From Informal Project Management Artifacts to Formal System Models

Jonas Helming, Nitesh Narayan, Holger Arndt,
Maximilian Koegel, Walid Maalej
Technische Universität München
Bolzmannstraße 3
85748 Garching
{helming, narayan, arndt, koegel, maalejw}@in.tum.de

ABSTRACT
Software systems are specified with formal artifacts such as requirements or architecture models. However, informal project artifacts such as bug reports, tasks or discussion threads also include relevant information about the respective software systems and their development. It is beneficial to externalize such information in formalized representations, e.g. to increase the automation of development activities.

In this paper we describe a model that integrates formal system models and informal artifacts of software development projects. We show how this integration eases the manual transition of information from project management artifacts to system models, and discuss how this transition can be automated. To facilitate this transition we propose an approach for the automated identification of informal management artifacts, which contain information about functional requirements and other system specifications such as classes.

Categories and Subject Descriptors
D.2.1 [**Requirements/Specifications**]: Elicitation Methods and Tools, D.2.2 [**Design Tools and Techniques**]: Elicitation Methods and Tools

General Terms
Management, Documentation, Design, Experimentation

Keywords
System model, Project model, Requirement Detection, Formalization

1. INTRODUCTION
The documentation of software development projects consists of two different types of artifacts. On the one hand there are formal artifacts describing the system under construction on different levels of abstraction such as functional requirements, UML models and detailed system specifications. We call these artifacts system models. On the other hand other artifacts rather describe the project itself in terms of tasks, bug reports and informal communication. We call these artifacts project models.

Both, system models and project model are subject to change. For example requirements or bug report change over time. We claim that changes are often reflected in project models rather than in system models [1]. Repositories for project models such as bug trackers, task management systems or simple to-do lists are often accessed more informally then system specifications. As a consequence developers update project models such as tasks on a daily basis. It is even a trend in current software development to open these repositories to other groups beside the project management such as end-users allowing them to enter new work items [2].

Changes applied in informal project model are often not propagated to the more formalized models. This is also true for new elements such as a bug report raising a new requirement. In a survey, developers stated that "Changes of requirements are documented in minutes, change requests or in test cases" "there is often no time left, to repeat work done in other tools, just to have everything consistent". „As a consequence, requirements documents are nothing but a data tomb" [3].

Formal documentation is beneficial for several reasons. In an extensive case study, Charette found that most failures of software projects are caused by improperly defined system requirements [4]. Lamsweerde claims that formal requirements are essential for the design, documentation, communication, reengineering and reuse [5]. Throughout the evolution of the software applications, the original requirements, become blurred, outdated and eventually lost. This "requirement loss" problem becomes critical when the need arises for migrating existing system to new or different platform or execution environment [6]. This might have severe impact on the maintenance cost because the correct understanding of the existing software and code base is an integral part for maintenance tasks [7], [8].

This paper is a part of a larger research towards a unified management of both, formal and informal information in software engineering projects. Our goal is to support the externalization of relevant information from informal project models to formal system models.

The contribution of the paper is twofold. First it shows how traceability between projects models and system models supports software engineers in propagating changes from project models to system models. In other words we show how traceability enables the formalization of informal information. Second, we show in

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Conference'04, Month 1–2, 2004, City, State, Country.
Copyright 2004 ACM 1-58113-000-0/00/0004…$5.00.

this paper, how this formalization process can be automated by identifying parts of the project models that should be formalized.

OUTLINE

We provide an overview of related approaches in section 2. In section 3, we present UNICASE a tool that supports explicit traceability between different levels of abstraction and between system models and project models. In section 4 we show how changes in project models are manually propagated to system models in UNICASE. Furthermore we shortly introduce four case studies, conducted in UNICASE. In section 5 we suggest approaches to semi-automatically support this propagation. As we present ongoing work, these approaches are not evaluated yet. The feasibility and benefits of the approaches are therefore subject to discussion. We conclude and discuss our results in section 6.

2. RELATED WORK

The externalization of informal requirements to formal specifications has been a major area of research and concern in the field of software development. But in the past the requirement elicitation has often been considered a one time job [9], [10], [11]. This is only the case for legacy projects or systems that do not evolve rapidly and only have to cope with the requirements of a defined customer. But with the evolution of Open Source Software (OSS) communities and agile methods the environment has changed completely. In open source projects almost all requirements are embedded in informal communication and the development starts with limited sets of informally communicated requirements. Scacchi [12] presents an extensive case study showing how informal artifacts like bug reports, discussion board and issue tracking systems etc. are used to refine, capture and express system requirements within OSS development and remains an inevitable part.

OSS repositories contains vast amounts of historical data and development pattern which has given rise to extensive research in terms of mining and extraction of the useful knowledge to further refine the underlying principle of software development to fit with evolving software development processes. A major body of research in terms of mining software repositories is based on natural language processing and rely primarily on identifying the semantic relationship between various entities of concern [13-17].

Research practices within this problem domain has produced considerable number of results, but to our best knowledge none of the existing approaches proposed the transition of information from informal project models to system models.

However there are several approaches mining informal project models such as bug reports. Maalej et al. ([18] and [19]) use linguistic processing to analyze work description artifacts (such as commit messages and work logs) and discuss possibilities of automating the creation and usage of work descriptions. Ko et al. [20] present a very similar linguistic approach, focusing on the title of the bug reports as the primary data source for analysis. Both approaches focus on the linguistic analysis of the informal artifacts. The goal is formalize them and mining useful information about the development process.

Fischer et al. [21] focus on analyzing the proximity of software features based on modification and problem report (bug reports) data. Finally they create a visualization of the tracked features to mine hidden dependencies and information. Populating and querying database to extract meaningful and evolution related information remains a major concern of research [22], [13]. These research works consider emails, version control systems and bug tracking system as primary source of information. German et al. [23] uses software trails to reconstruct the evolution of a software system.

Specification discovery from reverse-engineering software traces remains an active area of research. Lo et al.[7] proposes an approach to mine software specifications by employing pattern mining and rule mining techniques on the execution traces of the software system. Reverse Engineering approaches try to reconstruct system models from the source code, but not from informal project models.

3. UNICASE

Our analysis and suggested support for identifying system models are based on a unified model implemented in the tool UNICASE [8]. UNICASE provides a repository, which can handle arbitrary types of software engineering artifacts. These artifacts can either be part of the system model, i.e. the requirements model and the system specification, or the project model, such as work items or information on developers [1]. All concepts presented in this paper are also applicable on an environment where separate tools for system modeling and project modeling are used, as long as they are integrated with each other. We chose UNICASE, as the unified approach because it eases the data analysis without dealing with different data sources.

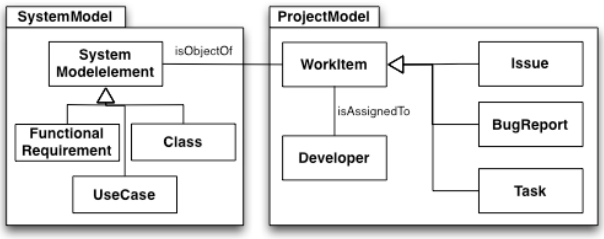

Figure 1: Excerpt from the unified model of UNICASE (UML class diagram)

Figure 1 shows an excerpt of relevant artifacts for our approach. The most important part is the association between work item and system model element. Work items in UNICASE can be issues, tasks or bug reports. In previous work [1], we showed how work items in UNICASE can be linked to the related system model elements modeled by the association isObjectOf. This expresses that activities conducted on the work item are related to the system model element or to its representation in the implementation. The type of relation is depending on the activity of the work item. For example an implementation work item linked to a requirement, means that this work needs to be done to fulfill the requirement. Or, an analysis work item related to a use case needs to be closed to finish the specification of the use case.

UNICASE is implemented as an Eclipse Plugin. As developers in our case-studies (chapter 4 and 5) used Eclipse for development, the tool is directly integrated in their daily work environment. Therefore all artifacts are directly accessible for developers in the same tool.

4. MANUAL PROPAGATION

In this section we show how relations between informal project models and system models support the change propagation between both models. In previous work we have compared two groups of system model elements [1]. While the first group was linked to corresponding work items, the second group was not.

We compared both groups based on two metrics, the number of changes and the up-to-dateness at the end of the project. The up-to-dateness was assessed by the project lead. The group with linked work items had significantly more changes and also was significantly more up-to-date. We were also able to correlate the number of linked work items with both metrics. This supports the assumption that traceability enables manual change propagation from work items to corresponding system models.

As a continuation of this work we were interested whether system models are also created based on existing work items. As an example a new bug report may contain a new requirements, which is externalized as a corresponding artifact. As another example an implementation task may mention a certain component in its description, which is later on added in the system documentation. We do not claim this behavior to be a direct result of the integration between system models and project models. In fact the integration enables us to measure the occurrence of such events. We analyzed the data of four case studies using UNICASE for system modeling and management. DOLLI 2 and DOLLI 3 are large student projects in collaboration with the Munich Airport as an industrial partner. Kings Tale is an ongoing industrial development project of a browser game. As a forth projects we analyzed the data of the UNICASE project itself.

We investigated system model elements, which were linked to work items. For these elements we compared the date of creation. Whenever one of the linked work items was created earlier then the corresponding system model elements, we claim that this system model element was derived from the project model. Please note that this is a conservative measurement as there might be work items with an earlier creation date, which are just not linked.

	Total	Hits	Percentage
DOLLI 2	120	24	20%
DOLLI 3	146	37	25%
Kings Tale	64	12	19%
UNICASE	446	84	19%

Table 1: System model elements created after corresponding work items.

A detailed statistical analysis goes beyond the scope of this paper. In fact this analysis can only be treated as a strong hint that a percentage of system models is derive from work items. This percentage is surprisingly high, especially for the two DOLLI projects, which followed a sequential process for requirements engineering and system design. Our next step will be to match these numbers with the results of user based surveys to substantiate the applied metric.

5. IDENTIFYING SYSTEM MODELS

In this section we propose approaches to automatically identify parts of the project model, which should be externalized in formal system models. We focus on the identification of requirements, classes and components. However, we believe that the suggested approaches are also applicable to identify other types of system models, like test cases or non-functional requirements.

We think that the most important step, which can and shall be supported automatically, is the identification of information to be externalized. Although a system could also provide support for the externalization itself, we believe, that this step remains to be done manually by the user. We suggest supporting the externalization by offering workflow support. As an example the system could offer a wizard to create a new requirement based on a work item. However this support is very specific, we will focus on the identification. In the following we present approaches for the identification of requirements, as well as classes and components.

5.1 Identifying Requirements

To get an inside we manually looked at 200 randomly selected work items from the project UNICASE. Our goal was to find out, how many informal project model artifacts contain information, which should be added in the formal requirements model. We identified two main types of occurrence. First tasks, which are related to the implementation of a requirement, describe the requirement redundantly and add more details, i.e. refining the specification. Second there were Bug Reports, which do not describe a failure, but propose to change requirements or even propose new features. The result of the manual classification shows that 20 % of the work items contain such information. Our goal is to identify those work items automatically. A closer look at the work items reveals, that informal requirements specifications often use certain words and formulations. A very obvious example is the use of the word “should”. Even if “should” is a anti-word for requirements [24] it is often used in informal specifications such as ”The setting should be user specific…”. To substantiate this hypothesis specifically for the UNICASE project we conducted a second experiment. We randomly sampled 50 work items of two groups. The first group of work items contains the word “should”, the second not. An independent UNICASE developer classified the work items. We compared the result of both groups.

For the two samples we calculate 95% confidence interval for the binomial distribution’s probability p. In other words, given the sample the parameter p of the distribution is with 95% probability in this interval. We calculate the interval based on the following formula by using the arithmetical average as a maximum likelihood estimate for p, where k is the number of instances of the sample where the variable is true and a is the constant for the respective significance level (1.96 for 95%).

$$\left|\frac{k}{n} - p\right| \leq a * \frac{\sqrt{(p*(1-p)}}{n}$$

The 95% confidence interval for the group without the word “should” is [0.0556, 0.2859], while the other group results in the interval [0.2964, 0.9036]. The intervals for both samples do not overlap. Thus we can accept the hypothesis that work items in the UNICASE project containing the word „should“ describe requirements more often. Based on this experiment the approach to identify all work items containing the word “should” had to a precision of 60,0% and a recall of 46,1%.

However this result is very specific to the UNICASE project, for a general applicability, we need a more flexible approach. Therefore have also evaluated a basic machine learning approach to automate the process of detecting requirements in work items. Work items have been transformed into a „bag of words“ matrix representation, where each row represents a work item and each column specifies if a given word is present in the description text

(or in the label) of this item or not. A part of this matrix has been used to train a support vector machine classifier (SVM) using the Java Data Mining Package (JDMP) [25]. The remaining part of the data served as a test set, which was used to compare the output of the classifier with the true values. This procedure has been repeated in a ten-times-ten-fold cross-validation scheme, where alternating parts of the data were used as training and test set. This technique was used to estimate the accuracy of this classifier.

However, we found that this basic machine learning approach was not able to detect requirements sufficiently well, as the results were not significantly better than what would have been achieved by chance. This does not necessarily mean, that automatic detection of requirements is impossible. Instead, we suppose, that our data set was too small to serve as a reliable indicator for the structural difference between requirement-related text and other information such as bug reports. Secondly, our machine learning approach should be seen as a very first test which should be improved, especially through a better pre-processing of the data using tf-idf scaling (term frequency / inverse document frequency), and algorithms which take into account relations within the text. This will be subject to further research.

5.2 Identifying Classes and Components

By mining work items, we found out, that classes and components of the system under construction are often used as nouns, even if they are not explicitly modeled [26]. Out goal is to identity this implicit system models. Therefore we used the Stanford Log-linear Part-Of-Speech Tagger [27] to identify the noun words occurring frequently within the informal textual data obtained from the project model. We created a list of nouns and their frequency of occurrence in work items. This list was presented to a project member. They determined, whether a noun is the name of a class or component, which should be part of the system model. We conducted this experiment in two projects, the UNICASE development project as a very large data set and the Kings Tale development project as a small data set.

The Kings Tale project showed promising results, 66,7% of words with an occurrence of 5 or more were classified as classes or components. In contrast, the result did not produce any valuable results for the UNICASE project. We believe this is due to the size of the project. In UNICASE several trivial words like “Developer” or “Bug” were suggested. Therefore we intend to apply the machine learning approach described above also to identify classes and components. However, as these tasks are not binary classification problems and the desired result can come from a set of classes and components (multi-class or even multi-label classification), this is even more demanding. Therefore, it might be necessary to apply more sophisticated approaches for these tasks.

One approach we would like to evaluate is sequence labeling using conditional random fields (CRF) [28]. This approach does not process work item as a whole, but instead is able to detect certain concepts in subsections of the description text. Like hidden Markov models (HMM), this approach takes into account relations between words and their positions in a sentence, and we suppose that it could be able to deal with difficult cases, e.g. differentiate between "a software bug in UNICASE" and "the Bug class used to represent bugs in other software projects".

It may also be interesting to evaluate clustering algorithms. They could give valuable insight to the inherent structure of a software project, when nothing about it is known in advance. The results could form the basis for semi-supervised classification algorithms, which could then be validated and improved through human experts.

6. CONCLUSION AND DISCUSSION

In this paper we discussed the externalization of information from informal project models such as tasks or bug reports to formal system models such as requirements or classes.

In the first part we showed how the integration and traceability between both kinds of models supports manual change propagation from informal to formal artifacts. In future work, we want to observe this behavior more in detail. Interesting future research questions are whether there is user specific behavior in the manual propagation and what events might be triggered for this behavior. We measured the percentage of system models, which were created after a corresponding project model element. Although the percentage seems to be high, it has to be compared with results from projects not using an integrated environment such as UNICASE.

In the second part we suggested approaches to automatically identify formal system model elements, which are actually captured in informal artifacts. A typical example is a requirement described in a bug report. We presented ongoing research and initial results based on experiments. Our results show the general feasibility of an automated support. As an example an occurrence of the word “should” often indicates an informal captured requirement. In a second example we show that in a specific project, nouns, which occur very often in project models indicate classes and components. For both experiments we only measured the precision, but not the completeness (recall) of the identified elements. That means we did not measure how many project model elements contain information about system models and were not identified by our suggested approaches. This is part of future work. Furthermore, the experiments with good results are very project specific. The generalization of the approach by the use of machine learning has to be refined as part of future work. As shown in an initial experiment, the amount of necessary training data might be a problem. Therefore we believe a mixture of both, project specific and general solutions would lead to the best results.

An open question is the workflow of presenting the results of an automated identification to the user. We believe it might not be beneficial to run the identification while users enter informal project models. First this could restrict their habits used way of informally adding new ideas and feedback. Second users might adapt to this identification by avoiding specific phrases and words. An alternative workflow of presenting results of the identification is the recommendation of a list of work items to be externalized to a responsible person, e.g. the project manager. This could be done on a regular basis, e.g. for every release. We believe this workflow would produce synergy effects especially if more than one work items affects a single system model element.

Finally the value of automated identification of system models is subject to discussion. The required precision and especially completeness (recall) of identified elements depends on the goal of the user and is therefore project specific. While even with a low precision automation is useful, a low recall could render the approach useless since no completeness can be achieved.

7. REFERENCES

[1] J. Helming, J. David, M. Koegel, und H. Naughton, "Integrating System Modeling with Project Management–a Case Study," *International Computer Software and Applications Conference, COMPSAC 2009*, 2009.

[2] E. Raymond, "The cathedral and the bazaar," *Knowledge, Technology & Policy*, vol. 12, 1999, S. 23–49.

[3] W. Maalej, "Task-First or Context-First? Tool Integration Revisited," *2009 IEEE/ACM International Conference on Automated Software Engineering*, Los Alamitos: IEEE Computer Society, 2009.

[4] R.N. Charette, "Why software fails," *IEEE spectrum*, vol. 42, 2005, S. 36.

[5] A.V. Lamsweerde, "Formal specification: a roadmap," *Proceedings of the Conference on The Future of Software Engineering*, Limerick, Ireland: ACM, 2000, S. 147-159.

[6] M. El-Ramly, E. Stroulia, und P. Sorenson, "Recovering software requirements from system-user interaction traces," *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, Ischia, Italy: ACM, 2002, S. 447-454.

[7] D. Lo und S.C. Khoo, "Mining patterns and rules for software specification discovery," *Proceedings of the VLDB Endowment archive*, vol. 1, 2008, S. 1609–1616.

[8] P. Jalote, *An integrated approach to software engineering*, Springer, 1997.

[9] B. Belkhouche und J. Kozma, "Semantic case analysis of informal requirements," *Proceedings of the Fourth Next Generation CASE Tools, La Sorbonne, Paris*, 1993.

[10] J.F.M. Burg und R.P. Van de Riet, "Analyzing informal requirements specifications: a first step towards conceptual modeling," *Applications of natural language to information systems: proceedings of the second international workshop, June 26-28, 1996, Amsterdam, The Netherlands*, 1996, S. 15.

[11] R. Clark und A. Moreira, "Constructing formal specifications from informal requirements," *Software Technology and Engineering Practice*, 1997, S. 68–75.

[12] W. Scacchi, "Understanding the requirements for developing open source software systems," *IEE Proceedings-Software*, vol. 149, 2002, S. 24–39.

[13] M. Fischer, M. Pinzger, und H. Gall, "Populating a release history database from version control and bug tracking systems," *Proceedings of the International Conference on Software Maintenance*, 2003, S. 23.

[14] D. Lubar, *It's Not a Bug, It's a Feature!: Computer Wit and Wisdom*, Addison-Wesley, Reading, Mass., 1995.

[15] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, und Y.G. Guéhéneuc, "Is it a bug or an enhancement?: a text-based approach to classify change requests," *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*, 2008.

[16] H.G. Perez-Gonzalez und J.K. Kalita, "Automatically generating object models from natural language analysis," *Conference on Object Oriented Programming Systems Languages and Applications*, 2002, S. 86–87.

[17] A. Hudaib, B. Hammo, und Y. Alkhader, "Towards software requirements extraction using natural language approach," *Proceedings of the 6th WSEAS International Conference on Software Engineering, Parallel and Distributed Systems table of contents*, 2007, S. 155–160.

[18] W. Maalej und H. Happel, "From work to word: How do software developers describe their work?," *Mining Software Repositories, 2009. MSR '09. 6th IEEE International Working Conference on*, 2009, S. 121 – 130.

[19] W. Maalej und H. Happel, "Can Development Work Describe Itself?," *Submitted To: Mining Software Repositories, 2009. MSR '09. 6th IEEE International Working Conference on*, 2010.

[20] A.J. Ko, B.A. Myers, und D.H. Chau, "A linguistic analysis of how people describe software problems," *Proceedings of the Visual Languages and Human-Centric Computing*, 2006, S. 127–134.

[21] M. Fischer, M. Pinzger, und H. Gall, "Analyzing and relating bug report data for feature tracking," *Proceedings of the 10th Working Conference on Reverse Engineering (WCRE 2003)*, 2003, S. 90–99.

[22] O. Alonso, P. Devanbu, und M. Gertz, "Database techniques for the analysis and exploration of software repositories," *MSR*, S. 37–41.

[23] D.M. German, "Using software trails to reconstruct the evolution of software," *Journal of Software Maintenance and Evolution Research and Practice*, vol. 16, 2004, S. 367–384.

[24] W.M. Wilson, L.H. Rosenberg, und L.E. Hyatt, "Automated analysis of requirement specifications," *Proceedings of the Nineteenth International Conferences on Software Engineering (ICSE-97)*, S. 161–171.

[25] H. Arndt, "The Java Data Mining Package - A Data Processing Library for Java," *33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC)*, 2009, S. 620 – 621.

[26] L. Kof, "Text analysis for requirements engineering," Dissertation, Technische Universit\ät M\ünchen, 2005.

[27] K. Toutanova, D. Klein, C.D. Manning, und Y. Singer, "Feature-rich part-of-speech tagging with a cyclic dependency network," *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, 2003, S. 180.

[28] J. Lafferty, A. McCallum, und F. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, 2001, S. 282–289.