Proceedings

Workshop on

Learning Software Organizations and Requirements Engineering

University of Hannover, Germany
March 27-28, 2006
Welcome

at the First Workshop on Learning Software Organizations and Requirements Engineering!

Requirements engineering has grown into a focus topic for most software-dependent companies. Both outsourcing and in-house development call for effective elicitation of requirements, and for rich communication between customer and software developers.

Adequate requirements come from appropriate requirements engineering techniques and processes. They are at the basis of successful software projects. However, requirements engineering is a so-called "wicked problem": by trying to solve it, we create new problems. Documenting requirements well takes time. During that time, requirements changes or re-interpretation of existing requirements may escape our attention. However, poorly documented requirements cause problems in testing and acceptance phases. Wicked problems are ill-framed and ill-defined; there is no one correct and optimal solution, but an on-going process of negotiations and learning. The problem evolves together with attempted solutions.

Organizational learning is, therefore, a natural complement when we discuss requirements engineering. A learning software organization will heavily rely on competent and learning individuals – but it will also go beyond the individual learning level: Truly learning organizations are characterized by their attempts to foster and support learning on both individual and structure levels ("organizational"). Processes and tools are systematically improved, reflection and explicit learning is part of the company culture. Many real companies are still struggling to reach this goal.

LSO+RE continues the tradition of the International LSO Workshop series which addresses all aspects of learning in a software organization. In 2006, the main LSO workshop will be held in Brazil. We wanted to offer a “birds-of-a-feather workshop” in Europe, and we decided to give it a more narrow focus: LSO+RE is supposed to provide a forum for discussing the intersection of requirements engineering and learning software organizations in a more in-depth way.

The organizers wish all participants two thought-provoking days, stimulating discussions, and a good time in Hannover!

Kurt Schneider  Workshop Chair  University of Hannover, Germany
Andreas Birk  Program Chair  sd&m, Stuttgart, Germany
Torgeir Dingsøyr  Program Co-Chair  SINTEF, Norway
Stefanie Lindstaedt  Publication Chair  Know-Center, Graz, Austria
# Table of Contents

**Session 1: Getting better, Improving Requirements Engineering**
- Requirements Management in Distributed Projects ........................................ 9
- The Value of the Quality Gateway ....................................................................... 17
- A Case Study on Overcoming the Requirements Tar Pit ...................................... 25

**Session 2: Information Flow in Requirements Engineering**
- Learning to Tailor Documentation of Software Requirements .......................... 39
- The Requirements Engineering Gap in the OEM-Supplier Relationship ............. 47

**Panel: The Role of Learning in RE**
- Learning from own Experience and Advices in Literature ................................. 59
- Understanding Requirements through Filtering, Negotiating and Shifting .......... 65

**Session 3: Reuse and Product Lines**
- A Collaborative Learning Experience in Modelling the Requirements of Teleoperated Systems for Ship Hull Maintenance ...................................................... 71
- Project-oriented Reuse Approaches: Copy-and-Paste or Software Product Line Engineering? ................................................................. 81

**Session 4: Organizing Requirements**
- Using Requirements Engineering (RE) Patterns for Organizational Learning ...... 101
- Using Wikis to Manage Use Cases: Experience and Outlook ............................. 113
Session 1: Getting Better, Improving Requirements Engineering

Requirements Management in Distributed Projects
D. Smite, Riga Information Technology Institute, Latvia

The Value of the Quality Gateway
Marc Philipp Werner, Hewlett-Packard GmbH, Germany

A Case Study on Overcoming the Requirements Tar Pit
Samuel Fricker, ABB Switzerland Corporate Research and University of Zurich, Switzerland
Martin Glinz, University of Zurich, Switzerland
Peter Kolb, ERNI Consulting AG, Switzerland
Requirements Management in Distributed Projects

Darja Šmite
(Riga Information Technology Institute, Latvia
darja.smite@riti.lv)

Abstract: Global software development expands with every year providing software developers with new opportunities. However, practitioners face global challenges and threats particular for distributed environment that require new methods and tools to be implemented. This paper provides a report on requirements management practices in globally distributed projects in one of the leading software development organizations in Latvia. The paper discusses how to form requirement analysis teams wisely, how to reduce diversity between the involved parties, what to be aware of during the development phase, and how to facilitate successful communication for entire project.

Keywords: Requirements management, Global software development, Distributed software development projects, Lessons learned, Best practices

1 Introduction

Global software development nowadays is not a phenomenon. It expands with every year and turns from a trend into everyday type of doing business. GSD enable reaching mobility in resources, obtaining extra knowledge, speeding time-to-market and increasing operational efficiency [Smite, 05]. Nevertheless, globalization has also significantly changed the nature of software development projects (see Fig.1). Software development in distributed environment is facing changes by involving related partners which are distributed in time, space and culture, “…each with its own set of needs that require unique methods of organization and control” [Karolak, 98]. The complexity of collaboration grows with the number of parties involved in the supply chain of a global project.

Figure 1: Globally distributed project scheme [Smite, 05]
While software systems complexity grows along with sophistication of software development processes, the quality of requirement analysis and management remains one of the most important sources of risk. Most of the problems encountered in software development are attributable to shortcomings in the processes and practices used for requirements engineering [Wiegers, 99]. This consideration remains the same for distributed projects. An investigation of 28 distributed projects has demonstrated the importance of requirements management and adoption of new practices for coping with global risks in distributed projects.

This paper is organized as follows. The next chapter provides research overview. Discovered practices are described in chapter 3, which is followed by discussion of the results in chapter 4. The paper ends with conclusions and insights in future work.

2 Research Overview

The author leads a research which aims to improve software development processes in globally distributed projects in one of the leading software development companies in Latvia. The research major objective is to build a framework that would serve as an Experience Factory providing best practices for global project improvement.

2.1 Research Approach

This paper describes an exploratory research based on the field-studies. The research takes place in one of the largest software houses in Latvia. Data on globally distributed software development projects was gathered through series of interviews with experienced project managers [Smite, 05], surveys [Smite, 04], and related literature overview. The data was analyzed according to principles prescribed by a grounded theory as described by Strauss and Corbin [Strauss and Corbin, 98]. Grounded theory is a research method that seeks to develop theory that is grounded by data about a certain phenomenon (here – global software development) that has been systematically gathered and analyzed.

Through applying open coding followed by axial coding, and then selective coding, a set of global factors and associated global threats has been derived.

2.2 Survey Overview

In order to validate the research results, a survey was conducted considering the following objectives:

- to investigate, which threats are faced by global projects;
- to evaluate the magnitude of consequences of the threats.

Web-based inquiry forms were filled by 28 distributed project managers form 3 software houses in Latvia, which are involved in software development projects with geographically distributed customers or prime contractors. The nature of the projects investigated is either custom or product software development, and software maintenance including existing software improvement. The survey uncovered a list of most frequent threats and evaluation of their risk level.

Risk level (0-5) was evaluated as a combination of frequency of occurrence of the threat and the magnitude of its consequences for each of the following cost-related, time-related or morale-related project results: Unexpected management costs, Budget
overrun, Time delays, Late product delivery, Customer dissatisfaction, Undermined morale, Disputes and litigations, Customer costs escalation.

The following scale was used for frequency evaluation: 5 (81-100%); 4 (61-80%); 3 (41-60%); 2 (21-40%); 1 (1-20%); and 0 (0%).

The following scale was used to evaluate the magnitude of consequences: 5 (Disastrous); 4 (Significant); 3 (Moderate); 2 (Minor); 1 (Negligible); 0 (None).

Overall threat magnitude of consequences evaluation has been derived by calculating an average size of magnitude for each of the consequences caused by the threat, and then choosing maximum out of these values. Subsequently, if a threat significantly impacts cost-related results, but remains insignificant for e.g. morale-related results, it still is evaluated as significant.

3 Survey results

The survey uncovered the following TOP5 threats (see Table 1).

<table>
<thead>
<tr>
<th>Threats</th>
<th>Risk Level</th>
<th>Frequency</th>
<th>Magnitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poorly defined or inconsistent software requirements specifications</td>
<td>3</td>
<td>4 (62%)</td>
<td>3</td>
</tr>
<tr>
<td>Faulty effort estimates</td>
<td>3</td>
<td>4 (62%)</td>
<td>3</td>
</tr>
<tr>
<td>Diversity in process maturity and/or inconsistency in work practices between the partners</td>
<td>3</td>
<td>3 (52%)</td>
<td>3</td>
</tr>
<tr>
<td>Increased level of unstructured poorly-defined tasks</td>
<td>3</td>
<td>3 (45%)</td>
<td>3</td>
</tr>
<tr>
<td>Poor or disadvantageous distribution of software development activities between the customer and supplier(s)</td>
<td>3</td>
<td>3 (41%)</td>
<td>3</td>
</tr>
</tbody>
</table>

*Table 1: TOP 5 threats faced by distributed projects*

The results show that issues connected with requirements management have been named as one of the most important sources of risk. Comparing the results of TOP 5, we can conclude that the mentioned threats are correlated. Software development lifecycle processes in global projects are often distributed between the parties involved in the project. This produces problems for accurate effort estimates. Requirement analysis and further management seen as a core activity is frequently not being outsourced but instead performed in-house. However, followed by diversity in process maturity or inconsistency of work practices, this can cause various problems. Magnified by increased costs of logistics of holding face to face meetings, increased virtualness, lack of expertise in outsourcing projects, relatedness with other suppliers,
organizational culture mismatch, etc. requirements management needs new practices to be implemented in globally distributed projects.

The following additional threats can be named as reasons for poor requirements management:

- diversity in process maturity,
- inconsistency in work practices,
- lack of version control,
- relatedness with other suppliers,
- lack of language skills,
- terminology differences,
- customer employee unwillingness to collaborate.

One project manager reported „Customer claimed that they do not have sufficient human resources to validate all software requirements specifications“. Another reported that „it is customer’s strategy for decreasing the expenses“.

Further requirements clarification can be also put under threat due to various risks that lead to troubled communication, such as the following:

- increased virtualness due to dominant use of asynchronous communication,
- increased cost of logistics of holding face to face meetings,
- increased complexity of spreading awareness and knowledge,
- customer’s belief that the work cannot be done from a far off location,
- lack of team spirit,
- lack of clarity about responsibility share,
- poor cultural fit,
- lack of experience and expertise with outsourcing projects.

In the next chapter the major practices for successful requirements management in distributed environment are described. These practices are derived from the field studies and address threats that are specific for global projects.

4 Practices

4.1 Form the requirement analysis team wisely

Requirements elaboration is produced without the distributed development supplier involvement in 80% of the investigation projects.

However, distribution in space, time and culture can cause significant problems in requirement further clarification. In order to mitigate the risk of poor transition from analysis to development phase, it is advisable to consider involving supplier representatives in the requirement analysis team. This can be achieved by sending one or more analysts from the supplier side to participate throughout requirements analysis on the partner’s premises.

4.2 Reduce diversity

A consistent challenge experienced in distributed work is maintaining coherence, commitment, and continuity across the multiple locations, priorities, and interests of the hundreds of people involved in the collaborative effort [Orlikowski, 02]. Diversity
in process maturity and work practices can bring sudden risks to the project, such as time delays, unexpected management costs and low morale.

Awareness about diversity can be used to plan and mitigate the related threats. A common understanding of work practices shall be established and maintained through initial training and socialization workshops. Training in “soft skills” such as trust, cultural differences, communication, collaboration, context sharing, and knowledge management, is useful.

4.3 Agree upon requirement analysis template

Context differences, such as diversity in process maturity and practices, organizational and cultural differences, diversity in employee education, very often cause problems with the development team expectations for the requirements specifications. It is therefore useful to discuss what kind of specification the supplier expects in the very beginning of the requirement analysis phase. A template based on these expectations can be developed, approved and used during requirement specification.

4.4 Develop a glossary

Linguistic and context differences can cause misunderstandings in achieving an understanding of requirements. Some project managers report that what is seen as obvious for the customer, sometimes is hard to understand for us. Most of the problems are caused by terminology in a certain business sphere.

Therefore, it is commendatory to come to a general agreement on terminology by developing a joint glossary.

4.5 Implement a version control tool

Though poor version control is faced only in 14% of the projects, lack of joint version control accompanied by lacking proximity can lead to misunderstandings, time delays and rework. One project manager reported „Sometimes we even do not know about the changes. To do something new we have to ask always for the latest version”.

Advanced tools shall be implemented to support multiple teams and provide online access for every partner.

4.6 Establish clear responsibilities and priorities

Lacking next door closeness and proximity between the teams involved in a joint project, there is a risk of miscommunication. One project manager reported “We cannot allow everyone communicate with anyone”. If this happens, requirements and changes with high priority comes from everywhere and can be hardly managed in order.

Therefore, it is very important to define project member roles and establish proper communication liaisons. One project manager reported “We have defined not only the project members who should be added in “Cc” fields for emails, but we have also defined different titles and kinds of “Subjects” for every type of discussion”. 
4.7 Maintain constant communication

The partners shall understand that not all the problems and questions can be solved by email or phone. One Developer reported that when their systems analyst comes to the distributed development team on a visit, developers always discuss huge amount of insignificant problems that they can’t solve through emails of phone calls.

Experienced project managers advise to organize face to face meetings once in a month or two for planning and requirement ambiguity clarification. Advanced communication tools as videoconferencing are very effective and should be used extensively whenever possible to discuss the requirements and changes.

Many project managers report that employees who lack fluent language skills are afraid to speak with the partner over phone. This causes dominant use of asynchronous communication tools and brings time delays in problem solution turnaround. A good thing to do is sending one supplier representative to the partner side for the entire project to relieve communication (particularly feasible in large projects).

4.8 Involve optimistic employees

When a company starts to outsource a part of software development life cycle activities, the in-house employees can be put under risk of possible human resource optimization. This can cause customer’s employee unwillingness to collaborate with the sub-contractors throughout the project. Accompanied by a belief that the work cannot be done from a far off location it can significantly trouble cooperation between the distributed teams. In result, the developer, who has received requirements specifications for further development, is left without a helping hand.

A good piece of advice is to involve only optimistic and enthusiastic employees in global projects.

5 Discussion

Despite a wide variety of literature on whether and why to outsource, there is still a lack of research on how to achieve successful performance in distributed environment. Many companies fail in the execution of strategic outsourcing [Laplante et al., 04].

Most of the problems discussed in this paper address diversity, relatedness and poor communication issues that highlight particularities of software development involving geographically distributed teams. Managing requirements in a distributed environment can become a tough task if the process is not well defined and the teams are not experienced or prepared for this cooperation model.

To overcome the problem of distance in GSD, various managers are experimenting and quickly adjusting their tactical approaches [Carmel and Agarwal, 01]. However, it’s difficult to muster the energy needed to overcome obstacles to change and to put new knowledge into action [Wiegers, 99]. Organizations are naturally resistant to changes. Supplier teams often report on the customer unwillingness to adopt mature processes because it requires more time and resources. One project manager reported “We can do nothing about it. The word of the customer is a rule”.
Knowledge and information distribution among the involved parties is also a challenge. One project manager reported that poor requirement specifications actually “reflect poor process for knowledge distribution by the customer team”. Another reported about problems with a mediating contractor company that prevented supplier and customer direct collaboration and introduced delays, passed on the information selectively, or added some redundant surplus information.

Dealing effectively with global factors requires much effort and a deep competence in what may be labelled “distributed organizing” – the capability of operating effectively across the temporal, geographical, political, and cultural boundaries routinely encountered in global operations [Orlikowski, 02]. Organizations that involve subcontractors should experiment and adjust the global product delivery models by decreasing the processes and interaction layers that do not add value and consider improvements that would enable effective cooperation of distributed teams. Diversity and lack of common goals make organizations consider new approaches as partnership to be implemented to achieve outsourcing benefits more effectively [Lee et al, 00].

6 Conclusions and Future Work

Virtual product development is considerably more complex than even the most complex project managed entirely in-house [Karolak, 98]. A set of global factors cause various threats that are particular for distributed environment. The survey results show that requirements management in global projects is one of the essential challenges that shall be paid adequate attention. Separation of the team that specifies requirements and the team that produces software is joined by diversity between processes, inconsistency in practices, linguistic and terminology differences, temporal distance etc. This makes practitioners to seek for new approaches and practices to be implemented in global projects for better performance.

Communication plays an important role in successful distance overcoming. Therefore training in “soft skills” such as trust, cultural differences, communication, collaboration, context sharing, and knowledge management, is essential.

While global factors preclude distributed environment transformation into a common way of producing software in-house, practices that mitigate global risks shall be implemented in order to overcome global challenges such as distribution in space, time and culture. The author’s future work is related to deriving facilitates, methods and practices for better performance in global project environment.

The practices described in this paper of course cannot be spread to any global project, as the nature of software development projects in distributed environment is very diverse. Nevertheless, the description of threats and possible outcomes may be useful, in order to make the right decision on project activity distribution and joint performance.

7 Acknowledgements

The author appreciates valuable research input received from the developers and project managers from the investigated software house.
This research is partly supported by European Social Fund and the Latvian Council of Science project Nr. 02.2002 “Latvian Informatics Production Unit Support Program in the Area of Engineering, Computer Networks and Signal Processing”.

References


[Smite, 04] D. Smite, Global Software Development Project Management – Distance Overcoming, In Proc. of Int. Conf. on European Software Process Improvement (EuroSPI), Springer, November 2004, …


The Value of the Quality Gateway

Marc Philipp Werner
Hewlett-Packard GmbH, Germany
maphi@hp.com

Abstract: Over the last nine years, requirements management in the HP OpenView Operations for Windows R&D team evolved from informal documents to a documented, tool-supported process. Early quality and content checks ensure that only well-defined requirements are entered into the system; this is called the quality gateway. Clear definitions of ownership over the whole life cycle of a requirement create accountability and enable the various stakeholders in the organization to track what is being done, who is working on it, and what the status is.

Keywords: Requirements management, case study

Categories: D.2.1

1 Introduction

The HP OpenView Operations for Windows R&D group operates in a highly distributed environment. The various teams are distributed in five sites, four counties, and three time zones. This makes it impossible to meet face-to-face, and even telephone conferences are hard to set up, because there is no time when all sites are in working hours.

The Operations product is standard software, not project software. This means that the environment is relatively stable, the stakeholders remain more or less the same, and revolutionary changes of the development processes are impossible. The team has the responsibility to increase market share and revenues from the product. This is accomplished by selecting those requirements that maximize customer value and revenue, and by optimizing the processes to minimize the cost for development.

2 The early years

When the project was launched originally in 1997, requirements management was not done in a very formal way. Although the development process demanded a requirements document, there was no real definition of the form or content of this document.

At that time, the Operations product was available on Unix only. To gain a foothold in the growing Microsoft Windows market, HP acquired a small company that already had a product available. The first project was defined as bringing the capabilities of the Unix product into the acquired product, based on Microsoft technologies.

Right from the start, it was obvious that the Unix product had too many features to bring all of them into the first release of the new Windows product. To decide on
the priorities, a list of features, which grew and changed over time, was compiled on whiteboards and spreadsheets. In absence of a real customer representative, a senior manager with experience in the Unix market took over that role.

About a year later than expected, the first release of Operations for Windows finally happened in 2000. The subsequent release only had two requirements: “Deliver a Japanese version of the product” and “fix the defects”. Effectively, requirements management was done in the defect tracking system, by prioritizing the defects and fixing the most critical ones first.

3 First steps towards requirements management

While R&D was still working on the second release, the organization decided to invest more into requirements management. An internal expert was chartered with evaluating different requirements tools and processes. The final choice was to go with the Volere template and the Caliber tool.

In 2000, for the third release, the R&D group decided to use this tool to manage the growing list of enhancement requests that had started to arrive. At first, only high-level entries made it into the system; the typical requirement consisted of a single line of text, an importance (usually, but not always, “MUST”), and a project version (“current release” or “next release”). The team managers used these entries to assess their work. During this time, there was no engineering representation in the process, because there was no overall software architect for the whole group.

At this stage, most engineers did not even use the requirements tool. Whenever they did, they reacted with frustration: the one-line descriptions were not detailed enough to work with. So, for each item, the author (who might or might not have been available for this task) had to be consulted for more details, or the details had to be made up. Depending on the experience of the engineer, the alignment between original request and implemented result varied.

4 Train the users

To improve the situation, the whole lab took the “Mastering the Requirements Process” class from Robertson & Robertson, which introduced the idea of a clear “fit criterion” that describes measurements for requirements. Initially, engineers were happy because this described exactly what they wanted.

Unfortunately, reality quickly showed that fit criteria were not provided; the requirements submitters did not believe that it would be a good investment of their time to write a good fit criterion, and nothing really changed in the process.

At this stage, there was a lot of frustration in the team. The marketing representatives believed that they had given clear guidance to the R&D folks. The R&D managers told their engineers to base their work on the requirements. The engineers, however, could not find the information they needed. Furthermore, the tool had some performance issues for all remote users, which did not help to convince people to use it.

Nobody really took care to maintain the data, either. Although there was some structure in the tool, there was no documentation about where to put what. Duplicate
entries crept into the system; already-implemented requirements were not updated and remained in the system for years.

5 Introduction of the quality gateway

Then, in 2002, two architects were assigned to the group. Among other duties, they took control of the requirements and introduced a quality gateway, as described in [Robertson 99]. All requirements had to be reviewed and approved by the quality gateway before they could be presented to the teams at all. This principle was applied to new entries as well as to existing ones. In essence, to get into the process, a requirement had to contain the following:

- A clear name
- A description of what is requested
- A purpose, giving the reasons why this is requested
- A ‘fit’ criterion, detailing how the requirement can be tested from a customer point of view
- A source, stating where this requirement comes from
- A statement, from marketing, of the importance of the requirement
- A project version, also from marketing

This was a cultural shock at first. The requirements database contained over 800 open entries at that time, many of which were outdated. In addition, there were duplicate entries, and several entries where the submitter had left the group and no-one quite knew what was being asked for. The marketing representatives had an especial rude awakening, as they were being told that the majority of the items in the system needed to be reworked, augmented, or otherwise fixed.

The clean-up took the better part of a year. The submitters quickly changed from resistance to support, when they realized that they were finally getting immediate feedback on their work. The architects checked the new entries at least once a month, instead of looking at new submittals only when new releases were planned. The database started to feel like an active tool instead of a data grave. As obsolete entries were identified, over 200 requirements were deleted. The remaining items were well understood by project management, marketing, and the software architects alike. As a side effect, the architects acquired a good understanding what was already in the system, which helped to prevent further duplicates.

6 Reaping the benefits

This clean-up effort allowed the introduction of a more detailed requirements management process in the next release.

First, requirements were grouped into “themes” which provide marketing and management with the high-level concepts that they could use in external communications. Because the themes of the release were decided upon earlier, the exercise of prioritizing the requirements became much easier; requirements that did not fit the themes were scratched immediately.

Second, an owner was assigned for each requirement. The owner had the responsibility of investigating the requirement and coming up with the effort
estimation. These estimations were also tracked in the requirements tool. By delegating the requirements work to several people, the work load on the architects was significantly reduced. The engineers felt more connected to the project, because they were involved in the process. And finally, individual responsibility for each item also created accountability, which helped to ensure that no requirements were forgotten.

Third, requirements were mapped to project iterations. Products are developed using time-boxed cycles, referred to individually as an “integration cycle” (IC) or alternatively as a “vertical slice” (VS). A requirements report specified the integration cycle in which each item would be delivered, allowing the system test team to plan their test sequence accordingly.

The increased quality of the requirements enabled the individual engineers to work from the requirements directly. Whereas previously, they had to ask for details on 100% of the requirements, now only 10-15% needed follow-up discussions, most of which were on a much more detailed level than any of the original ones. Furthermore, the software architects could usually act on behalf of the original submitters in the discussions, because they had sufficient understanding of the intentions behind the requirement. Today, the original submitters are asked for more information in less than 5% of the requirements — once the requirement has passed the quality gateway. Of course, the effort spent on the quality gateway itself is significant; one of the architects spends about 25% of his time with requirements management.

As a result of these changes, the teams have embraced the requirements process, and more and more work is being done in the requirements tool instead of using spreadsheets and email. Project management has much more confidence in requirements planning, and now demands that each requirement must be entered in the requirements database before a schedule can be committed to. The team work items can be assessed directly from the database. Testers also work from the requirements database, creating system test cases for each requirement.

And at the end of the release, it was relatively easy to find all requirements that had been completed, so that they could be archived, making it easier to manage the data for the remaining product requirements.

7 Further refinements

Following the positive experiences with the new process, further refinements were made for the next release, which is the current one as this document is being written.

It transpired that effort estimation tracking does not really work in the tool. For most requirements, more than one person needs to be involved in tracking the requirement. For example, on our products, we need input from people in the server, GUI, testing, and documentation teams. This made it hard to track the data in the single field that the tool provides. Therefore, effort estimation has been moved back to the investigation reports, which describe what needs to be done to realize a requirement.

For the same reasons, a single owner for each requirement did not work out too well either. However, the tool actually provided the means to assign multiple
responsible persons in various roles. Using this capability, each person on the team can now search all requirements on which he or she needs to contribute.

Another pain point in the last release was the discussion of the requirements, which usually happened by email. People not on the distribution list lacked the background to understand changes to the requirements. Therefore, the discussion on the items was moved into the tool as well, which provides a small forum-like capability for this.

Finally, the life cycle of the requirement has been defined more closely, and all the people involved are asked to keep the status up-to-date. This makes it easier for management to track the project progress.

8 The process in detail

All new requirements are entered into a special folder called “Incoming”. Anybody in the organization is allowed to enter requirements here, although most requirements come from marketing.

About once a month, the architects review this folder. All requirements are checked for completeness (see above) and quality. Requirements that do not pass this quality gateway are rejected; the author is notified and is given 60 days to improve the requirement, or it will be deleted. Requirements that pass the quality gateway are given the status “R&D analysis”.

Then, the marketing representative checks the new entry, and decides which planned project version (current release, next release, and so on) should satisfy the requirement. The marketing representative also decides on the importance of the requirement such as “will delay release for this”, “has to be investigated in detail”, or “opportunistic requirement”.

![MSO requirements state flow](image-url)
When both reviews have been completed, the requirement is moved from the “Incoming” folder to the final destination in the system.

All requirements that belong to the current release are assigned to a group of responsible engineers who have to investigate the requirement and come up with the effort estimation. One engineer has the special role of owner, which means that he or she is asked to coordinate the group. The owner is also responsible for updating the requirement status. When the investigation is completed, the requirement is given an R&D proposed priority (in-plan, out-plan, opportunistic) which reflects the confidence that the requirement can be implemented in that release, given the manpower and time available. The requirement status changes to “Pending”.

In parallel, the testers perform another review of the requirement quality from their perspective. If the requirement does not meet the testability criteria, the author is contacted to add the missing details. Although this should be part of the quality gateway already, organizational reality forces us to postpone this step to the investigation phase, unfortunately.

As part of the “end-of-investigation” milestone, marketing and R&D management review the list of requirements again and discuss the proposed priorities. Trade-offs are made, sometimes the schedule is extended to allow for some more opportunistic items. At the end of this scoping session, all priorities are updated to reflect the agreement, and the requirement statuses change to “Accepted”. Also at this time, requirements that are accepted as out-of-plan are usually moved to the next release. After this milestone, new requirements for the current release are accepted only with a formal change request that makes visible the effort involved and potential project delay.

Next, R&D comes up with an implementation plan that shows which project iteration will deliver which requirements. This is used as a basis for the cross-functional teams (testing, documentation) to plan their schedules.

As soon as a requirement is implemented, its status is changed to “Done” by the requirement owner. Weekly reports show the progress of the team to management.
9 Conclusions

In summary, the value we gain from a solid quality gateway is as follows:

- Stakeholders trust requirement data that is alive, and reflects current thinking.
- R&D can work directly from requirements instead of querying the submitter.
- At least two people have a good overview of all the requirements, both planned and future ones.
- Project schedules can be created on a per-requirement basis.
- Testing can be planned for each requirement before R&D knows the implementation specifics.
- R&D avoids spending time investigating fuzzy requirements.
- Stakeholders can track the progress of all requirements, for example, marketing can always check the implementation status for each requirement.

10 Future Plans

No process is ever perfect. Our engineers have already asked for the next improvements, which will be incorporated in future releases.

One request is for finer granularity for requirements in the development process. Today, component teams may face unpleasant surprises if they had not expected to contribute to a requirement. Each team would prefer to have individual requirements for each component, (for example, the server component or the GUI component) instead of sharing a single item. This would allow for a better model of dependencies, for example, where the server team has to deliver before the GUI team. As the component teams are not necessarily in the same location, this would help to decouple their activities. It would also allow tracking the effort estimations in the requirements management tool again. In the end, the complete investigation report document would be a derivative of information stored in the requirements tool.

Another challenge is the necessary integration with changing R&D development process paradigms. Our current approach is still based on the traditional waterfall model which places the investigations and specifications at the beginning of the project. Current theories suggest the superiority of agile processes that investigate, design, and implement in each iteration. Although our current requirements management process is not mutually exclusive with such agile approaches, a change of thought process is required from management and marketing that might be a similar cultural shock as the original introduction of the quality gateway.

Acknowledgements

Gerald Heller provided guidance and consulting both for the requirements management process described inhere, as well as for this paper itself. Keith Hodson helped with language and grammar.
References

[Robertson 99] Robertson, Suzanne & James: Mastering the requirements process. Addison-Wesley, Harlow, Great Britain, 1999


A Case Study on Overcoming the Requirements Tar Pit

Samuel Fricker  
(ABB Switzerland Ltd., Corporate Research, Switzerland, samuel.fricker@ch.abb.com  
University of Zurich, Switzerland, fricker@ifi.unizh.ch)

Martin Glinz  
(University of Zurich, Switzerland, glinz@ifi.unizh.ch)

Peter Kolb  
(ERNI Consulting AG, Switzerland, peter.kolb@erni.ch)

Abstract: Software requirements are defined in many industries using informal software requirements specifications that are based on standards such as IEEE 830. Also, in teams of small to medium-sized projects there are often no experienced requirements engineers. These two factors leave product development efforts in a tar pit of ambiguities and misunderstandings that is risking product success. We investigated the adoption of systematic requirements engineering techniques in such a mid-sized software development project. We show how the project found itself in this tar pit and went through several failing attempts of using methods believed appropriate to finally discover and tailor a non-standard approach that led to a massive improvement of the requirements for the product to be developed.

Keywords: Requirements Engineering, Methods, Learning, Business Modelling  
Categories: D.2.1, D.2.9, I.6.5, J.6, K.3.2, K.4.3

1 Introduction

Hundreds of methods have been created to address the problem of acquiring, analyzing, and communicating requirements for software systems. A majority of these methods promises a holistic solution to this problem, however without considering situational characteristics [Fitzgerald, 96].

Industry is rather reluctant in adopting such methods, because they are believed not suitable to the needs of development organizations and projects. Methods are often seen as limiting, slowing down engineering work, and generating bureaucracy [Smolander, 90]. In requirements engineering (RE), the methods that get into use are consequently often of very general nature and provide little guidance to the requirements engineer due to lacking specificity. A typical approach is the use of word processor templates based on standards like IEEE 830-1998 [IEEE, 98].

Promotion mechanisms ensure that skilled and experienced employees rapidly get allocated to large and complex projects. Small and mid-sized projects thus get into a situation where they lack knowledge required by the task at hand and adopt practices that may not prove adequate. In such projects there is usually no time available to identify and introduce an RE method fitting the circumstances. This situation leaves project teams with the choice of struggling in the tar pit or trying to learn and apply RE methods on a trial and error basis.
Overcoming this problem requires understanding how methods are selected and tailored to project needs and understanding how knowledge and skills of a project team evolves. With such knowledge, processes can be shaped to ensure success of RE in conditions of thin-spread knowledge and experience. Such knowledge is also required for understanding how methods can be deployed in a sustainable manner.

We investigated the adoption of RE techniques in a mid-sized software development project. In this paper we describe how the project found itself in the requirements tar pit, went through several failing attempts of using methods believed appropriate, and finally discovered and tailored a non-standard approach that led to a massive improvement of the requirements for the product to be developed.

The study provides a rich picture of the evolution of the team’s knowledge and behaviour, the motives for taking up and tailoring various practices, and the reasons for the final success. The study contributes thus with a deep first-hand understanding of adopting and tailoring of RE methods in practical circumstances.

The paper is structured as follows. Section 2 outlines the research approach. Section 3 describes the project’s initial situation, the evolution of practices, and the impact of the successful approach. Section 4 discusses the factors that influenced method adoption and success. Section 5 summarizes and concludes.

2 Research Method and Process
The research underlying this paper focuses on understanding the evolution of RE knowledge and skills in industrial circumstances. The research question is formulated as: how does a software team learn RE while practicing it?

The nature of the research problem calls for a qualitative approach that retains the holistic and meaningful characteristics of real-live events. This paper is based on an exploratory case study [Yin, 03] that used participant-observation, documentation, and interviews as information sources. The primary unit of analysis was the behaviour of the project team in RE-related activities.

The first author has participated in the described project by filling the role of a requirements engineer. At the start of the project he had a few years of software engineering experience and was not specifically trained in RE practices. Recorded information was drawn from minutes of meetings, e-mails, intermediate work results, presentations, formal documents required by the organization’s software development process, the requirements model, and reports. After concluding the project phase that focussed on RE, key stakeholders were interviewed to understand their view and opinions about the experience. This interview was repeated after an additional year.

The personal experience, above-mentioned data, and study of literature were used to formulate the project narrative and to understand the implications. Finally multiple versions of this paper were inspected by the project team.

3 The Tar Pit and Escape Attempts
The considered project has been carried out in a company, which is part of ABB, a global leader in power and automation technologies with about 105’000 employees. Product development is carried out in that company in a multi-project framework. The projects pursue a sequential development lifecycle.
The software to be implemented by the project was a new tool suite for engineering and maintaining intelligent hardware products. It provided the user with data and rule management features and the capability of down- and uploading this data with status information to and from the hardware devices.

The project team consisted of seven software engineers that maintained such software for up to a dozen years. The team included the first author who was the only one new to the domain. The team was planned to grow to about thirty members, which would implement the new software product within one year. Due to the focus on maintenance in the past, the team and stakeholders had no particular experience in engineering requirements for new products.

All stakeholders had engineering background and experience in the tools domain. Not all were easily accessible, though. On-site were peer projects, end-user training, and line, program, and quality management. Other stakeholders, including product management, customer service, and domain experts, were working at remote sites and belonged to formerly competing companies. Access to end-users was not possible.

3.1 Initial Requirements

According to the company’s processes, technical software requirements specifications (TRS) lay the basis for any development effort. A TRS is defined on the basis of a market requirements specification (MRS). Both documents are written in natural language. Such requirements seem easy to write and share, because the templates focus on the right subjects and any professional masters natural language.

As the project team received the TRS, with the objective to define the solution, they had no clue of what was expected. The TRS was a mere compilation of 120 requirements from related products, which were believed relevant. The requirements were of such quality that it was impossible to interpret their meaning. RID-001 (Table 1) is a representative example to illustrate the problems the team was confronted with.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>RID-001</th>
<th>Priority</th>
<th>High</th>
<th>Source</th>
<th>MRS-001</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Same software for same tasks in each project phase.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Description| - Should be understood as “same software component for same tasks …”.
- The software should work as thin-client over a network connection to the server software and installed together with the server software on a notebook for local usage. |

Table 1: Example of a problematic formulation of a functional software requirement

All requirements in the document were similarly fragmentary and presented with inadequate structure. They had been written in abbreviated sentences without enough explaining context. If known terms were used, they frequently changed their meanings. The team members, thus, could often not agree what the requirements meant and had major difficulties in reaching a common understanding.

The number of the requirements contained in the specification clashed with the staffing plan of the project. Clearly, to specify software for a thirty person-year project in a predictable manner, 120 requirements are not sufficient. The requirements also expressed wishes that were hardly realizable without doing extensive studies, for which there was not enough time. For example, it was unclear how to realize a requirement “not to build limitations into a system”. Finally, stakeholder conflicts had not been sorted out. Features were requested that were not realizable without
questioning investments that already had been done by the company and without questioning strategic partnerships of the company with other companies.

### 3.2 Evolution of Techniques

The team quickly recognized that designing software on the basis of such requirements was not possible. Hence, the team decided to improve the requirements and their understanding thereof before proceeding with the solution. In a sequence of steps, summarized in Table 2, different techniques were tried over a period of three months. Recalling that neither the team nor the stakeholders were experienced in requirements engineering, they followed a trial-and-error learning pattern.

<table>
<thead>
<tr>
<th>Method</th>
<th>Acquired Knowledge</th>
<th>Reasons for Success or Failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word-Processor Template</td>
<td>Incomplete, vague, and isolated requirement fragments.</td>
<td>Team without mental model of RE.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Shortcuts in RE process.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Insufficient help for inexperienced analysts.</td>
</tr>
<tr>
<td>Informal Stakeholder Interviews</td>
<td>Product components with roadmap.</td>
<td>Little RE experience.</td>
</tr>
<tr>
<td></td>
<td>Hardware configuration process with dataflows.</td>
<td>Too much heterogeneous information.</td>
</tr>
<tr>
<td></td>
<td>Project dependencies, cross-product data flows.</td>
<td>Focus on data collection at expense of analysis.</td>
</tr>
<tr>
<td></td>
<td>Some business processes and artefacts.</td>
<td>No guidance for inexperienced analysts.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No access to end-users.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Method not capturing all relevant kinds of data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lacking tool.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Method not capturing all relevant kinds of data.</td>
</tr>
<tr>
<td></td>
<td>Hierarchical model of business processes, user-roles, concepts, artefacts, software and hardware components, and locations.</td>
<td>Method adaptation supporting learning of RE.</td>
</tr>
<tr>
<td></td>
<td>Graphical user interface (GUI) prototype.</td>
<td>Stakeholders with experience in diagramming.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Low cost tool.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Method able to capture relevant kinds of data.</td>
</tr>
</tbody>
</table>

Table 2: Summary of techniques and results (RE: requirements engineering).

#### 3.2.1 Informal Stakeholder Interviews, Semi-Formal User Centred Modelling, and Use Case Modelling

At first, it was natural to interview stakeholders and record the results of the meetings informally. Documents of value to the project were exchanged upon request. The interviews were useful in completing and concretizing some of the fragmentary requirements. The system objectives and the system-level use cases could entirely be discussed and agreed upon. Some business processes could entirely be documented and related to associated artefacts. Some user-level requirements as well as the interfaces between software and hardware could be defined.

However, the discussions between the project team and its stakeholders were informal and often lacked concreteness. Consequently, the requirements remained mostly vague. Some chunks of requirements had major inconsistencies. The team was not able to formulate a single view that it could agree on with the stakeholders.

Due to their little RE experience, the team members only started to develop a mental model of what should be described by the requirements. The discussions typically were staying around requirements of secondary importance. The team could not make clear, which important questions were open and which stakeholders they
needed to discuss and agree the requirements with. As a result, the team was denied access to stakeholders like customer support and domain experts.

They team realized that stakeholder interviews were not sufficient for arriving at a usable requirements. Since important information was lacking in user-software interaction, they got advised by a person outside the company to look at Contextual Design [Beyer, 97], a user-centred elicitation technique. Contextual Design is based on five kinds of diagrams to describe actor responsibilities and communication, activities and intentions, artefacts, values, and physical locations of user’s work.

It turned out that Contextual Design could not be applied, because of three major obstacles: Contextual Design assumes that users can be observed. For the team, however, it was not possible to overcome the organizational barriers to get in contact with users. To use the models meaningfully, not enough appropriate information could be elicited from the stakeholders to whom the project had access. Contextual Design is centred solely on the documentation and analysis of requirements on the user level. It was not possible to consider other viewpoints like system objectives, business processes, data, and software-hardw are interaction. The size of the project required the support by a tool for modelling and managing the data that gets gathered and analyzed. A software tool supporting Contextual Design could not be identified.

To cope with the ever increasing amount of information to be considered, the team turned to UML [OMG, 03] use case and class modelling [Booch, 98], for which tool support is readily available. Unfortunately, the team failed to exploit these concepts for the following reasons. Neither the class nor the use case models are appropriate to model relationships between entities and the functional requirements, and use cases lack structuring mechanisms for describing business processes [Glinz, 00]. As a result, the team was unable to use these models for discovering user-level requirements on the basis of available data. That other UML diagrams would have been more appropriate was not realized by the team at that moment.

To comprehensively document and analyze the functional requirements, it was deemed important to integrate all abstraction levels from the system objectives down to user-software-hardware interaction (similar to the decomposition scheme of Structured Analysis [DeMarco, 79]). This was not achievable with UML use cases, because they lack appropriate hierarchical structuring mechanisms [Glinz, 00]. To the team, use cases did not represent a significant improvement for structuring functional requirements compared with the company’s TRS practices. Thus, there was the risk to run into problems similar to the ones outlined in Section 3.1.

### 3.2.2 Semi-Formal Business Modelling

The approach that finally enabled the break-through was Eriksson-Penker business modelling (EP) [Eriksson, 00], an extension to UML that was supported by the tool chosen in the preceding step. The profile is intended to study and improve business processes and to formulate requirements for information system support.

A subset of the EP diagrams was utilized. The conceptual model, a variant of a UML class model, defines key work-related concepts of the software users. The resource model, also a variant of a class model, defines business-relevant objects such as people, material, and products. The process diagrams, a variant of UML activity diagrams, describe user activities.
The selected EP diagrams were used to model the software requirements at all levels of abstraction, including system-level use cases, business processes, and user-software-hardware interaction. Traceability between the abstraction levels was established by the refinement relations nesting, aggregation, and sub-typing.

The team incrementally adapted the EP profile. Legibility of the diagrams was improved by changing the appearance shapes using UML stereotyping features. Data-flow relations were modelled more precisely by modifying the proposed EP stereotypes. An example of a tailored diagram is shown in Figure 1.

![Figure 1: Example of a tailored Eriksson-Penker process diagram.](image)

EP goals and goal achievements relations were replaced by process refinements. Two refinement principles were used: decomposition to split a process into parts and delegation for reuse of known processes. For legibility reasons, process refinements were shown by explosive-zoom that retained the immediate context of a process.

To derive software requirements from business processes, the suggested EP approach was not used. Instead, requirements allocation was defined with an “enable” dataflow between process and software component. Non-functional requirements and any other idea, question, remark, or requirement was recorded with UML comments or tagged values attached to the relevant model element or diagram. The project scope and preconditions and results of product use were colour-coded: red for elements out of scope, blue for assumptions, and green for results. The team added UML deployment diagrams to the EP models to describe scenarios of geographical distribution of users and systems – an idea coming from Contextual Design.

Finally, to mitigate the risk of stakeholders not understanding diagrams and to specify the look and feel of the software, a prototype of the graphical user interface was developed that showed the appearance of the tool during the described processes.

To manage complexity, tool support was a central concern. To overcome the bad reputation of UML tools, which for historical reasons were regarded as expensive tools with meagre drawing capabilities, a cheap and capable tool with support for concurrent modelling needed to be identified. With the help of Internet directories, Enterprise Architect [Sparks, 03] was discovered. That tool acted as an important catalyst to identify, tailor, and use the EP UML profile.

Based on early positive experiences with modelling data that was available at the time were the EP-based RE effort started off, the project team formed an RE core
team with the most skilled modellers to pursue the EP approach. It was this team that tailored the original EP guidelines.

With increasing experience, a set of patterns was defined that described common modelling constructs. By that, the team codified their knowledge of language use and defined conventions for representing common situations of varying complexity. In addition, analysis patterns were established for checking the model quality.

The model was used to test the team’s understanding of the requirements. Vague information was much more difficult to formalize than concrete data. Guessing was required to integrate available information into a consistent model. Gaps in knowledge and understanding became evident by the missing parts of the model.

Problems with the requirements were first sorted out team-internally, then in stakeholder meetings that had the goal of validating and completing the model. The first such meeting was a half-day workshop with local stakeholders. The participants were introduced to the graphical language with basic modelling, completeness checking, and refinement patterns. For the discussion of the requirements, all diagrams were posted on a wall.

The stakeholders were positively surprised that the team did not discuss the requirements in terms of solution elements and technology, but rather in terms and concepts relevant to the stakeholders’ daily work. No meeting participant had problems understanding the diagrams. During the discussion of the diagrams, the stakeholders either agreed or proposed changes. Gradually moving in the model from the system objectives down into details provided rich context and scoped the discussions and negotiations.

To improve the requirements quality, the team adopted an interview technique that was grounded on their analysis patterns. For vague requirements, stakeholders were asked how things shall work in reality. Every model element was checked for completeness by asking for information commonly related to the type of model element. Relations between the dynamic process diagrams and the static conceptual, resource, and location models were another basis for cross-checking the consistency and completeness of the requirements. When changes were proposed, the meeting participants followed the links between the model elements to study their impact. In cases of conflict, the team insisted on finding a solution acceptable for everybody.

Besides improving the requirements, the team also succeeded to make clear what stakeholders needed to be consulted in addition. Ten representatives of roles close to the users (domain experts, customer service, etc.) were invited to a second two-day workshop, where diagrams and prototypes were validated and interactively modified.

The two workshops and further discussions with on-site stakeholders yielded significant additions to the model. The team succeeded to complete the requirements to a reasonable level of detail and quality and got agreement on the requirements. Also contacts were tied that proved useful in the remainder of the project.

3.2.3 Results

The complete RE effort, covering all attempts, took about three months of calendar time and seven person-months of effort. One person was working entirely on RE, two people to a large extent, and the rest of the team irregularly. The Eriksson-Penker attempt accounted for about seventy percent of the total effort.
The final requirements model consisted of almost thousand model elements. On seven levels of abstraction, it contained approximately 160 activities, 230 components or artefacts, 180 concepts, 30 actors, 50 boundaries, 10 nodes, and 300 notes or constraints. These elements were connected with about 1300 connections. The need for effective tool support was evident to make the model manageable.

RE was performed not only with better results, but also in a much more focused and effective way than in previous projects. Based on their experience, the team members estimated that they would have needed three years for arriving at requirements of similar quality with the company’s standard IEEE 830-based approach and informal stakeholder interviews.

For concluding the RE work, the project’s TRS needed to be updated to reflect the model. Still, in the ensuing project phases, the model was the main tool used for requirements management. The requirements were reported to be quite stable. Two major incidents happened. Senior management, not involved in the meetings, wished reusing legacy components. And compliancy to a domain standard, which was not modelled in detail, yielded significantly more effort than estimated.

The model provided a first blueprint for the software architecture. The use of UML for modelling requirements enabled the team to relate UML models for architecture and design back to the requirements. The requirements model was used as one input for predicting the implementation effort. As the project team grew, the model was used to train new members.

The stakeholders were satisfied with the Eriksson-Penker-based method. Without big effort, they could understand product impact, influence the deliverables, and align the team’s goals with their own concerns. Hardly any stakeholder had difficulties to understand the diagrams in the moderated meetings: they had engineering background with a tradition in diagramming, and the modelled application domain was well known. Access to the diagrams was also enabled by the use of demonstrators. Conversely, understanding the model without facilitation proved to be difficult, making it impossible to distribute the model to the stakeholders for feedback.

Quality management made the Eriksson-Penker-based method and the results visible inside the company. The ensuing reaction was though not strong enough to institutionalize the practices.

When the experience became visible to ABB Corporate Research, a research project on RE was launched that aimed at further understanding and spreading the method. Success was limited. Maintenance projects had to remodel already implemented requirements – work that was not perceived as value adding. Also, it was too difficult to teach the language and method – available time was usually too limited and much success-relevant knowledge was not possible to transfer (similar to [Fairbanks, 03]). Together with former stakeholders, the project now investigates UML modelling approaches based on round-trip engineering [Henriksson, 03].

4 Discussion

There is a discussion ongoing about the factors influencing the adoption of development methodologies in practice. For example, it was argued that developer experience is an indicator for method use [Fitzgerald, 97]. The present case study
indicates that RE method adoption may be a question of struggle for understanding a project’s goals and problems to be solved. The team sought help from structured methods because of the team’s high level of uncertainty.

The selection of the right method was influenced by the boundary conditions of the project, the willingness of the team to experiment, and the team’s learning. Several approaches to RE had been considered, tried out, and finally rejected because of insufficient guidance, lacking user observation opportunities, lacking tool support, or insufficient support of requirements structuring and analysis needs. The team initially showed great openness to experiment by trying out methods according to their prescription. With the fifth approach, however, the team turned their method adoption strategy to method tailoring. This turn was motivated by the team’s increasing understanding of RE concepts and tactics.

The question which possibly matching method is selected is probably a matter of chance. The consideration of methods was influenced first by institutionalized company standards, then thoughtful action, suggestions from experienced colleagues, common industry practices, and finally by tool documentation. There was, however, a tendency to move from informal RE approaches to more formal and rigorous ones.

Method tailoring was only possible after getting experience with a few RE approaches and after learning of the project’s boundary conditions through method testing. The tailoring approach was much influenced by the team’s needs for requirements analysis and structuring, the tool capabilities, and constant striving for simplification of the modelling language and the model itself.

Many of the team’s insights were already well established in the RE community, but new to the team at that moment. For example, they rediscovered ideas from SA [DeMarco, 79] and SADT [Ross, 77]. The team also struggled with challenges to using UML for RE, which would have been readily documented [Glinz, 00].

Three major factors explain why only the final method did have success and only when it was applied by the discussed project team. The fit of the RE method to the project situation was crucial. It was important that the requirements language was able to capture required kinds of information like business processes, roles, concepts, artefacts, software, hardware, and locations.

Another success factor was the team’s mastering of the language. The team needed to understand the basic language constructs, language use to describe the application domain, and techniques to share the understanding of the language and to verify the model quality. This knowledge was partially achieved by inventing parts of the language and was reflected in the team’s body of modelling idioms and patterns.

The third major success factor was the understanding of how the language was to be used to achieve the goals of RE [El Emam, 95]. The team needed to understand how to use the model to integrate and structure the wealth of requirements-relevant information, how to elicit further information, how to validate the team’s knowledge, and how to align stakeholder expectations [Ovaska, 05]. The team did not have such knowledge easily accessible and learned it through intuition while doing their RE experiences.
5 Summary and Conclusions

The case study describes a project that went within three months through five attempts of understanding and documenting the requirements for a mid-sized new software product. With every attempt, the team learned lessons on requirements engineering and discovered new requirements for a selecting the technique that would be adequate in their situation. They finally discovered Eriksson-Penker modelling, which after tailoring provided an effective means to escape the requirements tar pit.

The study is presented as an exploratory case study that draws on first-hand experience, documentation, and interviews. The goal of the study was to provide insights into learning and adopting requirements engineering practices in industrial circumstances by describing the behaviour of an initially inexperienced team. Major factors affecting the adoption and tailoring of methods were discussed, and key reasons for successful method use were presented.

Further research should go into the direction of replicating the case study’s findings to further enhance the understanding of the influences to method adoption and method tailoring. A major open question is how methods that proved successful may be taught to new project teams that act in possibly changed circumstances.

The authors would like to thank the members and stakeholders of the project team and ABB Corporate Research for their assistance and feedback.

References


Session 2: Information Flow in Requirements Engineering

Learning to Tailor Documentation of Software Requirements
Eric Knauss, Daniel Lübke, Thomas Flohr, University of Hannover, Germany

The Requirements Engineering Gap in the OEM-Supplier Relationship
Christian Allmann, Lydia Winkler, Thorsten Kölzow, Audi Electronics Venture GmbH
Learning to Tailor Documentation of Software Requirements

Eric Knauss
(FG Software Engineering, University Hannover, Germany
eric.knauss@inf.uni-hannover.de)

Daniel Lübke
(FG Software Engineering, University Hannover, Germany
daniel.luebke@inf.uni-hannover.de)

Thomas Flohr
(FG Software Engineering, University Hannover, Germany
thomas.flohr@inf.uni-hannover.de)

Abstract: In software projects, it is important to determine the right amount of requirements documentation. If the documentation is not detailed enough, it is an insufficient base for contracts. If it is too long, it is expensive to maintain. Therefore the amount of documentation should be adjusted for each project. Often this step is omitted, partly to avoid the effort of tailoring, but mainly because project members do not know how to tailor or even are afraid of the consequences and associated risks. In this paper we share our experience in addressing both aspects with a mixture of organizational and individual learning. We successfully used our approach in university teaching and in parts with industrial partners.

Keywords: Documentation Tailoring, Experience Base, Agile Hour, FLOW, University Teaching

Categories: D.2.1, D.2.9

1 Introduction

Process-driven software projects requirements (e.g. the V-Model, [IABG, 1997]) aim for a very high documentation standard: Requirements should be fully and clearly documented and possibly traceable. This leads to very large documents which cannot be easily managed, used and maintained. In practice we encountered that this high standard is often the goal of management and quality assurance teams, but many attempts to reach that standard normally fail. For example, in one company requirements are written down but only after they were communicated to other stakeholders. This way documents become only a persistent memory and lose their role as a communication channel: It seems that the demanded theoretical standard cannot be reached in practice.

Therefore, “agile” methods, like eXtreme Programming (XP) [Beck, 2000], try to capture requirements differently: Requirements are orally exchanged between the customer and the developers. For example, in XP only small story cards containing some notes are used to support and document the communication.
However, neither approach seems to be appropriate under all circumstances. Participants in software projects need to learn to mix both approaches to requirements management in order to achieve a good solution which is well-suited to their project.

In university teaching this leads to the problem of presenting contrasting techniques and teach students the ability to combine these techniques’ strengths appropriately. Moreover, the experience to tailor and improve the development process can be valuable for experienced developers, too. They are normally used to and involved in highly process-driven projects and do not see problems and possible improvements associated with their approach. They can benefit from being confronted with contrasting techniques in order to think about their daily work.

For presenting the tailoring ideas this paper is organized as follows: In section 2 we show our approach for organizational learning applied to software projects in university teaching. Section 3 contains our experiences with individual learning based on so-called Agile Hours. The benefits of the combination of these two approaches are shown in section 4. Based on our conclusions in section 5 we suggest a new way to support the tailoring process of requirements in our outlook.

2 Organizational Learning in process-driven approaches in University

As part of our curriculum, students have to participate in a one-term software project. These software projects are organized as a simulation of a process-oriented company [Lübke, 2005a] in which different projects are being worked on by different teams. These teams are coordinated using Quality Gates [Lübke, 2004] which impose a certain development process by defining several phases, like requirements gathering.

For the requirements process, students get the following assistance:

1. Requirements Template: Students receive a template for their documents which they have to use to document the project’s requirements.
2. Checklists for Quality Gates: By this the students learn about the formal requirements for their specification.
3. Experience Base: Students get access to an internal experience base web-based tool [Buchloh, 2005] in which example documents, comments and experiences by older projects are provided and can be viewed and downloaded. Our experience base resembles the ones introduced at large commercial software organizations.

At the end of each project, all teams elicit experience. With a light-weight Post-Mortem (e.g. [Birk, 2002]) technique, the LIDs [Schneider, 2000] method, experiences are collected and written down in LIDs documents containing approximately 12 pages.

The LIDs session, in which all projects members are able to share their experiences and insights, is guided by a template. Everything which is talked about is instantly written directly into the LIDs template by a moderator. The document is visible all the time to the project members. This facilitates feedback and improves the discussion. The LIDs template contains following sections:
• Motivation: What was the motivation to participate? Is it a typical situation that will reoccur?
• Expectations and fears in advance of the project.
• The course of events from the project members’ point of view.
• What worked out, what did not?
• Description of the best and the worst moment during the project.

These experiences are used to refine the projects’ templates and checklists as well as to feed back examples, experiences and best practices to the experience base as is shown in figure 1.

Figure 1: The Experience Base arranges experiences around the process description.

In this way, the entire simulated software company learns. Consequently, we could observe that student teams improved from term to term: interview techniques
(e.g. using dictating machines) which proved to be useful in one term were used more frequently in following terms. Several changes to the templates have improved the overall documentation level. Furthermore the experiences led to changes of the software development process used.

The second way of learning takes place within our department. The feedback and experiences provided by the students are used to improve teaching of critical or not well-understood techniques and aspects of software engineering. Therefore, overall teaching quality has improved as well.

All techniques described above aim to improve the overall organization’s performance. The organization as a whole learns as it collects LIDs documents and additional entries in its experience base which can be used by teams and the process designers to better conduct and manage the next projects.

3 Reason for Reflection: Agile Requirements in Agile Hours

Within software organizations individual learning is as important as organizational learning beyond individuals: First-hand experiences by developers are more valuable to them and facilitate their own improvement ideas and implementations.

Especially, within organizations which employ the same processes for a long time, new aspects introduced by externals can break up old behaviour and lead to new improvements.

In this context, Agile Methods are often new to large software organizations. Especially for demonstrating problems within requirements management, e.g. contact to the real stakeholders, they provide an efficient way to bring up deficiencies because they are provocative and completely different than established methods.

As part of our curriculum and in cooperation with commercial software development organizations we conducted 18 so-called Agile Hours [Lübke, 2005b]. Agile Hours are simulations of small XP projects done within 70 minutes and a following discussion. They focus on customer interaction and XP-style requirements management and documentation techniques. Within a prototype phase and two iterations a small project is “developed” by drawing a product on sheets of paper or building it using Lego bricks.

The requirements are documented story card-like by the customers as one-line requirements as is demonstrated in figure 2. These story cards are used for discussion in the planning game. The planning game is a meeting of all developers and the customers in which the story cards for the next iteration are selected.

Because XP is very customer-oriented and the interaction with the customer is very direct, deficiencies in requirements gathering are normally uncovered. Especially, experienced software developers get new points of view: Playing a customer in a simulated project is often very helpful for recognizing and understanding the problems of conveying requirements between the different parties.

In any case, the relationship between the development and organization and the stakeholders can be discussed. The questions if the software organization knows the real requirements and who can be asked in case of problems with requirements documents are very important. Astonishingly, these questions normally cannot be answered by the participating developers.
All in all, Agile Hours are a very good way on the individual level to raise the interest in the problems of customer interaction and requirements management. Students and professional developers can benefit from attending an Agile Hour in order to improve their own behaviour related to requirements and customers’ demands concerning their real-life projects. The individual experience of the participants is the basis for the tailoring of established processes to more effectiveness.

4 Learning to Tailor Documentation

We imagine a spectrum that runs from strict processes to absolute agility as in figure 3 (taken from [Boehm, 2002]). Tailoring the documentation of software requirements
translates into finding the optimal point in this spectrum. For this reason learning to tailor is learning about the spectrum. Only if project members know about the alternatives they are able to choose the appropriate ones for the specific project.

Figure 3: The planning spectrum (taken from [Boehm, 2002]) runs from unplanned ad hoc development on the left to micromanaged milestone planning on the right.

In our experience people often decide not to tailor their process, because they are unsure about the effect of the tailoring measures. A good knowledge about the spectrum of possible techniques is the only way to reduce this fear. In the domain of software requirements documentation one part of this learning process is gaining experiences with different templates for requirement documents. In our opinion this experience should be gathered by organizational learning, because the individual normally cannot try out all the different flavours of requirements documentation.

One aspect of lightweight requirements documentation like story cards is that it relies heavily on customer interaction. Project members need experience in this area for which individual learning, for example in an Agile Hour, is much better suited.

The importance of a good mixture of individual and organizational learning for tailoring requirements documentation becomes also evident from a more process driven point of view. On the one hand, organizational learning can optimize existing processes but is not able to break up process driven thinking if required or beneficial. For example, the tailoring of templates alone will never lead to an agile approach.

On the other hand agile techniques can be beneficial to know even for process-driven projects: culture of stakeholder interaction can be a crucial success factor.

5 Conclusions

We often observe organizations that are very process-oriented. Even our students learn “heavy” processes before more agile techniques like XP are introduced. We experienced that process-driven approach is better to start with, because it provides more guidance for “novice programmers” like students. The general direction of tailoring seems to be introducing more agile concepts into existing processes: The
organization has to learn to use as few processes as necessary in order to be as agile as possible because no unnecessary work is done. This especially holds true for requirements documentation.

The problem with too extensive documentation of requirements is well known (e.g. [Cockburn, 2001]): If there is too much documentation, it will not be read. At the same time it prevents developers from asking their customers, because they are unsure, if the answer to the question has not been written down already and the documentation can be a barrier between the customer and the developer over which only indirect, error-prone communication happens. Furthermore large documents are hard to maintain. Organizations have to learn the right amount of documentation in their specific context.

In this paper we pointed out two ways to help with this tailoring process. Within our software projects we have successfully established a learning environment using traditional approaches like experience bases combined with modern teaching like Agile Hours. On the one hand a slow but continuous learning takes place. Checklists are modified, templates are adjusted to the right degree of freedom and more generally the important and useful parts of the process are identified.

On the other hand we make use of Agile Hours as a foundation for a discussion that often leads to new insights. Even if this does not cause a shift to a more agile approach, it does support the tailoring process. In our experience software organizations benefit from this kind of organizational combined with individual learning. The resulting experiences have already been used with commercial partners.

6 Future Work

As part of our research we are now looking into analyzing the way requirements are being passed within different project settings using our FLOW notation [Schneider, 2005]. Figure 3 shows the Planning Game (a XP practice introduced by [Beck, 2000]) in this notation.

![Figure 4: Planning Game in FLOW-Notation.](image)

Note the focus on information flows that allows us to model how the resulting documentation (a sorted stack of story cards) is enhanced by communication between
customer and programmer. In this example the solid arrows represent document based information. The input of the planning game is the requirements which were written down to story cards. If read again a story card will produce the same information.

Currently we try to establish the analysis of processes with means of this notation as a third source for tailoring. We already observed certain patterns that become visible when processes are displayed in FLOW and point to problems in a process (like documents that are never read). But FLOW might also be useful in planning where to introduce more “dashed lines” into a given process, for example by giving feedback to the customer at defined points. FLOW’s aim is to offer a foundation for tailoring by giving more aspects to the ones responsible for the project: existing and new experiences as well as direct communication and document-based communication are considered.

References


[Lübke, 2005b] D. Lübke and K. Schneider, *Agile Hours - Teaching XP skills to Students and IT Professionals*, Oulu, Finland 2005b


The Requirements Engineering Gap in the OEM-Supplier Relationship

Christian Allmann  
(Audi Electronics Venture GmbH, Germany  
christian.allmann@audi.de)

Lydia Winkler  
(Audi Electronics Venture GmbH, Germany  
lydia.winkler@audi.de)

Thorsten Kölzow  
(Audi Electronics Venture GmbH, Germany  
thorsten.koelzow@audi.de)

Abstract: The OEM’s requirements engineering and management process is affected by many residual encumbrances and future constraints. The encumbrances’ origin comes from the OEM’s strong dependency on suppliers’ software development knowledge and support. In contrast, future constraints are dealing with the assembling of software engineering knowledge, especially requirements engineering, as OEM’s core ability. On this account reducing the OEM’s knowledge gap demands tailored engineering processes regarding the different kind of process knowledge, distributed development environment, changing project responsibilities, available but unstructured knowledge bases and different educational members’ background. This report examines main challenges in OEM’s development process to overcome the past OEM-Supplier relationship achieving common development in partnership. For clarification, three process flows are presented showing in some kind the historical evolution of the OEM-Supplier relationship. Based on the presented vital process constraints requirement methods, tools and project management guidelines are derived.

Keywords: Requirements Engineering, Automotive Software Engineering, Project Management, Knowledge Management

Categories:

1  Requirements engineering in the automotive industry today

The development and production of modern cars are affected by a strong relationship between OEM (original equipment manufacturer) and its suppliers. Due to the increasing importance of software in the automotive, both OEMs and suppliers have established a requirement engineering process to stay abreast of changes. The OEM’s requirements engineering process is actually tailored to the management and the integration of certain supplier products [Huhn, 03]. Therefore the OEM’s goal is to provide information about the system environment into which the supplier product must be integrated. The integration of supplier products itself demands a precise communication interface between OEM and supplier as well as between the suppliers themselves.
2 The ideal development process using the FLOW notation

To emphasize the challenges of the development, especially the requirements process, we present three different project flows using the FLOW notation [Schneider, 05]. The focus is to show the different communication channels and interfaces between the stakeholders and the developed documents, as well as the difference between the stakeholders and OEM’s experience bases.

In our opinion the FLOW concept developed by the University of Hanover is well suited to model the complex project process flows with respect to the OEM-Supplier relationship. FLOW is a graphical notation to visualize the information flow in a project by modelling direct communication channels between project artefacts and project members. For the presented examples, only a subset of the basic notation is used:

- **document symbol**, representing persistent data (e.g. requirement documents written in Word or DOORS)
- **face symbol**, representing project stakeholders (e.g. requirement engineer, project leader)
- **black arrow**, representing the information flow (e.g. review meeting)
- **solid line**, information from a persistent information source (e.g. written document)
- **dashed line**, information flow from a non-persistent information source (e.g. project member)

![Figure 1: The ideal process flow](image)

The first example, shown in figure 1, represents an idealized development process flow. On the left hand side the two face symbols represent an arbitrarily number of stakeholders employed by company A. These stakeholders are the main experts in a company’s business for who a tool has to be built. Company A places an order with Company B for the tool development.
In our idealized view, neither the company nor the stakeholders have any opinion how software is developed. Thus, the first essential development step is the elicitation of requirements the stakeholders have, carried out by the requirements engineering team of Company B. As figure 1 shows the requirement engineer manages the elicitation process by asking the stakeholders and writing the requirements down. The resulting document is the first persistent artefact in the ideal process flow. Based on the document, ongoing developing activities are building up, see figure 1.

On the supposition that the created requirements document is well structured and the requirements quality attributes (e.g. atomicity, consistency, unambiguousness, etc.) are fulfilled, the design engineers, and later the programmers, have the chance performing their work with the help of the company’s knowledge base. The knowledge base symbolizes the collected experience of the company’s completed projects that can be used to perform their daily work.

### 3 The relationship between OEM and supplier

The OEM-supplier project example, figure 2, shows the past and in parts the present relationship between customers and contractors. Compared to an ideal process flow, figure 2 shows some important differences to the described flow in figure 1. Just like in our first example, the OEM (resp. company A) assigned a supplier (resp. company B) developing software system.

![Figure 2: The OEM-Supplier process flow](image)

The most conspicuous distinction takes place on the left hand side. The generalized category stakeholder, like it is used in the ideal process flow, is broken down to form a distinct command structure. The exemplary command structure, shown in figure 2, represents a decision-making process starting by the OEM managers up to the department engineers. The depicted form of information exchange points out a rather
Proceedings, LSO+RE 2006, Version 1.0

A generalized version of the real process flow. Nevertheless, it is important for the understanding and all further derived challenges that a written description of the information flow is only kept at one single point, the responsible department engineer. Although all relevant managers or engineers have their own vision of what the system has to perform or not, only the responsible department engineer interprets the system’s requirements and writes them down. Therefore, the engineer represents a bottleneck for all requirement activities, although he is in many cases not skilled to perform requirement processes or techniques. All decisions made are based on experience he has gathered during his work life. In the past nobody would check the requirement document with respect to quality attributes like consistency, clarity, ambiguity, etc. because nobody has the experience to give evidence about the quality, especially the consistency. Generally getting feedback, the engineer presents and interprets his written text to the managers and engineers allowing a judgement about the system’s functionality.

The responsible engineers cannot be fully blamed as the exclusive cause for the bottleneck. On the one hand, the engineer does not have the required experience and education to fulfill all expectations, on the other hand the project time pressure allows no further education, and adequate OEM knowledge bases on requirements engineering do not exist. In summary, all mentioned aspects lead to different quality in requirement documents. Therefore, we called the requirements document a functional draft. The functional draft is the basic document inviting offers. Concerning the statements above, draft and engineer represent a weak source for potential suppliers building up a system’s understanding, they preferred. Managing functional drafts ranging in quality and detail has a deep impact on the supplier’s side. First, the supplier cannot estimate the development cost and time resulting in imprecise offering. Especially in the field of driver assistance systems some suppliers profit from the generalized requirement description and sell their own existing system - perhaps still developed for a competitor- without any further adaptation.

Thus, the suppliers have a main interest in requirements engineering because they must manage OEM’s requests and changes and perhaps have the chance to sell the developed system to another company. Accordingly, comparing figure 1 and 2, the smallest changes to the ideal process flow occur on the supplier’s side. Unlike the central knowledge base of company B, the supplier’s bases are more sophisticated with respect to product line development. The knowledge base assists the supplier’s requirements engineers performing the preparation of the requirements document by the usage of suitable templates, document structures and tools. In summary, the requirement process is oriented to perform requirements analysis and validation regarding the specific product line information.

4 Future directions of the requirements engineering process

The OEM’s disadvantages arising from the mentioned flow of figure 2 can be read out very easily. The knowledge about the integrated product (e.g. a new driver assistance system) is kept by the supplier which results in a strong dependency on the supplier concerning the integration of additional product features. Thus, the supplier
can easier dominate the price for new functionalities. For the OEM, this entails the loss of two main abilities: the knowledge about the integrated product (component protection) and the possibility to negotiate with different suppliers about new product feature prices. To avoid these main disadvantages, the OEM must participate in the requirements engineering process and perhaps take a leading role developing product parts on his own.

On this account the OEM must achieve three goals:

1. the reduction of knowledge loss about his integrated systems
2. the installation of own knowledge bases by means of collecting, verifying and integrating all existing company’s know-how
3. a progressive project involvement with regard to proprietary-development

---

**Figure 3: The potential future OEM-Supplier process flow**

All three goals have one main challenge: a *seamless transition* into daily work. The distinctions and resulting changes in the process flow are shown in figure 3.
The observable difference between figure 3 and all figures before is the omission of a clear separation between OEM and supplier part. Although the left hand part of figure 3 shows no real difference to the previous ones, a subtle distinction still exists – greater project responsibility of the department engineer in charge. One of the mentioned confusions of the OEM-Supplier process flow was the unbalanced relationship becoming manifest in the functional draft. Irrespective of the document quality and content, the supplier takes full responsibility for the success of the project. Establishing proprietary-development demands distributed responsibility like it is exposed in figure 3 by means of single workpackage leader. The scenario shows one big chance for OEM and supplier that is likewise the obstacle. Both can concentrate their efforts to their core abilities. In the case of driver assistance systems this could be the design of the sensor components and image processing as supplier’s part and advanced algorithm and functionality, e.g. lane detection as part of the OEM. The obstacle would be to manage the distributed responsibility with respect to the common goal, in our case the OEM driver assistance system.

For future development in highly innovative fields, like hybrid technologies, x-by-wire or sensor fusion, working in partnership is inevitable. The resulting distributed process flow is very sophisticated to organize and to lead. The described process flow shows only the first part of a potential future development cycle but illustrates the main constraints. Common development means sharing knowledge, the knowledge about the actual developed system as well as the project experience companies have collected over the years.

As a consequence of the different development “philosophies”, a clash of interest occurs. Now the job of the project leader is to establish some kind of cooperate document (generally called project manual) to organise the different development activities, determine the development chain and OEM’s and suppliers’ fields of responsibilities. Although figure 3 shows the existence of knowledge bases on both companies’ sides the usefulness of the contained information for partnership development must be questioned. Neither OEM nor supplier really has any experience in this kind of development because partnership means publication of at least essential and confidential information. For the requirements engineering process, this implicates common:

- requirement process including elicitation, analysis and validation activities
- tool framework for distributed development
- exchange of partial development artefacts
- development and process guidelines
- definition of partnership responsibility

*Time-shared* and *artefact-shared* development is mandatory concerning the effort for system’s development under the regular time pressure. All met implications can be subsumed to one essential point: a common development platform. To avoid misunderstandings, the development platform comprehends solutions for arising communication overhead, distributed access to the different knowledge basis, project
management and tools. The realisation and establishment of this kind of partnership development demand some requisites on the OEM’s side such as partially described in the following section.

5 OEM management and engineering constraints

5.1 User constraints

It must be kept in mind that although a lot of mechanical and hydraulic systems are replaced by mechatronical systems, the engineering team in charge for these components cannot be replaced in the same way. The OEM’s departments are mainly staffed with mechanical and electrical engineers with associated educational background. In replacing the mentioned systems by electronic and software these engineers not only have to consider the development of the usual hardware components, but also have to take care of the electronic devices and the integrated software. The impact of this evolution is that the engineers often do not have the necessary engineering background to evolve/maintain such systems effectively and efficiently.

With respect to figure 3, the partnership provides an opportunity to coach the engineers on the job. Coaching OEM’s engineers on the job by their project partners helps in finding a common discussion platform. Nevertheless, this know-how transfer can only be performed within project’s limits. Hence, the goal must be to establish a requirement engineering process and workflow that is on the one hand tailored to the development process and on the other hand tailored to the engineers’ abilities. Tailoring a process with respect to a special user group is necessary because nobody – as the experience in our company shows – will adhere to a process if the users can not identify with it. User constraints to process definition are:

- intuitive application handling (benchmark Microsoft Word)
- simple and seamless tool chain
- advised request and change board
- support team for process, methods and tools

5.2 The establishment of an information knowledge base

The OEM’s requirement engineering process currently often lacks the knowledge of how to perform requirements engineering effectively and efficiently. The experience, gained during the development process, must be retained by building up a knowledge base for future developments. Gained experience could be workflow information, description patterns, review documents or something else affecting the requirements process. Although in most organisation standardised document templates still exist, the practice shows that project concerns, topics, time pressure or quality gates demand reengineering of the common templates. That given statement comprised an interesting point namely the information on company’s exhibit engineering knowledge. The drawback, the information is scattered over the different company’s departments. On this account, the company’s departments reach a first evolution step
by adapting the common requirement standards to their personal needs. Unfortunately, the conclusions that are drawn, have no impact to company’s overall engineering evolution because no feedback loop is implemented. For us, the information knowledge base is a platform to evaluate the concerns and constraints of the different evolution steps towards more applicable process flows, methods and tools. Besides this “process” information the knowledge base is also a platform to assemble product information such as performance or dependability analysis results, which can be used to obtain judgements about development risks at an earlier stage. Risk and cost estimation for system development is as important as new technologies are evaluated and integrated in all business areas.

5.3 Automation of user constraints using the example of traceability

The following example describes the traceability problem and shows how important automation is with regard to requirements engineering. It is assumed that usual requirements documents consist of 200 to 600 pages [Heumesser, 04] with 12 requirements per page. Let us assume that 400 pages would result in 4800 listed requirements. The management of these requirements demands to trace each of them, linking each requirement to associated requirements or test cases to validate the implementation against them. Concerning the example above, the linking of each requirement to respective test cases costs 30 seconds working time each time the link must be changed. The resulting working time by 4800 links amounts up to 40 hours respectively 1 week of work. This fictive calculation shows that although there is a need for linking requirements to refined requirements or test cases the resulting work time increases tremendously.

The actual chapter has a strong correlation to the section of user constraints. Most engineers in our company, affected by requirement engineering do not have the time for testing or evaluating new approaches or tool add-ons the company’s engineering team prescribes. These engineers at the grass-root level use what is practicable for the actual project needs. Regardless of traceability advantages for them, it is more time consuming. Bridging the gap between user prejudices and required engineering steps, the engineering rollout can only be done incrementally. Regarding test case derivation, generation and maintaining of requirement traceability in documents is essential to achieve a consistent system view. A consistent system view is as important as a distributed development environment. Both constraints must be kept in mind, especially concerning the single workpackage leader’s responsibility (see figure 3). From their point of view, they can only observe the quality of the produced requirement document part (1..n). The question must be answered who guarantees the quality, mainly consistency, of the whole requirement document even if requirement changes have an impact of more than one document part. Therefore, the definition of the development platform and process must guarantee in each case a consistent system view.
5.4 Integration of formal notations

Unambiguous, correct, complete and understandable requirements documents of complex systems cannot be produced and administered by OEM members themselves in the role of requirement engineers, due to the mentioned user constraints (see chapter 5.1) as well as time, cost and capacity constraints. They need the support of adequate methods and tools. At this, the basic idea is to guide and support OEM’s engineers to write high quality (clear, complete, correct, understandable and testable) functional requirements by means of a requirements specification template and/or predefined requirements’ patterns.

Using a specification template can be a first step in obtaining more precise requirements, as it is shown in the following example. In the requirement

*If the ignition is on and the engine is not running or the ABS module signals a defect, the display module shall switch on the ABS control lamp.*

it is neither clear which conjunction is dominating nor if inclusive or exclusive ‘or’ is meant. Therefore, the requirement can be interpreted in several ways:

* If (the ignition is on and the engine is not running) or the ABS module signals a defect, ...
* If the ignition is on and (the engine is not running or the ABS module signals a defect), ...
* If either the ignition is on and the engine is not running or the ABS module signals a defect, ...

Applying mathematically logical operators according to the specification template defined by Chris Rupp [Rupp, 04] can easily avoid these kinds of misinterpretations:

*If (the ignition is on AND the engine is not running) OR the ABS module signals a defect, the display module shall switch on the ABS control lamp.*

Using a specification template can help to obtain well-structured, understandable and clear requirements, but it cannot guarantee that for instance requirement 127 conflicts with requirement 305 in a specific or even worse in two different but interdependent documents.

On this account one idea is to refine the specification template suggested by Chris Rupp by means of pre-defined requirements’ patterns. At this, the definition of a requirement is carried out in several phases. First of all application specific inputs and outputs must be defined in a glossary. If not all inputs and outputs are known initially, there is also the possibility to expand the glossary during the creation of the requirements specification.

Secondly the main structure of the requirement is selected:

* If … then …
* Only if … then …
* After … then …
In the third phase the conditions, like “the ignition is on” and the actions or processes, like “switch on the ABS control lamp”, are specified. At this, the nouns of the conditions and actions are derived from the glossary mentioned above and the verbs are selected from a pre-defined catalogue of automotive specific process words.

If requirements only consist of nouns, verbs and part-sentences which are defined in a global database it’s easy to implement specific search routines. Therefore it is for instance possible to display all requirements corresponding to a particular input signal. This again supports the requirements engineer in locating problems between requirements in adequate time and with adequate effort.

Once a controlled natural language exists, a further step could be to map the pre-defined requirement patterns to formal/mathematical representations to obtain a formal requirements specification document in the end. Engineers can specify requirements in their familiar but restricted natural language, whereas the advantages of formal specifications and the basis for their automated support are available at the same time [van Lamsweerde, 00].

6 Future Work

Currently our scope is to implement the mentioned knowledge base and to establish a change control board to evaluate user constraints and project requirements. The change control board has the mission to evolve a so called “requirement engineering kit” tailoring the requirement process to our specific project needs with respect to product quality, user experience, time and money.

References

[Heumesser, 04] N. Heumesser, F. Houdek, Experiences in Managing an Automotive Requirements Engineering Process, 12th International Conference on Requirements Engineering, 2004, Kyoto, Japan

[Huhn, 03] M. Huhn, P.-M. Hoffmann, M. Mutz, Digitale Lastenhefte für die Softwareentwicklung vernetzter Steuergeräte, 23. Tagung Elektronik im Automobil, June 2003, Stuttgart, Germany

[Rupp, 04] Ch. Rupp, Requirements-Engineering und –Management, 2004, München, Germany

[Schneider, 05] K. Schneider, D. Lübke, Systematic Tailoring of Quality Techniques, 3rd World Congress of Software Quality, 2005, Munich, Germany

Panel: The Role of Learning in RE

Learning from own Experience and Advices in Literature
Andrea Herrmann, University of Heidelberg, Germany

Understanding Requirements through Filtering, Negotiating and Shifting
Päivi Ovaska, South Carelia Polytechnic, Finland
Matti Rossi, Helsinki School of Economics, Finland
Kari Smolander, Lappeenranta University of Technology, Finland

Additional Contributions by
Frank Houdek, DaimlerChrysler Ulm, Germany
Erik Kamsties, Carmeq, Germany
Learning from own Experience and Advices in Literature

Andrea Herrmann
(University of Heidelberg, Germany
herrmann@informatik.uni-heidelberg.de)

Abstract: The saying goes: “A fool learns from his own mistakes, a wise person learns from the mistakes of others.” Of course, we also want to learn from our own successes and those of the others. The requirements engineering literature is full of Best Practices and other advices. But how to see the forest instead of the trees? How can own experience and experience of others be used in an integrated way for learning in RE?

Keywords: Knowledge Management, Requirements Engineering, Best Practices

Categories:

1 Introduction

Requirements engineering (RE) is an activity or sub-project taking place in an early phase of a project, and its product is the requirements document which is the basis and input for later activities. Therefore, it is not surprising that among the most important reasons for project failure, requirements engineering issues appear in high proportion, for instance among the “CHAOS ten”, i.e. the ten major project success factors, in the CHAOS study of the Standish Group [Standish, 00]. As Glass [Glass, 02] and other authors emphasize, in software engineering the same errors are made again and again. Learning in requirements engineering therefore is important. Learning in RE can mean the reuse of requirements or refer to the RE process. Here, only the requirements engineering process will be treated.

“Learning from success” means to search for Best Practices (also called “critical success factors”) which significantly raised the probability of success of real projects. Equivalently, “learning from failure” means to identify risk factors which significantly raised the probability of failure of projects. But listing Best Practices and risk factors will not be sufficient for learning from experiences. Projects are different, some practices contribute more strongly to project success in some cases and less in others, e.g. depending on the type of project risk [Couillard, 95], or on the project complexity [Shenhar, 96]. Some Best Practices only have a positive effect when applied in combination with others [Ebert, 05], [MacCormack, 03].

Therefore, we here propose an approach for combining both types of data (data about own projects and documented project experience of others) for better learning in RE. The two sources provide us a different quality of data. From our own experience, we can gather detailed data from projects which take place in our own working environment. But the number of projects will be limited. We also get an intuitive feeling of what is important, and this feeling can be supported by data. On the other
hand, literature contains data about many projects, but the level of detail is usually low, data not complete and the context of the projects maybe totally different from our own. Therefore, the external experience from literature sources must be evaluated by the requirements engineer according to her experience. The approach presented here is based on a project model which can be used for doing lessons learned analyses of projects and for classifying experiences from literature. It is presented in section 2.

2 Model for Project Data and Experiences from Literature

A meta model has been developed which shall serve for gathering project data as well as for classifying experiences from literature.

2.1 Project meta model

We developed a project meta model. It will not be presented here in detail, but only its main characteristics. It is keeping as close to the ideas of the PM BOK [PMI, 03] as possible. A project is modelled by activities and deliverables. Each activity is described by the resources used by it, constraints (e.g. constraints on resources), and principles. Project deliverables are characterized by their content, properties and sub-goals (e.g. wanted properties of the deliverable). Also data concerning the whole project (input and output) are gathered, like the goals of the project, its resources and constraints, risks, and results. Meta data for each data set or case study like author, date of object creation, and change history are also important.

2.2 Gathering data from our own projects

The project data from our own projects concern one specific project. The context will probably be similar for them, and it is totally known. Also, one knows exactly what was done during the project (activities performed, deliverables produced), which resources were used etc. Our model has been applied to half a dozen case studies so far and was able to model the relevant data.

2.3 Classifying experiences from literature in a Best Practices map

In literature, one finds others’ (often intuitive and subjective) project experiences or summarized project data. They are of different quality and reliability, apply more or less well to your own environment, and data is lacking. It makes no sense to try to reconstruct a case study project from the fragmentary data contained in the literature source. But usually, the project experience in literature is formulated in form of Best Practices or advices. They tell you which activities to do how and which deliverables to produce how, if you want to be successful.

Therefore, the project model captures others’ experience in the form of Best Practices. It was necessary to categorize them, for identifying doublets and for attributing them to activities and deliverables of real projects. We use the following product groups:

- project definition, contracting & planning
- Project Management, Reporting and Controlling
• Requirements Engineering
• Design
• Realisation
• QM & Testing

Within RE, the following activities and deliverables were defined (in a way to structure the Best Practices we found):

<table>
<thead>
<tr>
<th>Activity</th>
<th>Deliverable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements Elicitation</td>
<td>Requirements Document</td>
</tr>
<tr>
<td>Requirements Documentation</td>
<td>system test cases</td>
</tr>
<tr>
<td>Requirements Analysis</td>
<td>Requirements Document, Analysed Version</td>
</tr>
<tr>
<td>Requ. Management Definition</td>
<td>Requirements Management Policies</td>
</tr>
<tr>
<td>Requirements Management</td>
<td></td>
</tr>
<tr>
<td>Requirements Validation</td>
<td>Requirements Document, Validated Version</td>
</tr>
</tbody>
</table>

This classification leads to a Best Practices map where each activity and deliverable is attributed advices concerning their resources, properties, principles, etc.

For evaluating and weighting the Best Practices, it is important to add some information which is usually contained in the literature sources:

• Do the data refer to one or several projects? How many?
• Is the advice based on a statistical analysis of objective project data or based on subjective impressions?
• Which success criteria were used for defining this Best Practice? (Remark: They can be very different and can not be covered by the classical “meet budget, schedule, and specification”.)
• In which context did this Best Practice help improve the satisfaction of these success criteria? (The context can be characterized by: project duration in months, average team size in number of persons, company size, customer-specific or market-driven project, internal or external project, technological uncertainty, business sector. These are project characteristics which in empirical studies of project data have been shown to make a difference concerning the effectiveness of Best Practices.)

This information helps to evaluate the reliability of the Best Practices for your own context. We advice that the requirements engineer adds to the Best Practices the following information according to her own experience:

• number of projects where it has been applied by yourself
• number of projects where it has been applied successfully
• wanted effect of the Best Practices (not effect on success criteria, but for instance on which property of which deliverable)
• real effect of the Best Practices (positive and negative)
• When the Best Practice has not been applied: Why not?
• When the Best Practice did not have the wanted effect: Why not?
• Classification into “basic, intermediate and advanced” (see Sommerville and Ransom [Sommerville, 05])
This evaluated Best Practices map can now be used for planning RE activities (by the project manager or the requirements engineer), defining acceptance criteria, and in lessons learned analyses for identifying potential of improvement for critical activities and deliverables (by project managers, requirements engineers or managers responsible for process improvement). One can not say that the Best Practices map is the model of the ideal project, as some Best Practices contradict each others, and some might be irrelevant for some specific context.

So far, we have analyzed ten sources concerning RE practices and 19 concerning project management Best Practices. We extracted 120 RE Best Practices from them, which were grouped according to the project model.

2.4 Information which might be interesting to the requirements engineer for planning and performing her activity

Concerning the information which might be interesting to the requirements engineer for planning and performing her activity we propose the following, which can be extracted from a data base of her own project data:

- Knowledge about critical activities and deliverables for focusing attention
- Knowledge about effective and uneffective Best Practices in her own context, for prioritisation
- Knowledge about probability of and damage caused by risks
- Knowledge of project decisions, their rationale and consequences, re-evaluation of the decision criteria after the project (to see whether the decision would be taken again next time)

3 Summary and Further Work

This paper discusses the reuse of process knowledge in RE. Another topic would be the reuse of requirements, which is not treated here. In this work is discussed how a project model can be used for structuring and using project experience from literature and own project experience together. The model described above is work in progress. We so far classified 120 RE Best Practices and experience from some case studies. The model was able to model all data. The usage of these data within projects has not been performed so far.

Another next research issue will be the knowledge management process, i.e. where such knowledge is produced and needed. The data described above is gathered by a literature search and an evaluation according to one’s own experience on the one hand and lessons learned analyses of own projects on the other hand.

Many information is produced after it is needed, like knowledge about efficiency of Best Practices within a certain context. Therefore, reuse of knowledge from former projects is essential.
References


[Ebert, 05] C. Ebert, J. De Man J, Requirements Uncertainty - Influencing Factors and Concrete Improvements, in Proc. ICSE, 2005, pp.553-560


Understanding Requirements through Filtering, Negotiating and Shifting

Päivi Ovaska
(South Carelia Polytechnic, Finland
paivi.ovaska@scp.fi)

Matti Rossi
(Helsinki School of Economics, Finland
mrossi@hse.fi)

Kari Smolander
(Lappeenranta University of Technology, Finland
kari.smolander@lut.fi)

Abstract: Traditional requirement engineering approaches pay little attention to how the requirements are interpreted and shared by different parties in an organization. Our study suggests that requirement shaping during a project can be described as a process where attitudes and expectations are filtered, shifted and negotiated repeatedly. We studied a large e-commerce platform development project and observed that preconceptions, attitudes and expectations about systems development among project participants filtered the understanding of software requirements, negotiating between project participants resolved the issues caused by filtering and shifts in these attitudes and expectations facilitated changes in the understanding of requirements. The study provides a new interpretation of how technology is collectively interpreted in organizations.

Keywords: Requirements engineering, requirements understanding, socio-cognitive processes
Categories: D.2.1, K.6.1,

1 Introduction

Studies on large software projects show that they fail at an unacceptably high rate. Even if a software project is completed in time, it produces software which is of heterogeneous quality and often exceeds its budget. Many of the problems encountered during systems development are attributable to shortcomings in the software product’s requirements. The problems reported in earlier research on requirement engineering typically involve the following issues: insufficient user involvement, ambiguous, changing and incomplete requirements.

Intensive research on software tools, modelling methods and processes for performing requirement elicitation has not yet delivered tools or techniques that could guarantee foolproof success in software projects. Traditional requirement engineering (RE) approaches offer poor understanding of how to specify and manage requirements for large evolving systems, how the requirements are understood and what kind of problems exist in the commercial practice. Recent research directed into the social and organizational processes in requirement elicitation sees it as an emergent political process [Bergman, 2002] constructed through conflicting interests and agendas, resource constraints and political influences or as a socio-cognitive
problem solving process [Orlikowski, 1994; Davidson, 2002]. This socio-cognitive problem solving process is characterized by differences and repeated shifts in assumptions and expectations of technology, which disrupt the project participants’ understanding of requirements.

2 Research subject

We studied, using a grounded theory approach a large-scale e-commerce platform project with internal customers within a large telecom operator. In the beginning of the study we observed that the requirements did not stabilize during the project but instead kept changing, causing problems and delays for the project. This led us to study how the software requirements were shaped and interpreted during the project.

The study was carried out in a software development department of an international ICT company. The development of new applications and services was assigned to an in-house software development unit (Internal Development Unit, later referred as IDU) or outsourced companies. The use of IDU for development of new services was mandated by the company’s top management. IDU had approximately 150 employees that had formerly focused on R&D work in the company. During the past few years it had tried to improve its software skills and processes in order to make its development more effective and also to prove its capability to other business units. All the business units of the company did not agree with IDU’s processes and did not trust in its software development capability. Their attitudes towards IDU competencies in software development were quite suspicious, mainly because of IDU’s history as an R&D department. Quite often business units preferred outsourcing instead of developing in-house.

The business owner of our case study project was the business unit (later referred to as BU) responsible for marketing new business ideas related to electronic and mobile commerce services. The project developed a mobile commerce service platform. The system was intended to enable organizers or their sponsors to promote their products in all kinds of events, such as ice hockey and football games. The system was composed of two subsystems: the platform in which the services were running (Platform subsystem) and the toolbox, which allowed adding, configuring and simulating these services (Tool subsystem). This toolbox was intended to run in a Windows PC and the service platform in UNIX environment.

3 Findings

The project highlighted problems in current approaches to requirement elicitation and systems development in general, which still largely assume that projects proceed with distinct phases and more or less in a waterfall fashion from a vague understanding of the idea of the system into a concrete system. Instead, we observed that the requirement understanding was filtered by project participants in the beginning of the project, negotiating between project participants resolved the issues caused by filtering and shifts in these attitudes and expectations facilitated changes in the understanding of requirements. This process of filtering, negotiating and shifting can be described as an ad-hoc and iterative process where the software requirements unfold during social interaction, communication and negotiation between parties.
During the analysis, we observed four categories of attitudes and expectations that affected the understanding of requirements of various project participants. The identified categories were:

- **Business value of system development**, i.e. the attitudes and expectations about the relationship between business and system development.
- **System development strategy**, i.e. the attitudes and expectations about the suitable system development life cycle model and processes.
- **System development capability**, i.e. the assumptions and expectations about competencies in different areas of system development, such as user interfaces and databases.
- **System development resource allocation**, i.e. the assumptions and expectations about scheduling, budgeting, and priorities of systems development projects in time-to-market pressures.

Within these four categories, we identified a process of stereotypical “tensions” that had important effects on how the project participants understood requirements in different phases of the project. We named these tensions as:

- **Filtering** that occurred when a stakeholder of the development process left something out of the scope because of his/her understanding, attitudes, expectations, or experiences
- **Negotiating** that tried to resolve the incongruence between stakeholders.

<table>
<thead>
<tr>
<th>Tension</th>
<th>Filtering</th>
<th>Negotiating</th>
<th>Frame shifting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business value of system development</td>
<td>Customer’s view of developers as a technical resource only.</td>
<td>Emphasis in business vs. in technology among the parties. The decision of outsourcing vs. in-house development</td>
<td>The change of developers’ role in the process.</td>
</tr>
<tr>
<td>System development strategy</td>
<td>Strict reliance on organization’s processes</td>
<td>The use of waterfall model vs. iterative/interactive way of development</td>
<td>Changes in system development towards more iterative development</td>
</tr>
<tr>
<td>System development capability</td>
<td>Avoiding UI software development because of the lack of skills and experiences</td>
<td>Negative attitudes towards competences of developers vs. developers’ own reliance on their capability.</td>
<td>Changes in capability of understanding of UI requirements. UI requirements became more important.</td>
</tr>
<tr>
<td>System development resource allocation</td>
<td>Lack of project resources, especially UI expertise</td>
<td>Pre-planned priorities, schedule and budget vs. the necessity of the situation.</td>
<td>Changes in project resources – UI expertise was acquired.</td>
</tr>
</tbody>
</table>
Incongruence happened when understanding, attitudes, or expectations differed among the stakeholders, causing conflicts and misunderstanding. After negotiating the understanding of attitudes and expectations were the same.

- **Shifting** that took place when the understanding of a frame changed. After a frame shift, the parties involved got an understanding of a frame that was more aligned with and suitable for the current situation than before the shift.

In the beginning of our Electronic Commerce platform project differences in attitudes and expectations redirected the participants’ attention away from the relevant information and filtered their understanding of the project requirements. In the later phases, the ability of the project to resolve these differences in the negotiations between participants redirected their focus towards the relevant information and led to shifting in their attitudes and expectations helping the project to make sense of the information in a new way. This sense making in the system context was an iterative ad hoc-process that happened through social interaction, communication and negotiation between the parties.

The results of our study contribute to existing requirement research in an important way. This study makes a substantive contribution to the understanding of the requirement elicitation process and systems development in general. While the current approaches still largely assume that projects proceed with distinct phases in a more or less waterfall fashion and the system is developed from an understanding of the idea into a final system, which satisfies the originally stated requirements. Instead, our study implies that the requirement shaping is an ad-hoc and iterative process in which filtering, negotiating and shifting of different attitudes and expectations about systems development change the participants’ interpretation and understanding of requirements during the project.

The full description of the case and the results can be found from [Ovaska, 2005].

**References**


Session 3: Reuse and Product Lines

An Experience on Modelling Teleoperated Systems for Ship Coating Removal through Features and Generic Use Cases
Joaquín Nicolás, Joaquín Lasheras, Ambrosio Toval, Software Engineering Research Group, Universidad de Murcia, Spain
Francisco J. Ortiz, Bárbara Alvarez, Systems and Electronic Engineering Division, Universidad Politécnica de Cartagena, Spain

Project-oriented Reuse Approaches: Copy-and-Paste or Software Product Line Engineering?
Ramin Tavakoli Kolagari, Fraunhofer IESE, Kaiserslautern, Germany
Nicolas Fernando Porta, University of Ulm, Germany
A Collaborative Learning Experience in Modelling the Requirements of Teleoperated Systems for Ship Hull Maintenance

Joaquín Nicolás, Joaquín Lasheras, Ambrosio Toval
Universidad de Murcia. Campus de Espinardo. 30071 Murcia (Spain)
{jnr, jolave, atoval}@um.es

Francisco J. Ortiz, Bárbara Álvarez
30202 Cartagena (Spain)
{francisco.ortiz, balvarez}@upct.es

Abstract: This paper presents a joint experience in modelling the requirements of the product line of teleoperated systems for ship hull maintenance, which are basically robotic systems used for ship maintenance operations, such as cleaning or painting the ship hull. It is proposed to specify the product line requirements through a feature model, a conceptual model, and a use case model, which together allow domain understanding, derivation of reusable product line requirements, and efficient decision-making in the specification of new systems developed in the product line. Action Research, a qualitative research method in software engineering, has been applied to define the collaborative research process.

Key Words: Domain Analysis, Product Lines, Requirements Reuse, Teleoperated Systems, Feature Modelling
Categories: D.2.1, D.2.13

1 Introduction

This paper presents an experience on modelling the requirements of the product line of the teleoperated systems for ship hull maintenance (TOS hereafter) through a collaborative learning process carried out by two Spanish research groups of different but complementary fields: the Systems and Electronic Engineering Division (SEED) of the Polytechnic University of Cartagena and the Software Engineering Research Group (SERG) of the University of Murcia.

In recent years, SEED has gained considerable experience in developing software reference architectures in the TOS domain [Fernández et al., 2004]. To date, SEED has paid less attention to requirements reuse in this product line than to architectural components reuse.

1 Partially financed by the CICYT (Science and Technology Joint Committee), Spanish Ministry of Science and Technology, project DYNAMICA (DYNamic and Aspect-OrienteD Modeling for Integrated Component-based Architectures, TIC2003-07804-C05).
Independently, SERG has defined a reuse-based requirements engineering method and has developed catalogues of reusable, textual requirements in security [Toval et al., 2002a] and personal data protection [Toval et al., 2002b] domains.

An increment in the abstraction level of knowledge management could improve future reuse (see for example [Cybulsky and Reed, 2000]), so that collaboration between SEED and SERG appears in a natural way. The goal is to obtain a TOS domain model to accelerate domain knowledge acquisition and to facilitate the specification of new systems in the product line and the reuse of architectural components.

Experience of SERG with the personal data protection and security requirements catalogues is insufficient. Both domains can be considered well structured, if the sources of requirements considered are, respectively, Spanish and European legal documents related to personal data protection, and a well documented method for risk analysis and management in Spanish public administration. This is why it was relatively straightforward (in an ad hoc manner) to structure the knowledge of these domains into two textual requirements catalogues. However, what do we do when the problem domain is complex and knowledge is not previously structured, as occurs in the TOS? In these cases an ad hoc approach is not enough to obtain a quality requirements catalogue. Hence, the need for a better representation and organisation of the knowledge in wide, complex and slightly structured domains appears.

This work presents a modelling of the TOS product line requirements. Several existing domain analysis techniques have been adapted in this approach, which basically consists of: (1) a feature model, as a high level interface that favours the reuse of the product line requirements; (2) a conceptual model, which leads to greater domain understanding; and (3) a generic use cases model, enabling descriptions of the scenarios related to the execution of certain functional features in the feature model. The use of features and generic use cases makes it possible to capture variability in the product line. A qualitative research method in software engineering, Action Research, has been applied to define the research process formally.

This paper is structured in the following way: Section 2 shows how the collaboration between SEED and SERG has been designed by means of Action Research. Section 3 briefly outlines the TOS domain. Section 4 discusses the general approach to the case study, and Sections 5, 6 and 7 show feature, conceptual and use case models. Finally, conclusions and future works are given in Section 8.

2 Research framework

This contribution reports the results of an experiment designed to obtain a TOS domain model. Action Research [Baskerville, 1999], one of the most well known qualitative research methods in software engineering, was used to design the experiment. The application of Action Research produces a cyclic process in which all the parts involved in the research participate, examining the existing situation with the intention of changing and improving it. Action Research is a valid approach for studying the effects in human organizations of specific changes in systems development and maintenance methods.

In line with the Action Research terminology, the following roles have been considered in this experiment:
• The researcher is the SERG.
• The researched is the TOS product line, from a domain analysis point of view.
• The critical reference group (CRG) is SEED, i.e. the researched for (in the sense of having the problem the research is to solve). According to Action Research, the CRG has to participate in the research process too, although it can be involved less actively than the researcher.
• The stakeholders are all those organizations that might benefit from the results of the research: in this case, the CRG and, in general, companies that perform ship hull maintenance tasks.

The activities performed in each cycle of Action Research can be summarised as follows:

• To begin, a planning is made, in which the questions to guide the research are identified and the actions to solve those questions are specified. In this case study, first the interest of performing an analysis of the TOS domain was justified and then the state of the art in the domain analysis field was studied in order to propose an approach to tackle the problem.
• An action task follows, in which the researcher takes part in the real situation through a careful, deliberate, and controlled variation of the practice. In this case study, the researcher, together with the CRG, built an initial model of the TOS.
• Then, an observation or evaluation is made, in which information on the effects of the action is collected.
• The cycle ends with a reflection, in which the results are shared and analysed by all the interested parts, and new important questions can be raised which can be tackled in a new cycle of Action Research.

3 Teleoperated Systems for Ship hull Maintenance

The global objective of the European project EFTCoR (Enviromental Friendly and Cost-Effective Technology for Coating Removal, Fifth Framework Programme) [Fernández et al., 2004], carried out by SEED, is the development of a new technology for ship coating removal. The project tries to solve a critical problem in the European naval industry: the preparation of the hull surface for painting in an environmentally friendly way.

TOS consist of the following subsystems: the Robotic Device Control Unit (RDCU), which controls the devices used for coating removal; the monitoring subsystem, which carries out the tasks of management of information related to ship maintenance; the vision subsystem, which performs the visual analysis of the hull; and the recycling subsystem, which is in charge of removing wastes from the work area and recycling it.
4 A Framework for Domain Analysis

In planning the collaboration, the most important decision was to base modelling on features (cf. FODA [Kang et al., 1990], FORM [Kang et al., 1998] or PLA [Kang et al., 2001]), which intuitively specify the vision of the product line that the stakeholders have. With features the domain can be explored quickly in order to know the main issues and the common and variable points. In addition to the feature model, the following have also been used in the modelling of the TOS:

- A conceptual model, showing the concepts of the domain and their relationships.
- A use case model, describing in detail the interactions between the external actors and the system that can occur during the execution of some functional features of the feature model.

In contrast to previous experiences [Toval et al., 2002a, Toval et al., 2002b], the structure of the TOS catalogue of requirements does not consist of a list of (mainly textual) requirements which are structured in a document hierarchy. Now the requirements are directly structured in the catalogue starting from the feature model: features are related to use cases and non-functional requirements through traceability links. By using features instead of natural language requirements, a more agile reasoning –without considering the location of the requirements in the requirements documents– is sought. Thus, during the specification of a product belonging to the product line, customers can “navigate” in the “problem space” through the “decision space” that features make up, selecting one feature or another, in an easier way than scanning a requirements list in order to select one requirement or another. Moreover, during the initial requirements specification, it is more convenient to perform a feature analysis than to write quality textual requirements, which have to be unambiguous, complete, consistent, and verifiable, as the IEEE 830 standard establishes.

5 Feature Model

The functional and non-functional capacities and the technology constraints that can appear in the products of the product line are modelled in the feature model. This feature model plays an essential role in the domain analysis of the TOS: clients usually specify their necessities intuitively in terms of the “features that the new system has to have or provide”, understanding these as abstractions of the capacities of the system that must be implemented, proven, given and maintained [Kang et al., 1998]. Both clients and developers can intuitively understand the feature model.

In this case study, the feature model defined in FORM has been adopted, and extended as follows:

- With the goal of simplifying the feature model, the four layers of features of FORM (whose complexity is criticised in [Trigaux and Heymans, 2003]) have been reduced to only two: capability and implementation. The latter gathers the original layers of “operating environment”, “domain
technology”, and “implementation technique”, which are very close and sometimes seem to overlap, giving rise to confusion.

- In [van der Maßen and Lichter, 2002] the requirements are described which any notation must satisfy to model variability in a product line. Subsequently, [Trigaux and Heymans, 2003] add another requirement: the graphical representation of variability, and in particular, of variants, variation points and cardinalities of variation points. With the goal of satisfying all the requirements, the original notation of FORM has been extended with the graphical representation of variation points (as per [Griss et al., 1998, van Gurp et al., 2001]) and cardinalities (following [Riebisch et al., 2004]).

A part of the feature diagram is shown in Figure 1, where the services offered by the family product are reflected. The relationships between the layers of capability and technology are specified through the implemented-by traces, such as the one that shows the types of technology used to make cleaning (Sand-Blasting and Hydro-Blasting). This implemented-by trace also represents a variation point within the product line, identified as Coating Removal Technology. The cardinalities of the variation points can be also observed in the diagram: in this case study, only alternative features of which at least one must be chosen have been identified, so each has cardinality 1.

Figure 1. A part of the feature model.
Each feature is described textually by means of a template (Table 1). Then it is presented the textual description of the Spot feature (see Figure 1), which belongs to the capability layer and is mandatory.

<table>
<thead>
<tr>
<th>FEATURE</th>
<th>Spot (capability, mandatory)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description:</td>
<td>cleaning performed only in isolated points of the surface of the ship hull.</td>
</tr>
<tr>
<td>Rationale:</td>
<td>It is often necessary when the cleaning of the complete hull (full-cleaning) is not of interest or it can not be performed, because it is more expensive or there are parts of difficult access.</td>
</tr>
<tr>
<td>Composition rules:</td>
<td>Does not have (i.e., does not break down into other ones)</td>
</tr>
<tr>
<td>Binding time:</td>
<td>at definition time</td>
</tr>
<tr>
<td>Trade-offs:</td>
<td>Requires Primary AND Secondary. It is necessary to have secondary and primary positioning systems, since the primary positioning system is not accurate enough by itself.</td>
</tr>
<tr>
<td>Trace to requirements:</td>
<td>Spot Cleaning use case</td>
</tr>
</tbody>
</table>

Table 1. “Spot” feature.

6 Conceptual Model

The complex TOS domain was alien to SERG, so a conceptual model was first used to begin modelling. A classic vision in software engineering (described e.g. by [Larman, 2005]) was used, according to which the conceptual model describes the vocabulary of the domain, i.e. the concepts of the problem space and the relationships between them. Requirements (features and use case) must have as "direct objects" the concepts of this “information model”.

A part of the conceptual model for the RDCU within the EFTCoR system is presented in Figure 2, including interactions between the RDCU and the other subsystems.

Figure 2. A part of the RDCU conceptual model.
7 Use Case Model

The execution of some functional features of the feature model can be naturally specified as a use case: when a system’s actor requires the execution of a feature to obtain an observable value, causing a set of interactions with the system that can be captured within a use case (e.g. Cleaning feature of Figure 1). An N:M relationship between features and use cases can be established: for example, the Cleaning feature can be related to several use cases in Figure 3 (Full Cleaning and Spot Cleaning), and the Spot Cleaning use case can be related to several features (Cleaning, Spot, Automatic, Semi-Automatic and Teleoperated).

The use case model is structured through features, expressing with these the variation points of the product line, and thus avoiding as much as possible an overload of the use case model with the complexity associated to the product line variability. However, there is variability intrinsic to use cases in a product line—that associated to the possible variations in the steps of the different scenarios captured in the description template of the use case—.

After reviewing different approaches to capture the variability of a product line in the use cases [Gomaa and Shin, 2002, John and Muthig, 2002, Halmans and Pohl, 2003, van der Maßen and Lichter, 2002, Eriksson et al., 2004], that of [Eriksson et al., 2004] has been adopted, since it is in line with the focus previously described, and because it proposes the use of two interesting techniques:

- **Change cases**, which capture the possible impact in the use cases of the adoption of future, anticipated extensions of the system that are still unavailable. Such possible changes are grouped in special use cases denominated change cases [Ecklund et al., 1996]. What use cases can be affected by each change case is indicated by means of the impact link trace relationships. For example, Figure 3 shows a change case, Hydro-Blasting, implying a change in the cleaning technique used (Sand-Blasting until now).

- **Modelling of the variability in the description of use cases**, using (1) (local and global) parameters in the description of the use case, and (2) an extended version of the textual description of the black box flow of events of the RUP SE (Rational Unified for Process Engineering Systems) [Cantor, 2003], in which the steps of the scenario where variation can appear are expressed in a special notation (see Table 2). Firstly, the description of the flows of events is made considering the system as a black box (Table 2). Later, the description is reviewed and each black box step is replaced by a sequence of white box steps, showing the interactions of the different subsystems to support each black box step.

In order to make the use case diagram more legible, and following [Gomaa, 2005], optional and alternative use cases are labelled with the «optional» and «alternative» stereotypes. The details of the variability are specified in the feature model.

It can be observed that several steps are alternatives in the textual description of the use case in Table 2 (they use the same number), evincing that the action can be made in a teleoperated or automatic form: a step 2 would be traced to the...
Teleoperated feature and the other step 2 to Automatic (analogously to step 3). Notice that trace relationships can be established between complete use cases—or steps in use cases—and features. Moreover, optional steps such as (4) are also captured. In addition, a global variable ($MAX\_TIME\_BUTTON$) together with two local ones (@MAX\_TIME\_TOOL, @MAX\_TIME\_SAFE\_STOP) have been used to express the maximum response time to certain actions within the use case.

![Use case diagram](image)

**Figure 3. A part of the use case diagram.**

<table>
<thead>
<tr>
<th>Step</th>
<th>Actor Action</th>
<th>Black box</th>
<th>Black Box Budget Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>This use case starts when the operator pushes the cleaning button.</td>
<td>The system is started to carry out the cleaning operation</td>
<td>Max. response time is $MAX_TIME_BUTTON$</td>
</tr>
<tr>
<td>2</td>
<td>The operator sees images of the ship hull surface and executes commands to place the tool in the cleaning area.</td>
<td>The tool is placed in the cleaning area.</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>The vision system executes commands of the positioning systems to place the tool in the cleaning area.</td>
<td>The tool is placed in the cleaning area.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>The operator pushes the button to activate the tool.</td>
<td>The cleaning tool is activated.</td>
<td>Max. response time is @MAX_TIME_TOOL</td>
</tr>
<tr>
<td>3</td>
<td>The system activates the cleaning tool.</td>
<td>The cleaning tool is activated.</td>
<td>Max. response time is @MAX_TIME_TOOL</td>
</tr>
<tr>
<td>(4)</td>
<td>The operator pushes the button of emergency stop.</td>
<td>The system stops securely.</td>
<td>Max. response time is @MAX_TIME_SAFE_STOP</td>
</tr>
<tr>
<td>5</td>
<td>The operator pushes the cancellation button.</td>
<td>The cleaning stops at that point.</td>
<td></td>
</tr>
</tbody>
</table>

**Table 2. Black-box description of the “Spot Cleaning” use case.**

8 Conclusions and Further Work

An experience of cooperative research work has been presented where a reusable requirements catalogue of the TOS product family has been obtained. The CRG affirms that the documentation of requirements generated for the product line has an...
added value through its spreading to clients and developers, given that it is not usually
developed in this type of systems.

According to the experience of the researcher in the field of personal data
protection and security, one risk in the elaboration of a reusable requirements
catalogue for a wide domain is that a catalogue can be obtained, formed by a long list
of textual requirements, which may be correct and very precise but as the same time
difficult to handle. In order to avoid this problem, a feature model has been
incorporated into the catalogue, which can be used as the starting point to structure it.
This model acts as an interface which facilitates requirements selection, permitting an
intuitive navigation through the space of the problem through the feature model. The
natural integration between features and use cases has been seen.

The graphical representation of the variation points in the feature model
–extending the FODA and FORM notations– has been useful to stress the decisions
that have to be taken in the instantiation of the family. Nevertheless, the use of a
notation for the cardinalities has not been crucial: in this respect the FODA and
FORM notations would have been enough in this case study.

The CRG considers that the approach would be more useful with a tool to
navigate easily through the feature model and to manage its graphical complexity,
which quickly makes it difficult to handle.

A possible line of further work consists of assessing the adoption of a process
model to define, develop and maintain the requirements of a product line, like for
example that proposed by PuLSE (Product Line Software Engineering) [PuLSE].

References

[Baskerville, 1999] Baskerville, R. L. Investigating Information Systems with Action Research,
Communications of the Association for Information Systems, 2. 1999


[Cybulsky and Reed, 2000] Cybulsky, J. and Reed, K. Requirements Classification and Reuse:
Crossing Domains Boundaries, 6th International Conference on Software Reuse (ICSR’2000),
Vienna 2000

[Ecklund et al., 1996] Ecklund, E., Delcambre, L. and Freiling, M. Change cases: use cases
that identify future requirements, ACM SIGPLAN Notices, 31, 342 - 358. 1996

[Eriksson et al., 2004] Eriksson, M., Börstler, J. and Borg, K. Marrying Features and Use
Cases for Product Line Requirements Modeling of Embedded Systems, Fourth Conference on
Software Engineering Research and Practice in Sweden (SERPS’04), 2004

Co-operative Robots for Hull Blasting in European Shiprepair Industry, Robotics and
Automation Magazine (RAM), 2004

Pattern-Based Software Architectures, Addison-Wesley. 2005


[van der Ma llen and Lichter, 2002] van der Ma llen, T. and Lichter, H. Modeling Variability by UML Use Case Diagrams, International Workshop on Requirements Engineering for Product Lines (REPL’02), 2002

Project-oriented Reuse Approaches: Copy-and-Paste or Software Product Line Engineering?

Ramin Tavakoli Kolagari
(Fraunhofer Institute Experimental Software Engineering, Germany
fraunhofer.tavakoli-kolagari@daimlerchrysler.com)

Nicolas Fernando Porta
(University of Ulm, Germany
nicolas_porta@web.de)

Abstract: Software reuse plays a crucial role in software development especially in the context of embedded systems, because functions and goals of established systems often do not change in contrast to technology, system architecture, and market demands. Although these changes are mostly in the solution domain, they do have an impact on the requirements, since requirements are not only placed in the problem domain as stated by most requirements engineering literature. Hence there is a need to develop reuse approaches applicable to different software development domains, layers, and artifacts. Meanwhile numerous different reuse approaches with different goals were developed that proved successful in practice. However, these approaches differ in their presentation and applicability, and it is not transparent which reuse approach is applicable to a specific project. In software product line research – a prominent and systematic representative of reuse concepts – formulas are presented estimating the benefit of a product line approach. But these formulas neither indicate which reuse approach should be applied in a specific project, because there is a product line approach assumed, nor take the relevant project characteristics into account. This situation is not satisfying. In the automotive industry, in which the research questions of this paper were analyzed, it is in fact possible to indicate which reuse approach is cost-benefit balanced and which project characteristics must be taken into account. This paper therefore proposes identifying organization and project specific reuse approaches by capturing relevant experience within an organization rather than developing complex formulas that are inherently deficient.

Keywords: cost/benefit, requirements engineering, (software) reuse, software product line engineering, software project management

Categories: D.2.1, D.2.9, D.2.13, K.6.3, K.6.4

1 Introduction

An enormous amount of reuse approaches exist in current research literature (e.g. [Atkinson, 00], [Bayer, 99], [Clements, 02], [Heumesser, 03], [Simos, 96], and [Weiss, 99]) and each approach claiming to be successful in the practical setting in which it was applied. For projects facing the challenge of coping with shorter innovation cycles and enhanced quality demands of the customer, it is not easy to identify a reuse approach applicable to the specific conditions of the project, or to what extent a reuse approach would
solve the current problems at all. Since the trigger when to use a reuse approach is quite generic – consisting mostly of the determination that the amount of work per amount of time becomes huge and that a non-negligible part of the work recurs – the project needs different information about the reuse approaches to select it properly. One possible selection criteria may be a cost/benefit analysis of different approaches in the context of the project. But there is no cost/benefit analysis ready-made, reliable and broadly accepted such that a project could use it as-is and generate a reliable result on which to base its choice for a reuse approach. The proposed cost/benefit analyses (e.g. [Böckle, 04], [Bollinger, 90], [Clements, 03], [Cruickshank, 93], [Gaffney 92], [Green, 85], [Pfleeger, 94], [Mili, 01], [Wöbey, 96], and [Zubrow, 03]) concentrate on understandable formulas based on a simplified underlying reuse scenario rather than identifying the crucial drivers for cost and benefit in a special project setting. In the latter case, the formulas naturally may become complex and would not be useful anymore, so the authors did without. Nevertheless there exist concepts taking special settings into account and hence presenting real support for the identification of reusable products (e.g. [Schmid, 03a], [Schmid, 00]), but a special reuse approach assumed. In the case that the reuse approach itself is in question, there exist concepts of the software productivity consortium or product line assessments answering the question whether a specific reuse approach is applicable in a project or not [DACS, 06]. From the view of a project, it is not relevant whether a specific approach is not applicable, but rather which approach fits best to the goals and challenges of a project with respect to costs and benefits. This question cannot be answered fundamentally for all possible projects, because not all relevant differences between projects can be captured and analyzed. Hence it is important to follow an experience-based approach.

In the Software Product Line Conference 2005 there was a panel discussion on different cost/benefit analyses. The representatives were asked to calculate the benefit and cost of a given scenario based on their formulas. Although each approach seemed reliable and convincing, the results were totally different from one another. In the subsequent discussion the panelists agreed that it was not important what the result says but which aspects of the scenario the formula consumed, hence which aspects must be viewed and analyzed properly if a shift to a reuse approach is in sight.

We also agree with this view on cost/benefit analyses – especially in the context of reuse-oriented approaches – and we think that the classical way of describing the internal relationships in complex formulas directs the attention to wrong aspects. Instead we tried to find the important aspects we see in requirements reuse. We want to encourage organizations to learn in their projects how the identified reuse aspects behave in the organization-specific setting. In this paper we want to show the basic aspects of which reuse approaches commonly exist and to what extent they involve one another. Because of the different applications of reuse approaches in practice, we are not arguing for a common cost/benefit analysis but rather propose an organization-specific learning during the project.

The paper supports organization-specific learning in this context by:

- identifying relevant project characteristics,
- identifying the main goals for specific reuse approaches,
- describing the impact between these different goals,
• presenting exemplary net diagrams for documenting the current status as well as the desired status of a project with respect to their reuse approach, and

• presenting exemplary models (argumentation-based rationale models) for documenting organization-specific behavior (the model was developed with the help of Ian Alexander and the scripts for the usage of the model in Telelogic DOORS may be downloaded at http://www.scenarioplus.org.uk/download_diagrams.html).

In the remainder of the introduction we will explain software product lines, because they are an important notion and are the most systematical reuse approach, whereas the copy-and-paste method on the other end of the scale is the ad-hoc reuse approach.

Software product lines

A software product line is “a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way” [Clements, 02]. The basic idea of product lines is therefore the creation of core assets for every artifact arising in the system development for a specific scope of products that are commonly managed. This is typically done in a two-layered process: core asset development (or domain engineering) and product development (or application engineering), both of which are related by management tasks. The concentration on the core assets and the consistency of the approach over all artifacts are the main benefits of software product lines. Hence, the management tasks can concentrate on the main aspects of the system (the core assets) and are not supercharged with caring for individual features. Also, product development does not need to explore a pool of possible components (as is the case in component-oriented reuse approaches [Atkinson, 00]), instead only needing to enhance the core assets as required for the future product development. Originally developed from software architecture research [Bass, 03], [Bosch, 00], [Kang, 90], software product lines find their main benefit in this area, since the artifact of software architectures is the link between requirements and the code. Changing software architecture is a main driver for costs in software development. Nevertheless, the introduction of product lines is only possible with increased costs, particularly because of organizational change and increased communication demands [Birk, 03], [Bosch, 02].

The following section presents typical cost-benefit-models – asking what the relevant ideas of the reuse approach are and how the cost/benefit analysis is performed. In section 3 there are some requirements reuse scenarios sketched, documenting the possible different shapes of reuse in practice. Section 4 presents project goals and characteristics for reuse approaches by identifying relevant project characteristics, main goals for specific reuse approaches, and the impact between these different goals. Section 5 describes the generic concepts reuse approaches consist of. Section 6 presents an example of what an experience-based project specific selection of a reuse approach might look like. The last section of this paper concludes in a discussion of the presented approach and explains further needed research.
2 Typical Cost-Benefit-Models

The interface between different development steps plays an important role, especially if a systematic reuse approach – like the one of software product lines – is followed. Software product lines demand a coherent reuse approach encompassing all artifacts produced during system development. Other research proposes to realize the product line concept solely on the level of requirements [Hardt, 02], but the benefit is not analyzed. Therefore the starting point of our research were different cost-benefit-models like [Böckle, 04], [Bollinger, 90], [Clements, 03], [Cruickshank, 93], [Gaffney 92], [Green, 85], [Pfleeger, 94], [Mili, 01], [Withey, 96], and [Zubrow, 03]. Our vision was originally to combine the relevant aspects of these different analyses in order to develop a reliable tool which should be able to recommend a specific reuse systematic which would meet the necessities of a concrete project. A generic statement evaluating the different aspects of a project leaving the different reuse systematic variable was supposed to indicate about which reuse concept fits best. This generic statement should be complex enough to allow relevant conclusions but also be applicable in practice. In the process of generating such a statement, we encountered a set of problems which we believe to be typical for cost-benefit-models in this context. In the following, two different cost-benefit-models are shortly outlined in order to demonstrate why we have chosen to abandon methods based on formulas and plead for an organization-specific learning as drafted in the introduction.

In [Cruickshank, 93], Cruickshank and Gaffney developed a model based on the division of generic and application effort. Reuse is done by multiple use of large granules of software functions which are organized in a library. The goal of the model is to show at which number of systems – given certain conditions – the break-even point is reached and how big the benefit – calculated as the difference of effort compared with conventional development – accounts for a distinct number of systems. The subsequent formula defines for which number of systems \( N_0 \) of a software family the break even point is reached [Cruickshank, 93]:

\[
N_0 = \frac{C_{DE}}{(C_{VN} - C_{VR})E}
\]

\( C_{DE} \) designates the unit costs of domain engineering, \( C_{VN} \) the unit costs of new code developed for an application system and \( C_{VR} \) the costs of reusing code from the library. \( E \) stands for the library efficiency which is defined as the proportion of the actual reused code compared to the potentially reusable code. As seductive formulas like this example may appear prima facie, they restrict more that they explain. Many simplifications are needed to model the complex reality of software projects into a small and therefore handy equation. Instead of gaining proper information many relevant aspects are hidden due to simplification. For example the model appeals to the different phases of the software development process only naively by defining their proportion of the whole development cycle. Furthermore project characteristics are only considered by the costs of software units and the degree of reuse. In sum, the model does not meet the requirements prevailing in the complex practices of software engineering, with many different scenarios and contexts which often differ from one project to another.

An interesting and modern approach is PuLSE-Eco [Schmid, 03b] which concentrates on the scope of a product line. The three main phases of PuLSE-Eco are
the product line mapping, a domain potential analysis and the scoping of the reuse infrastructure. The benefit given by cost reduction for a single functionality $f$ can be calculated with the following utility function:

$$E(f) = \sum_p \text{req}(f, p) \cdot \text{eff}(f, p) - (\text{eff}(f, p_{\text{gen}})) + \sum_p \text{req}(f, p) \cdot \text{eff}_{\text{reuse}}(f, p)$$

The benefit is given as the difference of effort for each functionality $f$ concerning conventional development (which does not include reuse) and a product line approach. Not all functionalities are used in the same product, therefore $\text{req}(f, p)$ determines if a specific functionality is required in a particular product $p$ ($1 = \text{yes}; 0 = \text{no}$). The effort needed to implement a functionality $f$ in a product $p$ is expressed by $\text{eff}(f, p)$; the effort to make a functionality $f$ generic is given by $\text{eff}(f, p_{\text{gen}})$; the effort which is needed to implement a generic function accounts with $\text{eff}_{\text{reuse}}(f, p)$. This formula meets our intuition about how benefit gaining could be measured for a set of functionalities. Still validation of utility functions seems difficult. The main reason is that reuse only makes sense if applied to large projects. To validate such functions, it would be necessary to conduct a controlled experiment consisting of two equal large software projects performed under comparable circumstances, which means extremely high costs.

Being aware of these difficulties and of the panel discussion in the Software Product Line Conference 2005 on cost-benefit analyses, we conclude that the aspects of the scenarios behind the formulas are the important point and not the formulas themselves. We therefore concentrated in identifying these aspects and formulate methods for organization-specific learning during the project as described in the following sections.

### 3 Scenarios for requirements reuse

Requirements reuse is dependent on project history, experience and technological knowledge of the engineers, involvement of different stakeholders and complexity of the system. Basically the same reuse concepts may have different realizations with respect to the tools applied, the development process it is embedded in, and the produced artifacts. Requirements reuse has an individual, project-specific, and organization-wide dimension with an intense interrelation between one another.

For an automotive original equipment manufacturer (OEM), the development of electronic control units (ECUs) often takes place solely at the requirements level. The resulting specifications are then handed over to software and hardware suppliers who implement the system. In the case of established and complex automotive systems such as the instrument cluster, the development of the OEM’s specifications is supported by sophisticated reuse approaches [Heumesser, 03], thereby enabling the quality of the specifications to be improved while development time is cut, in spite of an increase of functionality.

Systems with a small amount of functionality but high integrity time and safety constraints like the airbag system can be described only by features ([Hein, 00], [Kang, 90]) on the requirements level and hence the reuse approach may only include copy-
and-paste, while system and software design comprises complex diagrams and specifications and hence a more complex reuse approach.

As stated in section 2 it is crucial to keep the interfaces between different development steps in view. For the automotive domain, this means that costly reuse on the level of requirements should be evident on subsequent development steps, because most development costs can be saved on the levels that are attended by suppliers. Another important aspect is the usage of a reuse approach for innovative systems. Requirements for innovative systems should be formulated in a reusable way, because innovative systems are meant to be reused and advanced in further development cycles.

Current research literature constantly neglects these aspects that are obvious in practice. For a project it is now more important to learn specific approaches or workarounds that proved to be successful in (a special and partly situational) practice rather than learn generic concepts as presented in literature. This motivation holds for the identification of patterns as well as the identification of practical cost-benefit balanced organization-specific approaches.

In a Master’s Thesis [Deng, 04] at DaimlerChrysler, we tried to analyze if it is basically possible to identify special organizational reuse scenarios and to transfer one approach to another project. In Figure 2 the essential result is depicted: Four dimensions of the projects are presented and based on these dimension one can make a first proposition, which reuse approach should be followed for a new DaimlerChrysler development project. We are currently transferring successful reuse approach from the truck business unit to the bus business unit, because comparable development depth, innovativeness and size of the projects. The figure shows an

![Figure 2 Size (size of the cycles), systematic grade of reuse approach, and development depth of DaimlerChrysler projects and innovativeness of the system (from [Deng, 04]).](image-url)
interesting but obviously not astonishing result: Projects developing very innovative or very established systems are following more systematic reuse approaches than projects developing systems that are neither very innovative nor very established. This can be explained because the reuse approach for an innovative system differs from the reuse approach for an established system and systems that are no real innovations anymore are enroute to becoming established systems and no systematic reuse approaches exist for these kinds of systems.

This section provides the motivation for the following sections in which we try to identify the generic concepts of reuse approaches: how they can be related to project specific goals via an experience-based argumentation model. In this section we stated that an enormous amount of different reuse approaches exist in practice and that these differences can essentially be connected to project characteristics.

But since not all important project characteristics can be identified, a generic model for the derivation of a project specific reuse approaches via project characterization and goals is not possible. We therefore propose to capture organization-specific experience linked to obvious project characteristics.

4 Project characteristics and goals for reuse approaches

The first step to recommend proper reuse concepts is to analyze the project characteristics. As stated before, not all relevant project characteristics can be analyzed, still some clearly important points can be identified giving an indication which generic reuse concepts may be appropriate:

1. Software projects can be distinguished by the depth of development [Deng, 04]. In the case of ECUs, the development will probably consist of the requirement engineering phase. The producer of a text processing software, on the other hand, may control the whole development process. As a consequence, in the first case the interfaces between the different developments steps are more important. Additional information and organization may be needed.

2. The innovation of the project is decisive [Deng, 04]. Especially market needs can play a role. E.g. if marketing suggests a very short development time for the first product of a family there won’t be much time for upfront domain engineering. Examples can be easily found in the mobile phone market, where the pressure to deliver new features in very short time periods is immense. In contrast, the development of me-too-products concentrates merely on overall cost reductions for a large number of similar products and as a result large upfront domain engineering might be the best choice.

3. The similarity of products plays an important role. If the products do not share a big number of features the effort of building a library seems too big compared to the possible benefit. For products which share only few features, lightweight reuse concepts seem more appropriate.

4. The number of planned products is crucial. As stated in [Cruickshank, 93], the costs per product/system drops if reuse concepts are used.
5. The planning horizon can be important, as strategic decisions regarding future developments of depending products have an impact on the reuse concepts of current projects.

In addition to the characteristics of a project, the goals of reuse concepts are crucial. High level goals for all kind of reuse activities are shorter development cycles and an improved quality. There is a set of subgoals such as the handling of system scoping where the features of the product are identified. Another subgoal is a good understanding of the system which is essential since it delivers the information needed to control the requirements engineering process. Further goals consist of the reusable formulation of requirements and an enhanced usability of the system. Figure 3 shows all relevant goals and subgoals on three hierarchical levels and their relations. Arrows indicate a supporting relationship to another goal.

![Figure 3: Goals for reuse approaches.](image)

After identifying the project’s main characteristics and goals, the proper combination of reuse concepts can be created out of several generic concepts as described in the following section.

5 Generic requirements reuse concepts

The constantly growing set of presented reuse approaches contains a comparably small set of generic concepts. Knowing these concepts supports the identification of project-specific needs for a reuse approach. Each concept can act in different reuse approaches in either of the following two ways: 1) It can itself fulfill a specific goal that has to be met in the course of reuse or 2) it can act as a support for other concepts that fulfill special goals connected with a reuse approach. The essential goals of reuse
are shorter development cycles and improved quality by intensive review and revision as stated in the previous section.

In the following, we list the identified generic reuse concepts and document, which goal(s) it satisfies immediately and which goal(s) or other concept(s) are supported indirectly; the two essential goals are not named. Generic concepts can be applied together and sometimes generic concepts are full reuse approaches in the same way; at the end of this section we show as an example how software product line systematics on the level of requirements can be constructed out of the presented concepts. The list is not complete and we encourage the identification of further (and possibly organization-specific) concepts.

- **Abstraction layers**: Requirements are organized in abstraction layers describing goals, user requirements, and system requirements
  Immediate goal(s): abstraction, good understanding of the system
  Supported goal(s) or concepts: antibody, clustering of requirements

- **Antibody**: Requirements are used to recognize a problem and to identify an appropriate design element.
  Immediate goal(s): abstraction
  Supported goal(s) or concepts: clustering of requirements

- **Classification**: Requirements are indexed with suitable attributes.
  Immediate goal(s): none
  Supported goal(s) or concepts: good understanding of the system, clustering of requirements

- **Clustering of requirements**: Requirements are clustered with respect to the demands of other artifacts or process steps; hence requirements can be clustered by design elements, features, abstract requirements, test cases, safety integrity levels, production series, etc.
  Immediate goal(s): easy identification of requirements that belong together, abstraction, description of relationships
  Supported goal(s) or concepts: easy identification of impact of changes, good understanding of the system, dependency network

- **Decomposition layers**: Requirements are organized in decomposition layers describing system requirements, subsystem requirements, component requirements, etc.
  Immediate goal(s): abstraction, good understanding of the system
  Supported goal(s) or concepts: clustering of requirements

- **Dependency network**: Requirements are described with the dependencies between them. Dependencies can describe relations between requirements such as “derived from”, “excludes” (strong or weak), “includes” (strong or weak), “influences”, “support”, “constrains”, “communicates” etc.
  Immediate goal(s): description of relationships
  Supported goal(s) or concepts: easy identification of impact of changes, good understanding of the system, explicit description of variability, variability coupling, clustering of requirements

- **Detection and elimination of redundancy**: Requirements for a scope of similar systems are surveyed with respect to redundancy and possibly reformulated in the case that redundancy is stated.
Immediate goal(s): reusable formulation of requirements
Supported goal(s) or concepts: handling of system scoping

- **Known core (function):** A block of known reusable subfunctions is identified for a set of similar functions.
  Immediate goal(s): reusable formulation of requirements
  Supported goal(s) or concepts: good understanding of the system, enhanced usability of the system

- **Known core (system):** A block of known reusable requirements is identified for a scope of similar systems.
  Immediate goal(s): handling of system scoping
  Supported goal(s) or concepts: handling of evolution of requirements

- **Library:** Requirements for a scope of similar systems are described in a list. Basically each requirement is applicable to all systems of the scope, but technology or marketing decisions conclude in different specifications populated by the requirements of the library for the different systems.
  Immediate goal(s): handling of system scoping
  Supported goal(s) or concepts: handling of evolution of requirements, known core (system)

- **Naive Copy-and-Paste:** Requirements are written down for one system and are reused as is for another system by copying them from the original specification into a new one.
  Immediate goal(s): none (this is the simplest approach but not systematic at all)
  Supported goal(s) or concepts: handling of evolution of requirements, known core (system)

- **Parameterization:** Requirements are formulated with abstract parameters. Changes in technology, architecture or market demands can hence be denoted as differently instantiated parameters.
  Immediate goal(s): reusable formulation of requirements
  Supported goal(s) or concepts: explicit description of variability

- **System scope:** Systems are viewed with respect to their commonality and variability. A scope of similar systems is established, which systems share an amount of commonality and that differ at manageable points.
  Immediate goal(s): handling of system scoping
  Supported goal(s) or concepts: explicit description of variability

- **Variability coupling:** Requirements are coupled by a family of relations including alternatives (mutual exclusions), non-exclusive alternatives, options, et cetera.
  Immediate goal(s): explicit description of variability
  Supported goal(s) or concepts: none

The presented generic concepts can exist in different characteristics and this list of generic concepts can now be used to identify the need for (improved) reuse approaches. Most of the known reuse approaches are constructed out of these concepts. As an example, we describe the composition of software product line systematic on the level of requirements. Classical software product line approach for
requirements [Clements, 02] consists of the following concepts: dependency network, detection and elimination of redundancy, known core (system), system scope, and variability coupling.

6 Make it work in practice – an example

This section presents an example of how organization-specific knowledge concerning reuse can be applied for use in a project. This example uses widely known net diagrams for the documentation of project characteristics. Argumentation-based rationale models describing the reasons and arguments for the choice of generic reuse concepts are used based on project specific characteristics, goals, and constraints. The rationale models presented are based on concepts introduced in [Dewar, 02] and [Toulmin, 58].

The idea is to identify the current project situation and its goals and constraints by guided interviews especially revealing project characteristics having an impact on the reuse approaches. These characteristics are the input for the argumentation-based rationale models representing the experience of other projects in similar development work. These models give hints, how the project can improve its reuse tasks by describing the needed level of realization of generic reuse approaches. This can also be supported by a collection of practical patterns for a concrete proposal.

Imagine we are working for Ruritanian Autos in 1963. We wish to produce an innovative small car on a limited budget as state sponsorship is withdrawn. We could select an innovative engine as an attractive marketing feature. Imagine Ruritanian Autos concluded in a marketing study that the development of a small, clean and quiet Diesel engines is technically possible now and that the market also wishes to have a new kind of engines. A project is established to develop this new engine and the project team consists of domain experts, developers and managers – all highly motivated and trying to do their best.
Since management is convinced that Diesel engines for personal cars will be successful in the market and will possibly have to be advanced for future requirements after the installation of the project, it is decided that a reuse approach should be followed so that future advancement of that innovative engine may be easily realized.

In a guided interview the project manager reveals the following net diagram, representing the reuse concepts and the degree of their realization in the Diesel project (see Figure 4).

The degree of realization of the generic concepts is only for “abstraction layers”, “decomposition layers”, and “parameterization” distinctive, because these concepts are also needed for a systematic development of an innovation. The obvious reuse concepts are not pronounced in this case. The project has to decide which concept has to be realized to what degree.

Imagine that other projects at Ruritanian Autos already have experience with the development of innovative systems and how these projects succeeded with respect to the applied reuse approaches. An overview of the collected knowledge is given in Figure 5. The figure shows the principal features of a rationale model, namely chains of assumptions leading to a conclusion, supported by the business’ goals, and limited by some practical business constraints. Several of the assumptions seem to be load-bearing and hence they are depicted with a signpost i.e. a trigger denoting that a change of the described precondition may change the argumentation and ultimately the conclusion (the reuse concept options).
The project characteristics can now be mapped to the experiences of other projects that developed innovative systems. The depicted overview shows in extracts the principal chains of argumentation that were established by these projects. Essentially there are three signposts describing triggers from the environment of the system.

**Figure 5 Rationale model (extract) for the application of reuse concepts in projects developing innovative systems.**

The first signpost asks if the future advancement of the system is under 3. If this is the case, the argumentation depicted in this overview is broken and must be adapted. In case it is not true, the argumentation leads to the assumption that innovations must be supported by reuse. This can be done by either of the depicted reuse concept options (that are in this case specific concepts for the development of innovations). Each option can be discussed in an own argumentation-based rationale model similar to the overview argumentation model, but with a more detailed chain of arguments with respect to the specific characteristic of the concept and its environment. To go not beyond the length of this paper, we cannot show an example of such a kind of a rationale model, but it can be viewed at [http://www.scenarioplus.org.uk/download_diagrams.html](http://www.scenarioplus.org.uk/download_diagrams.html). Such a rationale model shows that the reuse concept options “systematic description of abstraction layers” and “systematic description of decomposition layers” effectively support future
advancements of the systems. They support the system understanding and a good documentation of the system understanding. Furthermore they are helpful if the system is a technical innovation and a certain amount of different stakeholders and experts is involved, which is depicted as its own signpost in the overview.

The signpost “system is part of a system family” asks if the system in view is part of an already existing product family. A more detailed and option-specific rationale model asks if the existing product family is described in a variability model and if any coupling exists between requirements. Using a variability model and coupling in an already existing product family has impact on the needed reuse concept options for the innovation since it needs to follow the agreed approach.

Each reuse concept option can be linked to one or more patterns, realizing the underlying technical goal of the reuse concept. The patterns are described briefly and consist essentially of one small and coherent example.

In the case of the Diesel development team at Ruritian Autos, we can conclude that it is a technical innovation and we expect a broad application of the new engine. So we need systematic abstraction and decomposition layers. The Diesel engine is part of an already existing family (family of engine ECUs), but the family is neither supported by a specific variability modelling nor a requirements coupling, so these aspects cannot be covered. The same holds for the identification of a system scope, because the existing product family of engine ECUs is not comparable to the innovation and the innovation is solitary, so that additional products currently do not exist.

7 Conclusion

Software reuse concepts – as crucial factors in software development and especially in the context of embedded systems – necessarily have to be adapted to different software development domains, layers and artifacts. Current literature regarding product lines – a high gloss reuse concept – offers estimation methods about the benefits of product line concepts. Still none of these models provide satisfactory recommendations about the best fitting reuse concept to a concrete project with distinct characteristics and goals. Our contribution to this challenge consists of supporting organization-specific learning during a project. The relevant project characteristics and goals for reuse approaches were identified and net diagrams which document the current status of the project (in consideration of the chosen reuse approach) were presented. Furthermore, an argumentation based rationale model was exemplarily introduced which facilitates the documentation of organization-specific behavior.

Essentially the presented approach is inherently used in current industrial practice, because teams are consulting each other on successful or failed concepts, but the knowledge is not documented except in the minds of the people involved. In addition, their extent is limited to personal communication. The presented approach tries to enhance the accessibility of the knowledge that already exists. The approach is easily applied and flexible and does not generate much overhead – the documented knowledge exists – it merely says: “Write it down”.

Future work may concentrate on elaborating a library of generic reuse concepts. An assessment model for reuse only – considering the goals of systematic reuse –
seems possible with such a library. In the same way, a collection of reuse patterns
could be created describing archetypes that were already successfully applied in
practice. Other ways of describing experiences may turn out more convenient – this
paper chose a simple and streamlined approach, since complex experience bases have
not fared well in practice. Looking ahead, a compromise between a complex
experience base and a streamlined approach may turn out to be the silver bullet.

Acknowledgements

We want to thank Ian Alexander (http://easyweb.easynet.co.uk/~iany/) for his
excellent support in various discussions about the argument based rationale models
and for his valuable advices regarding the identification of generic reuse concepts.

References

Development: The KobrA Approach. Software Product Lines: Experi-ences and Research
[Bayer, 99] Bayer, J., Flege, O., Knauber, P., Laqua, R., Muthig, D., Schmid, K., Widen, T.,
DeBaud, J.-M.: PuLSE: A Methodology to Develop Software Product Lines. Proceedings of
the 5th Symposium on Software Reusability: Bridging the Gap between Research and Practice,
Los Angeles (1999) 122-131
Line Engineering Industrial Nuts and Bolts. Fraunhofer IESE, 113.03/E, Kaiserslautern (2003)
Information and Software Technology September, (1990) 643-651
[Bosch, 02] Bosch, J.: Design and Use of Software Architectures: Adopting and evolving a
Workshop on Software Architectures for Product Fami-lies (IW-SAPF-3), Ed. F. van der
for Product Line Economics (SIMPLE). Software Engineering Institute of the Carnegie Mellon
University, CMU/SEI-2005-TR-003, Pittsburgh (2005)
[DACS, 06] DACS Gold Practises Website: Software Acquisition Gold Practice™ Assess


Session 4: Organizing Requirements

Using Requirements Engineering Patterns for Organizational Learning
Lars Hagge, Kathrin Lappe, Deutsches Elektronen-Synchrotron (DESY), Germany

Using Wikis to Manage Use Cases: Experience and Outlook
Björn Decker, Eric Ras, Jörg Rech, Fraunhofer IESE, Kaiserslautern, Germany
Bertin Klein, German Research Institute for Artificial Intelligence (DFKI), Germany
Christian Hoecht, Chair of Pedagogics, Technical University of Kaiserslautern, Germany
Using Requirements Engineering (RE) Patterns for Organizational Learning

Lars Hagge
(Deutsches Elektronen-Synchrotron, Germany
lars.hagge@desy.de)

Kathrin Lappe
(Deutsches Elektronen-Synchrotron, Germany
kathrin.lappe@desy.de)

Keywords: Requirements Engineering, Learning Software Organization, Patterns
Categories: D.2.1, D.2.9, H.3.5

1 Introduction
Organizational learning aims at learning from experience and incorporating the emerging knowledge and skills into the organization’s business processes. One way of achieving this goal is business process improvement approaches which for example customize process frameworks to an organization’s needs. But such process changes require time and effort which are scarcely available in project-oriented organizations. An alternative approach is the less formal and more situational learning based on exchanging experience, for which patterns are an established and well-known format (e.g. [Gamma et al. 94], [Buschmann et al. 96], [Eriksson and Penker 00]). Patterns are characterized by presenting engineering knowledge in self-contained units that address a particular context. They provide guidelines for good engineering, where engineering is the “art of applying scientific and methodological knowledge to practical problems”.

The paper describes the applicability of patterns to organizational learning and to the improvement of RE processes. It first briefly introduces RE patterns and the online pattern repository REPARE, then discusses the relation of RE patterns to prominent concepts of learning, and finally describes some applications of RE patterns to organizational learning.

2 RE Patterns and REPARE
RE patterns aim at making requirements engineering knowledge and experience reusable. They are derived from case studies and describe RE practices which have been successfully used and observed in two or more projects under comparable conditions. The patterns are written in an instructive format that makes the pattern content easily accessible. They are accessible via an online RE pattern repository (REPARE, [Repare 05]).
2.1 Objective

RE patterns set out to make good RE practices available for project teams on the job. They target practitioners from small and medium organizations without dedicated RE resources who often encounter project teams or environments with no or only little experience in requirements engineering. RE patterns aim at bridging the gap between existing RE knowledge and its successful application in different kinds of projects by improving the accessibility of RE knowledge and experience.

2.2 The Pattern Vector

RE patterns contain pairs of problems and solutions, where problem statements are explicitly specified in terms of two conflicting forces. The forces are directly referring to conditions or situations as they are encountered by project managers. The solution is provided in terms of a proposed action which has been observed to solve the conflict by compensating the forces. A formal structure called the “pattern vector” has been developed for capturing the essence of an RE pattern. It contains the RE task $T$ which needs to be conducted, the two conflicting forces $F_\text{\&}$ and $F_\text{\&}$ which characterize the problem, and the action $A$ which can solve the problem [Hagge et al. 05].

$$P = (T, F_\text{\&}, F_\text{\&}, A).$$

A generic pattern statement can be created from the pattern vector using

$$\text{IF } F_\text{\&} \text{ BUT } F_\text{\&} \text{ THEN } A \text{ TO } T.$$  

2.3 An Example for an RE Pattern

“Detail the Specification by Writing Test Cases” is a popular pattern which is contrasting a number of text books. It recommends that if a specification turns out to need improvement after it has been signed and the project is under way, an option is to provide detailed test cases in addition to the requirements. After generalization, the pattern statement reads:

| IF | the project should advance to meet the milestones |
| BUT | the project should step back and clarify the requirements |
| THEN | detail the specification documents by writing test cases |
| TO | validate and verify requirements |

The generic statement helps to decide at a glance on a pattern’s relevance and applicability, but for practical purposes, patterns have to be explained in a more elaborate format like in established pattern collections.

RE patterns are named after the proposed action. Their task is split into an objective and a context, the forces are embedded into a problem description, and the action is called the solution. The usability of the solution is detailed by listing application areas and constraints for its successful implementation, and by describing
additional experience, known uses, and related patterns. The full-text example for the above pattern can be obtained from REPARE.

2.4 RE Pattern Mining

The initial collection of RE patterns has been collected by a working group with participants from industry, research and academia [Lappe et al. 04]. Patterns have been derived from two or more similar observations which have been reported in different projects (Figure 6).

![Pattern Mining Diagram]

(Figure 6: Capturing Patterns from Observations in Case Studies.)

The procedure which has been applied for pattern mining includes four basic steps [Lappe et al. 04]:

4. **Case studies** are collected from real-world projects. They contain accounts of important events and experience from projects.
5. The case studies are analyzed and reorganized into a set of observations, which describe the reported events in the format of the pattern vector.
6. To identify patterns, the entire set of case studies is searched for identical observations from different projects. These observations are marked as **pattern candidates**.
7. The pattern candidates are elaborated into **pattern descriptions**, which are then published and made available for reuse in project organizations.

The working group on RE patterns has developed a routine program for one-day pattern workshops according this procedure. The group has held about ten of these workshops, identifying more than 80 observations and more than 20 RE pattern candidates.

2.5 The RE Pattern Repository REPARE

The pattern repository REPARE [Repare 05] collects the resulting RE patterns. It uses the pattern vector to store the patterns and the observations which were reported in the case studies. The repository uses the vector elements and additional attributes for filtering observations or patterns. Thus, it allows for searching patterns which are applicable to a specific project situation by letting the user specify characteristics like

<table>
<thead>
<tr>
<th>Objective</th>
<th>Context</th>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observation 1.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Observation 1.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Observation 1.3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Observation 1.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Observation 1.5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Observation 2.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Observation 2.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Observation 2.3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Observation 2.4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
e.g. the forces, the RE activity they are conducting or the quality goal they intend to reach. The search form for finding patterns (Figure 7) is built as a cloze text for building queries which allow users to fill in the blanks. REPARE returns a list of patterns that provide RE experience with regard to the given criteria in combination with optional additional free search terms. By choosing more or less descriptors and modifying the threshold, it is possible to gradually change the precision of the request.

I am looking for Requirements Engineering Patterns which ...

For each pattern, a short structured overview is available which links to the pattern’s full text for download. In addition to the search functionality, REPARE provides hyperlinks for navigation to related patterns and to the case studies on which the pattern is based. Discussion forums invite users to provide feedback on individual patterns or the repository itself [Hagge et al. 05].

3 RE Patterns and Learning

Many definitions describe learning as a process, for example the process of gaining knowledge, understanding or skills by study, instruction or experience [Merriam-Webster 05], or a process whereby knowledge is created through the transformation of experience [Kolb 84]. Learning yields knowledge and skills which are necessary e.g. for process improvement and is in this sense the basis for any change.

Learning can be performed by individuals as well as by organizations. While individual learning is frequently conducted within the framework of organized activities such as for example seminars and classroom teaching, coaching or e-learning, organizational learning often involves less formal approaches like e.g. experiential learning. This section examines how RE patterns relate to some prominent concepts of learning.

3.1 RE Patterns and Learning Organizations

Organizational Learning is the process by which an organization acquires the knowledge and skills it needs to survive in its environment. A learning organization is
an organization which facilitates organizational learning, i.e. which has a culture and methods in place which possess the capability of acquiring the required knowledge. Learning organizations can be identified by characteristics which are referred to as the five disciplines [Senge 90]:

- **Personal mastery** is the ability to openly see reality as it exists.
- **Mental models** describe the ability to compare reality (or personal visions) with perceptions and to converge them into a coherent understanding.
- **Shared vision** is the ability of a group to develop and hold a shared picture of a mutually desirable future.
- **Team learning** refers to the ability of groups to engage in dialogue rather than discussion.
- **Systems thinking** is the ability to see interrelationships, think in context and understand the consequences of actions on other parts of the system. It builds on the possession of the previous for characteristics.

Learning organizations rely on both individual and organizational learning: The first two characteristics are individual, while the other three are group based. Software organizations can use RE patterns (i.e. access the repository, but also hold their own internal pattern workshops) as an instrument on their way to becoming a learning organization, as RE patterns

- require individuals to report and discuss their positive and negative RE experience in a team (as a step to personal mastery)
- provide an abstract model for both guiding and reflecting RE activities (as a basis for building mental models)
- require an analysis team to build a consensus on the commonalities of different observations (as an effort of team learning)
- can be related and standardized in their names as a step towards a common RE (pattern) language (as a basis for a shared vision)

### 3.2 RE Patterns and Experiential Learning

Experiential learning includes both learning by and from experience. It focuses on learning by directly encountering phenomena, as learners are led into situations where they can practice their skills and gain experience, but it also includes abstracting and reflecting experiences, this way allowing organizations to learn from each other.
A frequent model, the experiential learning circle, consists of four major stages (according to [Kolb 84]):

1. Concrete experience is gathered by seeing the consequences of an action.
2. The particular action is reflected, it becomes predictable and reproducible.
3. The action is put as a general principle which is adaptable to other situations.
4. The action is applied under new circumstances, generating new experience.

Experiential learning mostly starts with gathering concrete experience, but in principle it could start at any of the four stages. If learning takes place, the circle should rather be presented as a spiral. The RE Pattern mining procedure which has been described in section 2.4 corresponds to experiential learning if pattern application is included:

5. Pattern workshops correspond to sharing concrete experience.
6. Identifying pattern candidates from observations corresponds to reflection.
7. Writing patterns corresponds to generalization.
8. Using patterns corresponds to testing concepts in new situations.

The similarity of the approaches has been experienced by the working group on RE Patterns, whose members confirmed the learning value of RE pattern mining [Lappe et al. 04]. As a consequence, some workshop participants have proposed to hold RE pattern workshops within their software organizations.

3.3 RE Patterns and Experience Factories

The idea of experimental learning is also driving the experience factory scheme (cf. Figure 9) described by [Basili et al. 02], which again also matches RE pattern mining. Experience from project organizations is collected after a project has ended. It is analyzed and packaged by an independent organization which collects experience from different projects. Generalized experience packages are fed back to the planning lifecycle of other projects. The working group on RE patterns has established the experience factory at the community level, but the concept might also be applied at smaller organizational scales.
The pattern concept provides a formalized structure for documenting and analyzing experience in a comparable way. The pattern repository acts as experience base by storing formalized observations from real-life projects as well as generalized patterns which have been synthesized from comparable observations.

The resulting patterns have thus passed through two steps of analysis: First, every observation (i.e. experience package) is formalized and generalized into the pattern vector format. Then, if two or more comparable observations yield a pattern candidate, they are further elaborated and synthesized into an RE pattern.

### 3.4 RE Patterns and Single- and Double-Loop Learning

The previously discussed concepts underline that learning improves the way people are behaving in and reacting to specific situations. The concept of single- and double-loop learning offers an approach to understanding how such behavioural changes can be achieved. It is built on the assumption that people are guided by mental maps or theories of action.

A mental map can be assumed to contain three basic building blocks:

- **The governing variables** are the dimensions and define the space in which people can act.
- **The action strategies** comprise the set of possible moves and actions people could take.
- **The consequences** are the obtained results of their actions.
If the consequences are not compatible with the initial expectations, the behaviour has to be modified – this modification is understood as learning. Learning can either modify the action strategies, i.e. propose a different set of actions to achieve the expected results, or it can question and change the governing variables, i.e. change the boundary conditions for the behaviour. The first is called single-loop learning, the latter double-loop learning.

RE pattern write-ups exhibit all the three elements of a theory of action:

- The forces (often supplemented with additional quality goals) describe the governing variables, i.e. the dimensions which need to be observed.
- The action proposes an action strategy for a specific set of variables, while related patterns propose additional and alternative actions.
- The observed consequences (not part of the pattern vector, but listed in the full pattern write-up) describe the results which have been obtained so far.

The RE Pattern Repository, REPARE, makes patterns accessible through any of these elements and may therefore be used for both single- and double-loop learning:

- In single-loop learning, REPARE can be queried for a given situation, i.e. a pair of forces which reflects the set governing variables, and be checked whether an appropriate proposed action is available for the intended result. This way, it can propose novel strategies for achieving an intended result.
- In double-loop learning, REPARE can be queried for an intended result, i.e. a task and a quality goal. It then has to be analysed which forces and additional preconditions have to be present to achieve the result, giving rise to a review and possible adaptation of the organization’s governing variables.

4 More Examples for Learning with RE Patterns

In organizational learning, many applications are possible for RE patterns which are available in a structured repository. They range from provision of material for various types of instructed and self-based learning, over support of specific learning concepts as discussed in the previous section, up to an RE maturity assessment and an evaluation of the organization’s RE learning needs. The later two are introduced by examples in this section.

4.1 Pattern-Based RE Maturity Assessment

A collection of existing patterns can be used as a basis for benchmarking the relevance of certain RE topics and conflicts in an organization. The occurrence or lack of “known conflicts” (i.e. conflicts which are recorded in the available RE pattern database) is used to identify potential fields for improvement of RE activities. Organizational learning should address strategies for both avoiding and possible remedies of these issues.

To use RE Patterns as a foundation for assessing an organization’s RE process maturity, it first has to be determined which “known conflicts” are present in the
Proceedings, LSO+RE 2006, Version 1.0

The conflicts are intentionally controlled (i.e., handled in the same way as proposed by the RE patterns). The ratio of controlled over present conflicts serves as an indicator of the RE process quality, while the ratio of conflicts which are intentionally controlled indicates the process reproducibility.

Figure 11: Result from an RE Maturity Test Using RE Patterns.

In the next step, the different conflicts can be associated with the different RE activities, thus allowing computing maturity levels for these activities. Figure 11 shows an example of such a RE maturity assessment which has been computed by REPARE.

4.2 What to Learn: RE Patterns as Indicator for Learning Needs

Taking the previous approach further, RE patterns can also be used for identifying the issues which need to be addressed in organizational learning. For this purpose, simply a given set of set of case studies has to be analyzed to determine which forces and which conflicts are reported most frequently. According to the current content of REPARE, RE is mainly suffering from three conflicts:

- The specification is not yet adequate and would need more time to complete, but meeting the milestones requires starting development nevertheless: faster and more efficient methods for building specifications are needed.
• Changes are occurring in the project up to the very last minute, although the specification should have been frozen for implementation and testing long before: methods for handling changes without risking the project are needed.
• Products need to employ the latest technology, even if neither customers nor developers are yet capable of handling it properly: methods are needed for keeping all the stakeholders technologically up to date.

REPAIR is offering possible solutions for all these conflicts.

5 Conclusion

RE Patterns offer an effective basis for organizational learning with applications ranging from short and specific learning units up to strategic concerns. They can be used for process optimization within a single organization as well as for collecting and exchanging know-how across several organizations. An online RE pattern repository is freely available on the Internet. The initial experience confirms the flexibility and value of RE patterns for organizational learning.

References


Using Wikis to Manage Use Cases: Experience and Outlook

Björn Decker, Eric Ras, Jörg Rech
(Fraunhofer Institute for Experimental Software Engineering, Germany
{bjoern.decker, eric.ras, joerg.rech}@iese.fraunhofer.de)

Bertin Klein
(German Research Institute for Artificial Intelligence
bertin.klein@dfki.de)

Christian Hoecht
(Chair of Pedagogics, Technical University of Kaiserslautern, Germany
hoecht@rhrk.uni-kl.de)

Abstract: Sound requirements engineering is one of the key practices for a successful software project. However, the risk exists that requirements documents degenerate into a write-only documentation that does not reflect the experiences with the system under development gained throughout a project. Furthermore, “classical”, self-contained templates for requirements documentation do not reflect the interrelated nature of requirements. For example, a user description is referenced in various places in a requirements document. In this paper, we present how freely available Wikis support requirements engineering based on a textual use case approach, in line with the following considerations: First, we believe that the agile nature of Wikis support the creation and change management of requirements. Second, as a hypertext system, they address the intertwined nature of requirements. Third, Wikis provide a low-effort, easy-to-test approach to requirements engineering, since most of them are open source and server-based.

Keywords: RISE, requirements engineering, Wiki, use cases
Categories: D2.1, D2.9

1 Introduction

The creation of sound requirements is supported by a number of well-established methods (e.g., [Cockburn, 2000]), templates (e.g., [Robbins, 2005] [Robertson and Robertson, 2005]), and tools ([Anonymous, 2005]). However, the CHAOS report of the Standish Group [SoftwareMarketSolutions, 2006] reports consistently that poor requirements are among the main reasons causing problems in IT projects. In the opinion of the authors, most approaches currently need a rather high initial investment into time and money. Therefore, the improvement of requirements engineering often falls prey to project pressure. Furthermore, requirements management is a creative and communicative process. In the experience of the authors, current approaches are often used in a way that causes relatively high administrative overhead to the involved persons.
In this paper, a hands-on approach for the agile creation of use cases based on Wiki technology is presented. This approach has been used in three projects. One of these projects was the RISE project, which ran over three years, involving 12 people, and creating about 100 requirements and related documents (i.e., Wiki pages). Information about the other projects cannot be provided due to confidentiality reasons. These case studies show that our approach has the potential to support learning software organizations. In particular, based on the low cost and intuitive usage of Wikis, our approach can provide a starting point for SMEs to gradually improve their requirements engineering process and hence, the quality of the requirements documentation.

Besides its direct relation to supporting requirements documentation in a single project, the approach supports organizational learning: First, an organizational learning circle is established by reusing requirements documents in follow-up projects. Second, Wikis can act as portal to further topics relevant to employees of an organization, including documents dedicated to organizational learning. In summary, Wikis serve as a technological platform to support organizational learning.

The paper is structured as follows: Section two explains the technical features and underlying philosophy of Wikis. The third section presents adaptable requirements document templates derived from approaches freely available on the Internet (e.g., [Robbins, 2005]) or to be found in literature (e.g., [Cockburn, 2000]). The fourth section lists requirements for Wikis to be used in a requirements engineering setting, which allows the reader to select a Wiki suitable for his needs (e.g., from [Portland Pattern Repository, 2005]). The fifth section describes Patterns on how to use Wikis for requirements engineering. The final section summarizes the paper and gives an outlook to future work, in particular, how reuse of requirements engineering can be further supported.

2 Wiki Philosophy

Wikis facilitate communication through a basic set of features, which allows the project team to coordinate their work in a flexible way. From the authors’ point of view, these basic features are: one place publishing, meaning that there is only one version of a document available that is regarded as the current version; simple and safe collaboration, which refers to versioning and locking mechanisms that most Wikis provide; easy linking, meaning that documents within a Wiki can be linked by their title using a simple markup; description on demand, which means that links can be defined to pages that have not been created yet, but might be filled with content in the future. Furthermore, the simple mechanism of URL allows easy reference and thus traceability of Wiki content into other software documents like code. Since the Wiki syntax allows the manipulation of text only, text-based approaches for requirements engineering like use cases can be supported best.

Besides those technical aspects, Wikis foster a mindset of a fit-for-use, evolutionary approach to requirements documentation and management. The creation of requirements is an evolutionary process that reflects the current understanding of the application domain and the system under development. The approach of Wikis – in particular, Wikipedia as the most prominent representative [Wikipedia, 2006] – demands that an initially created document is adequate for its intended usage (fit-for-
use). This initial version is then extended based on the demand of the people using this document.

However, the authors identified certain shortcomings, which may render the application of Wikis inappropriate: A) *Uncontrolled content sprawl* leads to a decrease in content quality and navigational structure. A well-defined information structure mitigates this problem. However, some project teams might not be able or willing to invest the additional effort into maintaining the integrity of the Wiki content. B) The missing *replication* of Wiki content might lead to problems in settings where project members need to work offline from time to time. FlexWiki [Jonathan., 2006] has a replication feature, but the user interface of the offline version is subject to improvement. C) The export to *self-contained documents* – in particular, in settings where the requirements template is determined by the customer – is also not supported. D) Finally, no overview of documents belonging to a certain version can be generated.

### 3 Requirements Templates

This section describes a basic requirements document structure, based on use cases in combination with some additional documents like project descriptions and overviews as depicted in Figure 1. Furthermore, adapting this structure to the users needs is described. This approach provides the basics for a quick, ready-to-use approach, which then can easily be adapted to further project needs. In the RISE project context, this approach – with some initial extensions – required about two hours of explanation to the project members.

![Figure 1: Basic Document Structure](image-url)
These templates provide a structure and an initial explanation to the users, e.g., which further types of documents should be linked. Therefore, they address the intertwined nature of requirements not reflected by the linear representation of a self-contained requirements document. Those relations – and their underlying semantics – are depicted in Figure 1 by “Related”, “PartOf” and “RefinedIn”. They are used by copying the content of the template into a newly created document. When each template contains a reference to itself, the instantiated document will contain a link to the template (“IsA” relationship). The following types of documents exist:

- **Project Homepage** is the entry page to the requirements. It contains information about the project such as its mission, and links to an overview of the requirements. Furthermore, it gives initial information for people joining the project.
- **User Homepages** contain information about the people involved in the project. Each User Homepage should contain the role of the person in the project as well as contact information.
- **Overviews** list the requirements documents according to their document type.
- **User Stories** are short prose specifications of system interactions. They should contain at least the actors involved and the responsible author. This style allows other users to specify their requirements in a consistent way.
- **Actors** define the roles involved in a use case or a user story.
- **Use Cases** are a structured representation of a user story. The actual structure is dependent on the needs of the project. A potential structure based on [Cockburn, 2000] and [Robertson and Robertson, 2005] is as follows: The **Primary Actor** is the responsible person to execute this use case. **Stakeholders / Further Actors** are involved or interested in the execution of the use case. **Preconditions** describe the state of the system before the use case can be executed. **Minimal guarantee** describe the state of the system independent of the use cases’ successful execution. **Success guarantee** is the state of the system after the main success scenario has been executed. The **Main Success Scenario** contains the steps of a successful execution of the use case, while **Extensions** add variants to the steps of the main success scenario.

This document structure can be further extended to the actual needs of the project. These extensions might be demanded by a project, since use cases cover about one third of the requirements [Cockburn, 2000]. In general, there are two types of extensions: Adding further sections to an existing template or integrating new templates into the document structure. In any case, it should be checked whether the information is already available in another document. Redundant captured information must be factored out to a separate page. The following extensions proved to be sensible in at least one project where we applied our approach:

- additional information about the project (e.g., software frameworks),
- project management related documents like meeting minutes,
- additional classifications beyond document types (e.g., according to scope or level classification [Cockburn, 2000] by adding additional sections to use cases and listing them on separate overviews),
- states defined on separate pages to support their consistent usage.
4 Requirements for Wikis

The previous section explained the structure of the information to be captured in a Wiki. This section describes the required features for a Wiki that could support this kind of work. This description will focus on features that are of particular relevance to requirements engineering. Further features can be found in [Vökel, Schaffert, Kiesel, Oren and Decker, 2005].

This overview of relevant features can be used to evaluate Wikis available in an organization as well as for selecting new Wikis. For this selection, [Portland_Pattern_Repository, 2005] provides a general overview of about 200 Wikis, while [Decker, Ras, Rech, Klein and Hoecht, 2005] gives an overview of Wiki features that support Software Engineering in particular.

To further support the evaluation process, the features were classified according to their importance, into “required” (R) and “nice-to-have” (N) features. The features are structured into five groups: search, structure, versioning, access, and misc. To facilitate identification, each feature is given a unique identifier.

Search supports locating relevant pages in the Wiki, whether it is for editing purposes or for linking them to other documents. Furthermore, the consistent use of document naming schemes is supported when those naming schemes provide a benefit during search:

- S1, fulltext-search (R): The Wiki should have at least a full text search. Without this feature, it will become difficult to maintain an overview of the Wiki content.
- S2, direct CGI access (N): A query to the search engine should be specified by a URL. This allows predefining queries for basic consistency checks.

Structure features are about classifying pages to generate overviews of the content. Those features, with the exception of the last one, allow classifying the content locally, i.e., from the page that is being edited. A suitable Wiki for requirements engineering should have at least one of those features. Therefore, no classification is given for any feature.

- St1, backlinks [Wikimedia, 2006a]: Backlinks are an overview of the pages that link to the current page. Therefore, they provide a powerful tool for simple consistency checks. Classification using backlinks works as follows: A page representing a category is set up. By referencing this category page, pages state themselves to be part of the category. By using backlink on the category page, one can get an overview of the pages belonging to this category. However, a category page will be linked for mere navigational purposes (e.g., from the project homepage), which adds a link that does not belong to this category.
- St2, categories [Wikimedia, 2006b]: The categories are quite similar to the backlink feature. However, there is a clear separation between links that refer to pages for classification and those used for navigational purposes.
- St3, tags [Alrubaie, 2006]: Tags are classifications that can be attached to Wiki pages. They can be provided individually by each user and do not need previous definitions like categories.
- St4, table of content [TikiWiki, 2006]: This feature is similar to tables of content found in word processors. It allows structuring and aligning the
content of a Wiki, which provides the basis for serialization into a linear document.

Despite *Versioning* being one main feature mentioned in the section on Wiki philosophy, some Wikis do not have it. However, since it builds the basis for safe collaboration, the authors do not recommend the use of Wikis without this feature.

- **V1**, versioning / rollback (R): Editing of pages should be versioned in general, and one should be able to go back to an earlier version (rollback).
- **V2**, diff (R): The differences between versions should be made visible (diff). Without this feature, rollback to an earlier version is not feasible.
- **V3**, linkable versions (N): Different versions of a page should be referenced via a unique URL.

*Access* features cover the logging of read / write to Wiki pages and access restrictions.

- **A1**, user authentication (R): In order to track edits and to allow notification about changes, the user should be logged in before editing a document.
- **A2**, access restrictions (N): Dependent on the project (e.g., whether the customer should be granted access to the Wiki), one might need to define separate read and edit access rights based on users or user groups, or roles. Access rights might be set for a single page or for a group of pages.

*Misc* features are relevant features that do not fit into any of the categories above

- **M1**, renaming (R): Renaming means that the change of a page title should automatically update all links to this page. For example, this might be needed to perform changes in the naming scheme.

All these features, except the access feature, can be easily tested in the so-called sandbox that most Wikis provide on their homepage. Hence, an installation of the software is not needed.

### 5 Patterns

This section describes - based on a pattern approach - how a project team should use the Wiki to document their requirements. Patterns help to quickly establish an agreed set of rules on how the Wiki should be used in the project team.

These patterns are based on the experience made within RISE and support a demand driven comprehension of how to use Wikis for requirements engineering. The patterns are described using the “classic” form defined in [Alexander, 1977]: Followed by the name, the problem description helps to determine whether a pattern is applicable or not. Solution describes how the problem is solved, while cause gives an explanation on why the solution addresses the problem. To further facilitate application, the following additional classification is provided: *Team Practice* patterns primarily focus on practices carried out by the whole project team. *Personal Practice* patterns are performed by a member of the project using the Wiki.
Use Your Name, Use Your Homepage (Personal Practice):

*Problem:* Finding the contact person concerning a certain document or as specific edit activity is difficult.

*Solution:* Register with your actual name. Furthermore, mark “your” documents with a link to your homepage. The homepage should contain contact information (email, phone).

*Cause:* When contact information is easily accessible, barriers against communication are lowered.

Social Tagging (Personal Practice):

*Problem:* Maintaining an overview of the status of the documents inside a Wiki is difficult. In particular, it is unclear which documents need to be reworked.

*Solution:* Tag pages that need rework (e.g., with a understandable link to a certain page or a category). For example, when using backlinks, refer to the page “ToDo”. Have a backlink on this page to list pages that are tagged with this concept.

*Cause:* By using a general Wiki feature for addressing workflow issues (e.g., ToDo), the usage of this feature is fortified and there is no need to learn further system features. This lowers the effort to get familiar with the Wiki.

Find Link (Personal Practice):

*Problem:* The link to a page is unknown.

*Solution:* Open second browser window to find titles of pages. Adhere to the naming schema. Use backlink to templates in order to determine potential candidates.

*Cause:* With an increasing number of documents, their titles are hard to remember. A consistent naming scheme makes it easier to remember titles.

Link with System design / Code (Personal Practice)

*Problem:* Traceability of requirements to the design down to the code is missing. The information acquired during implementation is not reflected in the requirements.

*Solution:* Link the page of requirements in the comments of the code. For example, in Java you could use the javadoc construct @link to capture those links.

*Cause:* The link is not directly available in the code.

Document early, Document Hypotheses (Team Practice):

*Problem:* Parts of the requirements are not written down, because information is vague. The documents do not reflect what was learned about the system under development.

*Solution:* Write down information, even if it is vague, and mark that it is vague (e.g., by a social tag). Open issues should be written down as well.

*Cause:* Going to a final state too early leads to poor requirements. Writing down vague requirements supports expressing the questions that need to be answered to remove this vagueness.

Set Window of Discussion (Team Practice).

*Problem:* Documents remain in an uncertain state for too long. Open issues are not answered.
*Solution:* Look at open issues and social tags denoting hypotheses on a regular basis. Then prioritize and assign a deadline to open issues.

*Cause:* Due to the self-determined growth of Wiki content, the effort needs to be refocused from time to time.

**Define Naming Conventions** (Team practice):

*Problem:* Links are hard to remember and find, the overview over a variety of links is lost.

*Solution:* Introduce naming conventions. As a minimum, documents should start with the name of the template they are derived from, followed by the title, and possibly the initials of the responsible author: DocumentType_<SpeakingTitle>_[Initials]. The speaking title should use a singular form.

*Cause:* A given structure makes it easier to remember names. When links are listed in alphabetical order, the documents belonging to the same type can be grouped easily.

**Define and Revise Communication procedures** (Team Practice)

*Problem:* The communication overhead is increasing.

*Solution:* Flashlight about current work and open questions at each meeting. Solve document interdependencies. Try to avoid discussing the actual work at the meeting. Have communication procedures that clarify at least when pages should be sent around via email (e.g., for review). In particular, clarify whether customers provide feedback directly to the Wiki. Discuss and change communication procedures if problems occur.

*Cause:* Since Wikis can support communication in many ways, the actual communication procedures need to be defined clearly to avoid misunderstanding. Furthermore, email is the main personal knowledge management tool, while social tags allow easy allocation of pages to be changed and thus, self-determined work on critical and most interesting content.

**Check Customer Wiki-Ability** (Team Practice)

*Problem:* Customers might not be willing or able to use the Wiki. An indicator for customer problems is when customer do not review the links sent around.

*Solution:* Clarify experience with Wiki when new customers join the project. If needed, show some customers on how to use the Wiki or determine a team member to act as customer proxy. Clarify how reviews are done (e.g., by showing how to edit a Wiki page). Ask customers to set up their user homepage and provide a sandbox inside the Wiki.

*Cause:* Using a Wiki and getting acquainted with its underlying philosophy might cause difficulties for “conservative” customers.
**One Wiki per Group** (Team practice)

**Problem:** If one Wiki is used in several projects, naming conflicts of the pages might occur. For example, two projects meet on the same day and want to create meeting minutes with the same title.

**Solution:** Create one Wiki per project. This could include framework development groups, which can be seen as a long running project. Furthermore, there should be one Wiki for the whole organization containing User Homepages, Project Homepages, and cross-project naming conventions. If available, the interwiki feature should be used to support easy linking of pages.

**Cause:** A member of a group normally works in the working space of one project. There is no need for overlapping (but maybe for linking) those working spaces. Furthermore, this lowers administrative overhead (e.g., assignment of access rights) by delegating those decisions to the groups actually using it.

### 6 Summary and Outlook

This paper has presented a hands-on approach to creating requirements based on use cases employing Wiki technology. By using the pattern “One Wiki per Group”, this approach also scales to application in multiple projects. However, cross-project (re-) use of documents is subject to the competencies of the people or their willingness to search other projects for suitable documents.

To further facilitate reuse across project boundaries as well as inside larger projects, further support to find relevant documents is needed. The underlying idea of a solution is to add document-metadata and typed relations to the requirements document structure (i.e., semantics). The resulting semantic Wiki [Wikipedia, 2005] supports the user in two ways: First, relevant or similar documents can be presented to the user while writing a requirements (e.g., use case) document. Second, consistency checks among documents can be performed (e.g., whether each user description is used in at least one use case). An overview of semantic Wikis in general can be found in [Völkel, Schaffert, Kiesel, Oren and Decker, 2005].

The application of regular and semantic Wikis will be further evaluated in upcoming projects. Among those projects are the development of an eGoverment application and the relaunch of a process documentation website.

### Acknowledgements

Our work is part of the project RISE (Reuse in Software Engineering), funded by the German Ministry of Education and Research (BMBF) grant number 01ISC13D. The results were applied in BelAmi (Bilateral German Hungarian Research Collaboration on Ambient Intelligence Systems), funded by the German Federal Ministry of Education and Research (BMBF), the Fraunhofer-Gesellschaft and the Ministry for Science, Education, Research and Culture (MWWFK) of Rheinland-Pfalz.
References


