2).

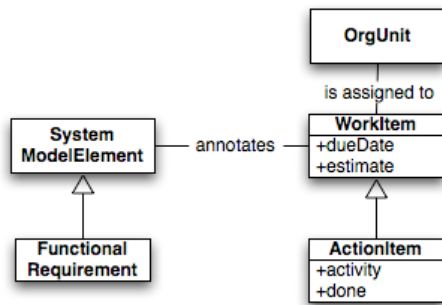


**Figure 2:** All elements of the system model are SystemModelElements. They can be linked by specific links.

Since this paper focuses on the integration between project model and system model, we consider this aspect of RUSE in greater detail. In our case study, RUSE was mainly used to model the system under construction on a high level of abstraction, i.e. on the level of requirements. Figure 2 shows the most important system model elements of RUSE, which are used to model requirements and the respective links. First, *UseCases* can be instantiated in *Scenarios*; second, *FunctionalRequirements* can be detailed in *UseCases*. The third type of link determines which *FunctionalRequirement* is required for a specific *Scenario*.

As shown in Figure 3, all system model elements

can be annotated by *WorkItems*. *WorkItems* represent some item of work, which needs to be done. There are several subtypes of *WorkItem* such as *ActionItem* or *Issue*.



**Figure 3:** Work items can be annotated to system model elements. For our evaluation, we focus on functional requirements and action items.

A system model element annotated with a *WorkItem* signifies that the associated work is related to this element. For example, a *WorkItem*, which is annotated to a *FunctionalRequirement* may describe an analysis task, which refers to the description of the functional requirement, but also an implementation *ActionItem*, which extends the system to fulfill this functional requirement. The activity the *ActionItems* and *Issues* belong to is captured by the corresponding attribute. *WorkItems* can be assigned to *OrgUnits*, which represent either users or whole teams. Without loss of generality, in our case study we focus on the annotation links between *ActionItems* and *FunctionalRequirements*. In the case study project we analyzed in detail, these two elements were most commonly used to describe the system model and the project model, respectively.

### 3.2 The PAUSE Approach

For our evaluation, we required information about the project model at different points in time. The Project Analyzer for Unified Software Engineering models (PAUSE) [14] builds on the Software Configuration Management (SCM) approach presented in [16]. This type of configuration management allows fine-grained change tracking on the level of model elements in the unified model, as opposed to the usual coarser “diffing approach”, which operates on the level of lines and files. The PAUSE approach does not only provide the states of the model elements, but also the changes that occurred between those states. This can be of interest in cases where one change masks another. In

these cases, change tracking logs both changes, while diffing will only keep the last change - in the database domain, this is called a “lost update”. Thus, our SCM system allows for retrieval of a specific project state and of a bundle of changes that occurred between the previous and the latter state. These changes are recorded on the attribute level of the model, e.g. changing the value of the attribute *DueDate* of a *Task*, providing a rich source for project analysis. In addition to these model-oriented changes, Sysiphus also records other change events such as accessing model elements (a change to the “read-time” attribute of a model element), capturing information beyond the content of model elements itself.

To conveniently access all data stored by Sysiphus, the PAUSE framework was developed. Using the SCM’s ability to recreate project states and the corresponding changes, it enables the user to quickly navigate the project history in specified steps, e.g. by version or by day. For each such step, the user of the framework can specify analyses to be run on the returned project data. The results of these analyses are then recorded in tabular form, reporting the statistics for the current day, which can be further evaluated in Microsoft Excel or statistical programs.

### 3.3 The Case Study

We evaluated our integration approach in a case study involving about 45 (low personal fluctuations) developers with a low to middle level of experience in the field of software engineering, organized in 11 teams. These teams worked for five months on a project in cooperation with an industrial partner. The scope of the project was the development of a software system with several partly distributed components. The project involved the following software engineering activities: requirements elicitation, analysis, system design, object design, implementation, and testing. All modeling activities were performed using the Sysiphus tool [12]. At the end of the project, the system contained about 15.000 model elements and showed a history of over 53.000 versions.

Due to the partly low experience level of the project participants, activities were executed sequentially in the first phase of the project. Each activity was officially concluded by an according review presentation. It was not expected to entirely fulfill the requirements after this first phase. Therefore, the participants worked for two more weeks to accomplish the most highly prioritized requirements in a second, more agile and more intensive phase.

## 4. Ability to Navigate

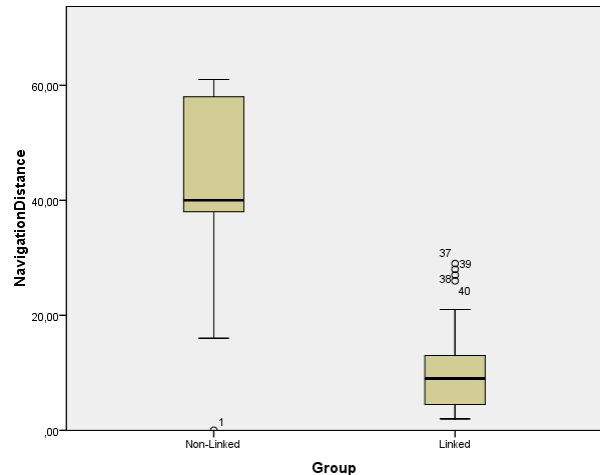
The most evident benefit of linking tasks to the according parts of the specification, in our case tasks to functional requirements, is the ability to navigate from a task to the part of the specification which describes the objective of the task in detail. A developer can use this navigation ability to gather detailed information about his implementation task. The crucial question we try to answer in this section is, whether or not model links really support this form of information retrieval.

As stated before, in this case study, we focus on the model link from action items to functional requirements. A large number of elements of both types were created and used during the project, which allows us to statistically analyze the navigation issue. For this purpose, we randomly created two groups of links between action items and functional requirements: The first group contained model links, which were already present in the project model and were thus potentially used for navigation by project participants. The second group was not existent during the project, but was created by an experienced project participant post-mortem. This means the project participant provided a group of links, which were missing in the model, but which he specified to be correct and meaningful. The project participants had no possibility to use this second group of model links during the project. Nevertheless, they could have navigated from an action item to the corresponding functional requirement using another way than a direct model link. Based on these two groups, we observed user behavior by analyzing the recorded read events. To detect navigation between an action item and a functional requirement, we first searched for a read event related to an action item and a subsequent read event concerning a functional requirement in the same session of a user. Furthermore, to assure temporal coherence, we only counted pairs of events which happened within a session timeout of 10 minutes.

In order to determine whether the project participants actually navigated linked artifacts as well as non-linked artifacts, we conducted a statistical test concerning the number of navigation activities in both groups (30 test instances for linked (*ActionItem,Requirement*)-pairs and 21 instances for non-linked pairs). The non-parametric Mann-Whitney-U test revealed that the null-hypothesis of equal mean values in the number of navigations must not be rejected, since one would risk to commit an alpha error with 28.1% probability ( $Z$ -value of  $-1.078$ ,  $\Phi(-1.078) +$

$(1-\Phi(1.078)) = 2 \Phi(-1.078) \approx 0.281$ <sup>1</sup>). From this result, we conclude that there is no significant difference between the numbers of navigations in both groups. That is, as long as action items and requirements stand in a meaningful relation, users navigate between them, even when there are no explicit links tying them together.

Subsequently, we measured the navigation distances between the action item and the respective requirement, that is, the number of read events in-between them. Even though the handling of navigation in Sysphus can cause more than one event for a navigated link this number is proportional to the number of navigated elements. Therefore the number of read events is on average a significant measure for the navigation distance. We divided these navigation pairs into two groups – those whose requirement was annotated by at least one action item ( $n_1=43$  items) and those, which were not ( $n_2=13$  items). Our hypothesis was that the means of the number of navigations differ significantly in both groups, since we measured  $m_{\text{annot}} = 10.26$  and  $m_{\text{non-annot}} = 40.69$ , respectively (see also Figure 4). Assuming a normal distribution of both mean values, we applied the t-test for independent samples to check the hypothesis of equal mean values. The 99%-confidence interval for the difference of the mean values  $m_{\text{annot}} - m_{\text{non-annot}} = 30.44$  is  $[15.42, 45.45]$ . The t-test returned a T-value of 6.083, which means that the critical value  $c(k=50, \alpha=1\%) = 2.40$  for  $n_1+n_2-2 = 54 \approx k$  degrees of freedom (rounded off) is exceeded and thus the null-hypothesis has to be rejected on the 1% level of significance.



**Figure 4:** Boxplot of the navigation distances for linked and non-linked requirements.

<sup>1</sup>  $\Phi$  is the standard-normal distribution function (cdf).

Thus we gained certain statistical significance for our claim that introducing direct links between frequently navigated pairs of model elements can shorten the navigation distances and can help the developer to find required information more quickly.

## 5. Up-to-date Documentation

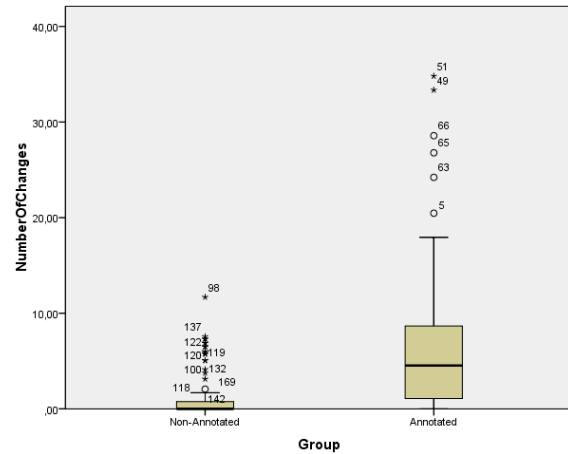
As shown in [3], up-to-dateness is the second most important attribute for documentation, e.g. requirements. By means of tools such as Sysiphus, one can observe that system model elements, which are not assigned to a specific person, are consulted less frequently by project participants than project model elements, which are assigned to a specific person. These project model elements usually represent tasks, which are accessible through a personal task list. We claim that links between these daily-considered artifacts from the project model and artifacts of the system model lead to more up-to-date documentation. For instance, a developer who navigates from his tasks to the according requirements is able to recognize outdated information and update the corresponding requirements. 65% of the project participants confirmed this assumption in a survey. To substantiate this claim, we carried out two different kinds of analyses. The first was based on the change history of the according system model elements (Section 5.1). The second was based on an interview with the project lead, who also acted as proxy client (Section 5.2).

### 5.1 Change Data Analysis

Again, we divided the requirements into two groups – those annotated by at least one action item ( $n_1=76$  items) and those being non-annotated ( $n_2=89$  items). As in the case of navigation distances, our hypothesis was that the means of the number of requirement changes differ significantly in both groups, since we measured  $m_{\text{annot}} = 6.59$  and  $m_{\text{non-ann}} = 1.30$ , respectively (see also figure 5). Assuming a normal distribution of both mean values, we again applied the t-test for independent samples to check the hypothesis of equal mean values. The 99%-confidence interval for the difference of the mean values  $m_{\text{annot}} - m_{\text{non-ann}} = 5.29$  is [2.88, 7.70].

The variances of both groups cannot be assumed to be equal (F-value = 31.76, significance for equal variances  $\approx 0.00$  (Levene test)). Based on this empirical fact, the t-test returned a T-value of 5.77, which means the critical value  $c(k=150, \alpha=1\%) = 2.35$  for  $n_1+n_2-2 = 163 \approx k$  degrees of freedom is exceeded

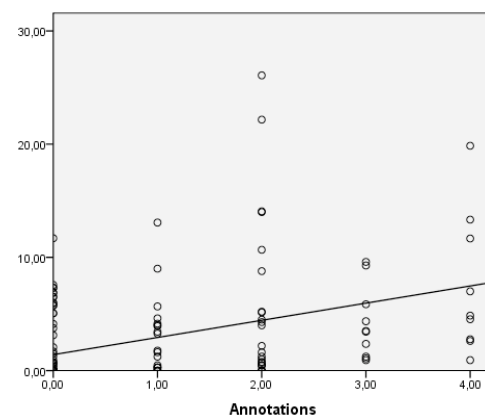
and thus the null-hypothesis has to be rejected on the 1% level of significance.



**Figure 5:** Boxplot of the number of changes for annotated and non-annotated requirements.

Even if one argues that the number of changes in each group is not normally distributed, the non-parametric Mann-Whitney-U test succeeds in rejecting the null-hypothesis of equal means. This test results a Z-value of -7.53 (standard-normal approximation<sup>2</sup>), which makes the probability to erroneously reject the null-hypothesis very small:  $2 \Phi(-7.53) \approx 0.00$ .

Independently from the question of different mean values, we analyzed the correlation between the number of annotations and the number of requirement changes. Both features show a positive correlation of  $\rho=0.423$  (Pearson coefficient, value space [0,1]), which is visually confirmed by the best-fit line in Figure 6.

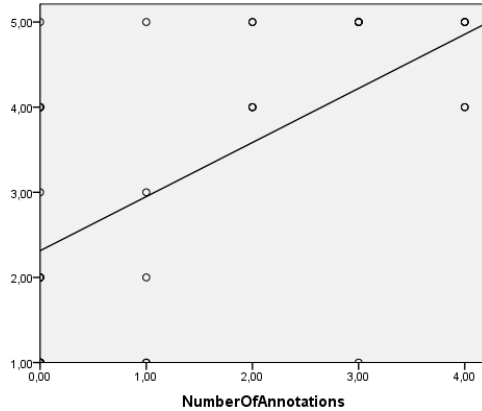


**Figure 6:** Number of requirement changes against number of annotations.

<sup>2</sup> Due to missing tabular values for U in the case  $k > 20$ .

## 5.2 Interview Analysis

We analyzed the correlation between the number of annotations (0 to 4) and the up-to-dateness of 44 annotated requirements, which was assessed by the project lead of the project.



**Figure 7:** Up-to-dateness of requirements against number of annotations. Multiple instances of the same (x,y)-pair appear as a single point.

The up-to-dateness was modeled as an ordinal variable  $U=\{1,2,\dots,5\}$ , where 5 represents the highest level of actuality. Both features show a positive correlation of  $\rho=0.582$  (Pearson coeff.) on the 1% level of significance.

Consequently, the number of annotations affects the up-to-dateness of requirements to a certain, statistically measurable, extent. This empirical coherence is illustrated in Figure 7.

## 6. Planning with System Models

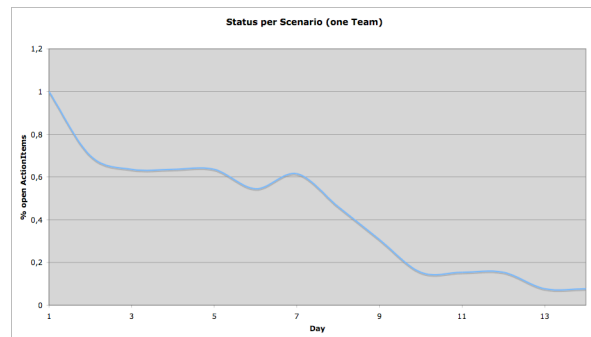
For many project participants, information on how much work remains to be done to complete a specific part of the system model is often quite valuable. For instance, managers may want to know which incomplete implementation tasks prevent the system under construction from fulfilling a specific requirement.

When working according to Scrum, it is common to use so-called burn-down charts as status indicators. These charts show a chronological sequence of the number of open tasks. For the last two weeks of the case study project, the Scrum methodology was used. To carry out the planning, scenarios were used as backlog items. In our case study, we wanted to show that it is possible to generate burn-down charts showing the completion of scenarios based on the

integration between project model and system model.

In the Sysiphus tool, tasks are linked to the corresponding part of the system model, which enables the automatic capture of the status of completion in the following way: The system model defines dependencies between system model elements using model links [17]. That means one system model element requires the completion of another element to be completed itself. For the completion of a certain scenario a number of functional requirements have to be completed. In turn, for the completion of functional requirements certain annotated action items need to be completed. In Sysiphus, you are able to define links, which represent these dependencies as so-called OpeningLinks. Sysiphus can recursively collect all work items along these OpeningLinks, which are required to be done in order to complete a scenario. To do this, Sysiphus first collects the requirements that are still open, and based on them the actual tasks, which prevent completion.

Figure 8 shows the burn-down chart of action items belonging to one of the project teams. Since Scrum was only used for two weeks of the case study and due to the low number of scenarios, we were not able to fully evaluate these features statistically, but according to the project lead, the graph gives a realistic overview of the team status over time.



**Figure 8:** The percentage of open action items of one team and one scenario over the Scrum phase.

## 7. Conclusion and Future Work

In this paper, we introduced a system that explicitly integrates the system model and the project model into one unified model. We described three concepts, which are enabled by such integration. To show the feasibility and advantages of our approach, we conducted a case study. Based on the statistical analysis of this study, we were able to conclude that links between project model and system model shortened the navigational path of

the user (Section 4). Furthermore, we showed that annotated requirements have a higher level of actuality. This evaluation was based on change tracking (Section 5.1) and an interview with the project lead, who rated the actuality of randomly selected requirements (Section 5.2). In section 6, we exemplarily showed that the status of scenarios can be aggregated using annotations and links in the system model.

For further evaluation, we are in the process of integrating our concepts in Eclipse [18], that is, into the workspace of developers. As a consequence, we expect developers to use action items as a starting point for their daily work more often and thus expect them to update related system model elements even more frequently. The successor system called Unibase will also offer status information of system model elements to project participants at project-runtime. We plan to evaluate the benefit of this information to project members using interviews and further statistical analysis.

## 8. References

- [1] C. Amelunxen, A. Königs, T. Röttschke, A. Schür: MOFLON: A Standard-Compliant Metamodeling Framework with Graph Transformations, in: A. Rensink, J. Warmer (eds.), *Model Driven Architecture - Foundations and Applications: Second European Conference, Heidelberg*: Springer Verlag, 2006; *Lecture Notes in Computer Science (LNCS)*, Vol. 4066, Springer Verlag, 361--375.
- [2] R. Gelbard, N. Pliskin, und I. Spiegler, "Integrating system analysis and project management tools," *International Journal of Project Management*, vol. 20, Aug. 2002, S. 461-468.
- [3] A. Forward and T.C. Lethbridge, "The relevance of software documentation, tools and technologies: a survey," *Proceedings of the 2002 ACM symposium on Document engineering*, McLean, Virginia, USA: ACM, 2002, pp. 26-33.
- [4] K. Schwaber. *Agile Project Management with Scrum*. Microsoft Press, 2004.
- [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] B. Ramesh and M. Jarke, "Toward reference models for requirements traceability," *Software Engineering, IEEE Transactions on*, vol. 27, 2001, pp. 58-93.
- [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](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] 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
- [12] B. Bruegge, A.H. Dutoit, and T. Wolf, "Sysiphus: Enabling informal collaboration in global software development," *Global Software Engineering, 2006. ICGSE '06. International Conference on, 2006*, pp. 139-148.
- [13] T. Wolf. *Rationale-based unified software engineering model*. In *Dissertation, Technische Universitaet Muenchen*, 2007.
- [14] J. Helming, M. Koegel, and H. Naughton. *Pause: A project analyzer for a unified software engineering environment*. In *In Workshop Proceedings of ICGSE 2008, Bangalore, India. IEEE CS Press, 2008*.
- [15] *Omg unified modeling language specification version 2.0.*, 2004.
- [16] 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.
- [17] J. Helming, *Integrating Software Lifecycle Models into a uniform Software Engineering Model*. In *Workshop Proceedings of Software Engineering Conference 2008, Munich, Germany*
- [18] B. Bruegge, O. Creighton, Jonas Helming, Maximilian Koegel, Unibase - 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)*.