Journal of Universal Knowledge Management, vol. 1, no. 2 (2006), 85-98
submitted: 15/5/06, accepted: 15/6/06, appeared: 28/9/06 © J.UKM

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
(Red-Expel GmbH, Switzerland
peter.kolb@red-expel.com)

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 examines the factors that influenced method adoption and discusses how the case affects learning software organization. Section 5 summarizes and concludes.

2 Discussion of the Research Approach

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.

In such a research setting, several risks need to be controlled. One such risk is bias of the subject's behaviour because the research question is known. The reported case exhibits no such bias: the case study was not prepared beforehand, but is an analysis with hindsight. As a consequence nobody involved in the case knew that activities and work results would be analyzed for understanding how RE is learned while practicing it.

Another risk is subjective judgment of what constitutes successful learning of RE and what the process of learning really is. This risk was reduced by using multiple sources of evidence, as well as by reviewing draft reports by key project members.

Finally, generalization of the discussed findings is not automatic: studied is a single case of escaping the requirements tar pit. The case presents data for testing theories that can provide analytical generalization. Any such theory must be tested by replicating its claims in a second or even third such case.

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.

Identifier	RID-001	Priority	High	Source	MRS-001
Title	Same software for same tasks in each project phase.				
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 different methods 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 2 outlines the sequence of methods that were tried as well as the reasons for failure of the first four methods and for success of the final method. Even though the first four methods did not yield success, the team succeeded to acquire requirements-relevant knowledge through their application. These pieces of knowledge are also summarized in Table 2.

Method	Acquired Knowledge	Reasons for Success or Failure
Word-Processor Template	Incomplete, vague, and isolated requirement fragments.	Team without mental model of RE. Shortcuts in RE process. Insufficient help for inexperienced analysts.
Informal Stakeholder Interviews	Product components with roadmap. Hardware configuration process with dataflows. Project dependencies, cross-product data flows. Some business processes and artefacts.	Little RE experience. Too much heterogeneous information. Focus on data collection at expense of analysis. No guidance for inexperienced analysts.
Beyer-Holtzblatt Contextual Design [Beyer, 97]	User and work modelling concepts.	Little experience in method. No access to end-users. Method not capturing all relevant kinds of data. Lacking tool.
UML UseCase and Class Modelling [Booch, 98]	Software and requirements modelling concepts.	Team with little experience in method. Method not capturing all relevant kinds of data.
Tailored Eriksson-Penker Business Modelling [Eriksson, 00]	Business modelling concepts, model customization, and requirement patterns. Hierarchical model of business processes, user-roles, concepts, artefacts, software and hardware components, and locations. Graphical user interface (GUI) prototype.	Thorough understanding of partially self-constructed method. Method adaptation supporting learning of RE. Stakeholders with experience in diagramming. Low cost tool. Method able to capture relevant kinds of data.

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 straying 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-hardware 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.

Business architecture is modelled by EP with four views. The *business vision view* describes a goal structure of a company and illustrates problems that must be solved to reach these goals. The *business process view* represents the activities and value created in the business and illustrates the interaction between the processes and resources. The *business structure view* describes the relationships among resources and products created. The *business behaviour view* describes the behaviour of important resources and processes.

These views were used for describing the use of the product to be developed. Only a subset of the EP diagrams was utilized, though. 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.

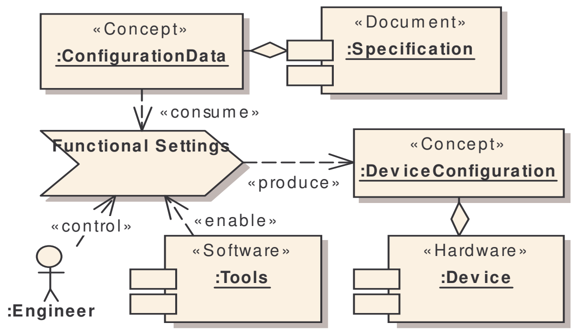

Figure 1: Example of a tailored Eriksson-Penker process diagram.

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 local product development company. A description was stored in the company's process database. Also, the team got the possibility to present that information to management. The reaction, though, was not strong enough to repeat the practice in other projects.

When the experience became visible to ABB Corporate Research, a research project 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] and natural language-oriented approaches to requirements engineering.

4 Discussion

The presented case is important from a learning software organization perspective in that it describes how requirements engineering practices are adopted by teams with initially little experience. Learning can be understood as a relatively permanent change in behaviour or in behavioural potentiality that results from experience [Hergenhahn, 05]. Learning can be observed by understanding previous behaviour, the conditions and process that mediate behaviour, and the resulting changes in behaviour of the subject.

The case presents the process of a software team for learning a requirements engineering method that was suited to their particular project situation. Even though the study focussed on the project team, learning also took place with the junior requirements engineer, the local product development company, and the global parent company.

The project team had no previous experience in engineering requirements for new products. Learning started with the understanding that the provided requirements specification could not assure that the right product would be developed. The learning process evolved from the company's standard process through acting thoughtfully, adopting a suggestion from a colleague with thorough knowledge in software product development, and implementing a common industry practice, to arrive at tailoring an approach that was suggested by tool documentation.

Every step of this evolution was started by the understanding that the project situation still needed improvement. In every such step, the approach that was selected depended much on the rapid availability of information that identified the approach. The sufficient prerequisite for trying out an approach was that the team's experience and understanding of the project situation did not indicate failure a priori. No effort was undertaken to systematically widen the team's understanding of generally known requirements engineering methods. Rather, the need for immediate results let the team engage in learning on a trial-and-error basis.

In the presented case, trial-and-error was not only the natural approach to learning, but was also efficient. The first four trials took only one calendar-month. The successful approach was tailored and used for completing the requirements during two calendar-months. The learned enabled the team to engineer requirements at a high level of quality one order of magnitude faster than with the company's standard practice.

Every considered method contributed positively and negatively to the finally adopted approach. Some practices, like the use of patterns to describe the modelling language, were retained without alteration. Other practices, like models for describing the structure of an artefact, were considerably tailored and interpreted in a new context. Some of the practices were not documented. For example, the use of graphical user interface (GUI) prototypes was a practice that all team members deemed important and useful, but no literature or person was consulted to identify the practice and get guidance on its use. Table 3 outlines which practices of the visited methods were retained or rejected.

The junior requirements engineer had no particular previous knowledge in requirements engineering. He had, though, practical experience in implementing and configuring information systems that support business processes. Also, the use of patterns for building software solutions was known to him. That knowledge was not sufficient but may have influenced the identification of the practices that were considered by the project team.

Method	**Retained Practices**	**Rejected Practices**
Word-Processor Template	Text-based final requirements documentation	Text-based requirements analysis
Informal Stakeholder Interviews	Stakeholder meetings	Use of unstructured electronic documents
Beyer-Holtzblatt Contextual Design [Beyer, 97]	Layered activity models Work location models Artefact models	Responsibility and Communication models Value models
UML UseCase and Class Modelling [Booch, 98]	Class models Tool	UseCase models
Eriksson-Penker Business Modelling [Eriksson, 00]	Process models Information models Resource models State models Tailoring of UML Pattern-based language description	Goal/problem models Assembly line models Organization models Sequence models Collaboration models Business rules
Undocumented practices	GUI Prototypes Requirements elicitation, analysis, and validation cycles Colour-coding Analysis patterns	

Table 3: Retained and rejected practices in the finally adopted RE method.

From the team, the requirements engineer, the key architect, and the quality engineer drove the identification, tailoring, and application of the methods in a collaborative manner. Meetings were held to understand how a method would be applied, to agree on modelling language, and to plan requirements analysis work. The project leader acted as a facilitator. The other engineers were concerned of understanding targeted software technologies and of creating GUI prototypes.

The presented case describes a mechanism for extending a company's knowledge in the challenging field of requirements engineering. The project reached the limits of best practices that were defined in guidelines and policies. As a result, the team members pioneered approaches to requirements engineering that were new to the company. To retain the successful approach, the experience was presented to management and documented in the process database.

Parts of the learned have found application during the design and implementation phases of the described project. For this, the engineers that participated in the requirements engineering effort played an important role. For example, they drove the use of UML for defining, analyzing, and communicating the architecture of the software product.

To retain the learned practices at the level of the parent company, efforts were undertaken to spread the method to other development units. The requirements engineer got the opportunity to apply the successful method in other projects. It turned out, however, that the same success could not be repeated. Circumstances and goals were too different to get the same yield as in the presented case. No systematic analysis of these reasons has been made, though.

Also, in the described organization no other project had adopted the tailored method so far. Possible reasons include lacking awareness by engineers, different nature of products to be developed, different team compositions, and different circumstances during the requirements engineering phases. Also here, systematic analysis of the reasons is future research.

The difficulties of spreading the successful method, as well as the evolution of the team's learning process indicate that it may not be possible to define one commonly applicable requirements engineering method. As a consequence, companies need to be prepared for their teams going through similar learning experiences.

A team's willingness to search for the appropriate method on a trial-and-error basis has its limits. The team initially showed great openness to experiment by trying out a variety of methods. With the fifth trial, the team turned their method adoption strategy into method tailoring. Tailoring was successful because of the team's increasing understanding of RE concepts and tactics.

Without guidance, the selection of a possibly matching method may be a matter of chance. The consideration of methods was influenced by institutionalized company standards, thoughtful action, suggestions from experienced colleagues, common industry practices, and tool documentation. While some of these factors can be controlled by the management of a company, others cannot.

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].

An approach to ease and guide the learning process, thus eliminating the many trial-and-error cycles, is to coach junior requirements engineers. A coach can bring the understanding of how to achieve the goals of requirements engineering [El Emam, 95]. Still, to recommend and adequately tailor an appropriate RE approach, he needs to invest time for understanding the project's situation, goals, and problems to be solved. In the presented case, the coach could have provided the understanding of how 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 had to learn 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. The case is discussed from a learning software organization perspective by drawing lessons from the evolution of the team's requirements engineering expertise.

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. One question that should be addressed is how the findings of the study can be generalized, and to what kind of domains, companies, and projects. Another 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

[Beyer, 97] H. Beyer and K. Holtzblatt, *Contextual Design: Defining Customer-Centered Systems*: Morgan Kaufmann, 1997.

[Booch, 98] G. Booch, I. Jacobson, and J. Rumbaugh, *The Unified Modeling Language User Guide*: Addison-Wesley Professional, 1998.

[DeMarco, 79] T. DeMarco, *Structured Analysis and System Specification*: Prentice Hall PTR, 1979.

[El Emam, 95] K. El Emam and N. H. Madhavji, "Measuring the Success of Requirements Engineering Processes," in *2nd IEEE International Symposium on Requirements Engineering*. York, England: IEEE, 1995.

[Eriksson, 00] H.-E. Eriksson and M. Penker, *Business Modeling With UML: Business Patterns at Work*. New York: John Wiley & Sons, 2000.

[Fairbanks, 03] G. Fairbanks, "Why Can't They Create Architecture Models Like "Developer X"? An Experience Report," in *25th International Conference on Software Engineering (ICSE'03)*, 2003.

[Fitzgerald, 96] B. Fitzgerald, "Formalised Systems Development Methodologies: A Critical Perspective," *The Information Systems Journal*, vol. 6, pp. 3-23, 1996.

[Glinz, 00] M. Glinz, "Problems and Deficiencies of UML as a Requirements Specification Language," presented at 10th International Workshop on Software Specification and Design, San Diego, 2000.

[Henriksson, 03] A. Henriksson and H. Larsson, "A Definition of Round-Trip Engineering," University of Linköping, Sweden, 2003.

[Hergenhahn, 05] B. R. Hergenhahn and M. H. Olson, *An Introduction to Theories of Learning*, 7th ed: Prentice Hall, 2005.

[IEEE, 98] IEEE, "IEEE Recommended Practice for Software Requirements Specifications," vol. 830-1998, 1998 ed: IEEE, 1998.

[OMG, 03] OMG, "Unified Modeling Language Specification Version 1.5," Object Management Group, 2003.

[Ovaska, 05] P. Ovaska, M. Rossi, and K. Smolander, "Filtering, Negotiating and Shifting in the Understanding of Information System Requirements," *Scandinavian Journal of Information Systems*, vol. 17, pp. 31-66, 2005.

[Ross, 77] D. Ross, "Structured Analysis (SA): A Language for Communicating Ideas," *IEEE Transactions on Software Engineering*, vol. 3, pp. 16-34, 1977.

[Smolander, 90] K. Smolander, T. Veli-Pekka, and K. Lyytinen, "How to Combine Tools and Methods in Practice - a Field Study," in *Advanced Information Systems Engineering, Second Nordic Conference (CAiSE'90)*, vol. 436, *Lecture Notes in Computer Science*, B. Steinholtz and L. Bergman, Eds. Stockholm, Sweden: Springer, 1990, pp. 195-214.

[Sparks, 03] G. Sparks, "Enterprise Architect," Sparx Systems, 2003, pp. UML Modeling Tool.

[Yin, 03] R. K. Yin, *Case Study Research: Design and Methods*, 3rd ed. Thousand Oaks, CA, USA: SAGE Publications, 2003.