Introducing a Software Modeling Concept in a Medium-Sized Company

Klaus Schmid,
Ulrike Becker-Kornstaedt, Peter Knauber
Fraunhofer Institute for Experimental
Software Engineering (IESE), Sauerwiesen 6
D-67661 Kaiserslautern, Germany
{schmid, becker, knauber}@iese.fhg.de

Florian Bernauer
Markant Südwest Software
und Dienstleistungs GmbH
Sauerwiesen 6
D-67661 Kaiserslautern, Germany
bernauer@software.markant-sw.de

ABSTRACT
In this paper, we describe, using the Quality Improvement Paradigm (QIP), how an improvement project aimed at improving the modeling and documentation approach of a medium-sized company (MSuD) was conducted. We discuss the new modeling approach which may serve for other companies as a template for deriving their own adapted approach. Further, we illustrate our insights from this project that can help in future technology transfer projects. A major characteristic of this project was that it was embedded in a long-term consulting relationship.

Keywords
Software Documentation, Requirements Modeling, Product Families, Maintenance Support, Technology Transfer

1 INTRODUCTION

Documentation is key to any well-organized software development process. However, there are still documentation problems that abound in industrial practice. There are many reasons to this:

- Standard approaches do often not fit the real requirements of a company
- Documentation requirements may change over time and what was previously a well-adapted approach becomes inappropriate.
- The entrance barrier to changing the documentation approach is high, as it requires considerable redocumentation effort.
- A lot of knowledge about the specific documentation requirements is needed to devise an adapted approach. At the same time people working in the company do usually not have sufficient time and expertise to devise a new approach.

While many obstacles to switching to a new approach exist, a new and tailored documentation structure holds considerable promises, foremost among them reduced effort, improved maintainability, and improved quality.

In this paper, we will describe how an existing documentation approach, which was found over time to be no longer sufficient, could be successfully replaced by a new approach. The new approach was tailored to support the specific requirements of the organization. We will not report all details of the project but focus on those aspects which made the cooperation successful. Thus, we will focus on organizational issues, how the documentation approach was set up to mirror the specific characteristics of the organization, and how the introduction was organized. We will also present our overall lessons learned. The lessons learned will also reflect experiences derived from the long-term consulting context in which this particular improvement activity was embedded.

The rest of this paper is structured as follows: in the next section we will provide some background on the work done in this project. In particular, we will discuss the company environment in which the project was performed, and the context of the – still ongoing – consulting project. In Section 2.2 we will describe the core ideas of the Quality Improvement Paradigm (QIP). Then, in Section 3, the improvement project will be described in more detail using the QIP as a basis. In particular, we will discuss in more detail the various lessons learned from the project in Section 3.6. Finally, we will present our conclusions on introducing new modeling approaches into industrial settings.

2 BACKGROUND

2.1 Company Context

The Markant Südwest Software und Dienstleistungs GmbH (MSuD) is a subsidiary of the Markant Südwest AG, a company which operates several hundred supermarkets in South-West Germany. The subsidiary was founded with the major goal to develop a family of new merchandising information systems (MIS) for different branches of the company and to sell MIS-systems to outside companies. These MISs support a wide range of processes related to buying, storing, and selling supermarket goods.

Fraunhofer IESE provided consulting services to the Markant Südwest AG even before MSuD was founded in early 1996 and helped in selecting an appropriate setup for the subsidiary. When MSuD was founded, it was established next door to the Fraunhofer IESE. This was regarded as a key factor for a tight cooperation and continuous consulting.

When MSuD started in 1996, it was basically a three person company. In this crucial phase Fraunhofer IESE provided general consulting, in particular, on setting up the initial processes, proposing approaches for documentation and providing advice for selecting a suitable client-server architecture. A major task during this phase was also to provide training in the relevant technologies (especially, object-orientation) to the new employees.

Over the following years contracts for additional systems were acquired and MSuD grew rapidly, reaching close to 70 employees in 1999. About half of them work directly in software development. Obviously, due to this strong growth, the established structures had to be constantly reviewed and improved. The strong growth had also the effect that Fraunhofer IESE could focus on more specialized topics. Examples of these activities, that have been performed recently, are a detailed analysis of the existing architecture, consulting and improvement of the documentation process, and replacement of the existing system documentation approach by a tailored and more scalable approach. In this paper we will concentrate on the latter activity, as we believe it will provide the best insights into the project environment. However, we will also report on some lessons learned from earlier cooperation activities. Note, however, that this improvement project partially overlapped with other improvement projects we performed in this environment, as we will discuss later.

2.2 QIP

This section describes the principles of the Quality Improvement Paradigm (QIP), as we will use it for detailing the various activities of the improvement project described in Section 3. The Quality Improvement Paradigm (QIP) [3] is an iterative, goal-driven framework for the continuous improvement of software development. The QIP is a closed-loop process which includes steps for planning, executing, and evaluating improvements to software development environments, as well as for incorporating experience gained from improvement efforts into future development (cf. Figure 1). The six steps of the QIP are the following:

1. Characterize and analyze the current situation: Based on business characteristics the current project has to be classified. This classification allows to find those projects with similar characteristics and goals. This is necessary to find reusable experiences and objects, processes, and to find suitable projects to compare the current project against. A large variety of factors need to be taken into account to classify the current project. These factors include [2]:
   - people factors (e.g., the number of people or roles, involved, organizational issues, team experience)
   - process factors (e.g., life-cycle model, methods, techniques, tools, programming language used)
   - product factors (e.g., deliverables, system size, required qualities)
   - resource factors (e.g., development time, budget, calendar time)

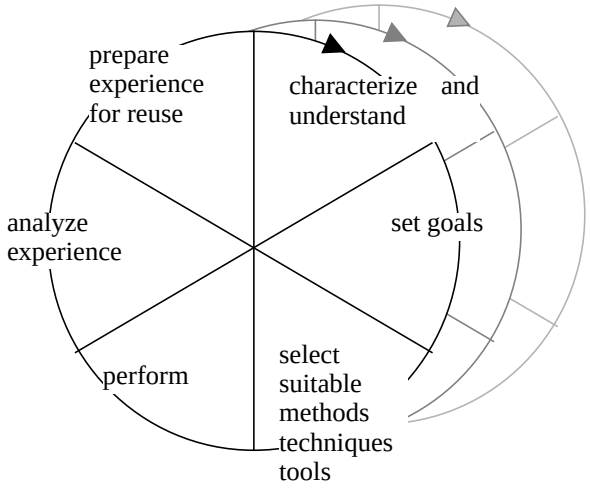

Figure 1: QIP cycle

2. Set sensible and reachable improvement goals. Based on the characterization of the current situation goals for the process have to be set. These goals can be qualitative or quantitative. The goals need to be stated in a way which allows to verify whether they have been achieved. A common way to do this is via a measurement program. Goals may be defined for any object, product or process, with respect to various quality models, from various viewpoints.
3. Select methods, techniques, and processes suitable for reaching a goal, prepare and plan project. Based on the characteristics stated in the first step, techniques, methods, and tools which are suited to reach the goal stated in the previous step are to be selected. Methods, techniques, and processes should be measurable and defined in terms of the goals they must satisfy. Often, the methods, techniques, and processes must be tailored to the specific characteristics of the organization and the goals.
4. Perform projects in a controlled way. The process is executed, integrating the new methods, techniques, or tools. In this stage, it is very important that the progress of the project is closely monitored and that deviations and problems are immediately dealt with. This step is typically supported by measurement programs. Often, in this step small pilot projects or experiments are performed, for instance, certain techniques are tested in such a pilot phase prior to their general introduction in the overall organization.
5. Analyze observations during the performance. Based upon the goals stated, the data collected on the execution has to be interpreted. This data will help to characterize and understand the project performance. For instance, such data may show during what phases of the project most errors are introduced, or whether the introduction of a certain technique helped to decrease the failure rate.
6. Prepare the experiences for reuse. In this step methods, techniques, or tools may be defined or refined to match

the experience gained in the project so that the experience can easily be reused in future projects. Data for instance may be used to guide or to predict future projects.

Often QIP cycles overlap as the figure shows: The experience gained in a QIP cycle may be used as characterization for a new cycle or new goals may be stated based on experience gained.

3 THE IMPROVEMENT PROJECT

In this section, we will discuss the subject matter of the improvement project and how it was performed in a step-by-step way. We use the QIP (cf. [2]) to present the project in a structured manner. Note, however, that in project-reality consecutive QIP steps may overlap. For example, during the *characterize and understand* phase usually some goals for the improvement will already become visible.

3.1 Characterize and Understand

As described in Section 2.1, the project described in this paper started at a point, where the company had already reached nearly its current size. Additionally, two systems had already been fielded, and further systems were under development. All of these systems can be regarded as part of a single product line as they overlapped considerably in functionality [6].

When the initiative for the improvement was started, documentation was already recognized as a major issue. At MSuD, two departments developed documentation for the software:

- *Analysis group.* This group is responsible for identifying and documenting the existing and aimed for business processes for the customer. Based on these processes it has to define the final requirements of the systems. Requirements are documented using a set of word-documents, where each document addresses a single work flow.
- *Design group.* This group is responsible for translating the final requirements documents into the software design. The design is described using GUI descriptions, control flows, and UML-models.

Thus, at least three levels of documentation can be identified:

- Business processes
- Software requirements
- Software design & control-flow documents

Additionally, while a basic documentation process existed, it was not consistently performed, rather its execution varied from individual to individual. These problems were aggravated by the fact that several variants of the system were developed and maintained in parallel.

Although other problems existed in the work-organization – especially those regarding coordination between departments and consistent process execution – it was decided to focus on core problems relating directly to the documentation as these were the most urgent ones. More precisely the following problems were identified as crucial to the project:

- It was hard to identify which documents were relevant to a certain aspect of the systems.
- Information in the documents was hard to trace in the documentation. In particular, it was difficult to trace documents from one department to the next.
- No consistent approach to developing documentation existed (no guidelines were available). Thus, the documents varied considerably wrt. the style and detailedness of the description.
- It was often unclear what documentation was related to which of the systems, that were under development.
- It is hard to understand how documentation captured in various forms (business processes, control-flows, text-documents) complements each other.

However, these problems are not unique to the environment under study, but — according to our experience — can be found in many software development organizations.

3.2 Set Goals

Based on the understanding of the problems outlined in the first step, goals were identified. Given the problems above and the decision to focus on core documentation aspects, the definition of the goals was rather straightforward. We came up with the following list of requirements for a new documentation approach:

- Consistent tool support, integrating the various types of documentation
- Ability to integrate documentation that has been captured using other tools (e.g., Word)
- Usage of an easy-to-understand graphical notation
- Adaptation to the notational needs of the various levels of documentation
- Support for integrated modeling of the various system under development (variants)
- Support for consistency checking within and across diagrams
- Support in migrating to a more disciplined documentation approach
- Support for communication with customers: The new documentation structure had to integrate business process models at the customer's abstraction level using a notation familiar to the customers

These requirements had to be satisfied without disturbing ongoing project work too much. This implied that a low risk approach had to be used that only consumed rather restricted resources. Furthermore, due to resource constraints at MSuD, the more of the transition effort could be delegated to the consulting partner, the better.

Consequently, one can state the goals of the project as achieving:

- high **usability of the tool environment** (in terms of availability, response time, and ease of use of the tool)
- high **accessibility of information** (in terms of traceability among and within levels and effort required to find information relevant to a certain task)

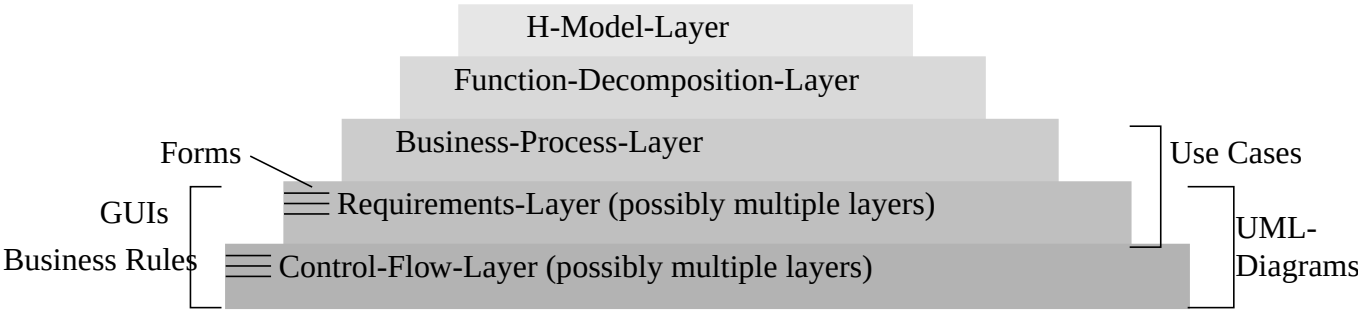

Figure 2: The documentation layers

- **ease of documenting** (in terms of ease of expressing information using the notation and of the ability of the notation to express the required information)
- improved **understandability of documentation**. This is relevant to external customers as well as for communication across groups.

Obviously measures for these goals are subjective and thus not easy to collect. This was one of the reasons why no measurement program was established at this point. Another reason was that due to the close cooperation feedback on significant problems and quick reaction on these issues was ensured.

3.3 Select Suitable Methods, Techniques, and Tools

The Solution Approach

Based on the resource constraints it was immediately visible that the documentation effort had to be based on existing, field-proven tools. However, no existing tool-environment covered the wide range of documentation needs the situation required. Thus, simply buying another tool wouldn't do the trick, but the tool had at least to be adapted to the specific company situation. Therefore, a solution was chosen that consisted of three main aspects:

1. Document organization
2. Modeling notation
3. Tool support

The first point was driven by the goal of providing higher accessibility of information. This directly led to the need for a basic model integrating the different types of documentation. For these purposes a multi-level approach was developed. The levels include the three levels of abstraction mentioned in Section 3.1, i.e., Business Process Layer, Requirements Layer, and Control Flow Layer. In addition, two upper layers, H-Model Layer, and Function Decomposition Layer, were added. The upper two layers which were mostly aimed at navigation were expected to be slight variations of a standard model for merchandising systems, the "H-Model" [1].

A model depicting all five layers is presented in Figure 2. This figure also shows how other work products that are produced in software development (use cases, GUI-descriptions, forms, etc.) are related to and integrated into the different layers. In particular, this layered model was

Figure 3: Basic notation for business processes (eEPK)

expected to help in organizing the various documents and to improve the traceability among the documents.

In order to 'live' this model, adequate modeling notations were needed. As it was obvious that no single modeling notation would be sufficient to capture all relevant aspects, it was decided to center the description around the notion of flow-modeling. However, different types of flows would be needed on the various levels. So, we finally settled on the idea of using a single flow-modeling tool with adapted notations for the different levels.

We decided to use the ARIS tool set [4], as it is a well-known and widely used business process modeling tool. In addition,

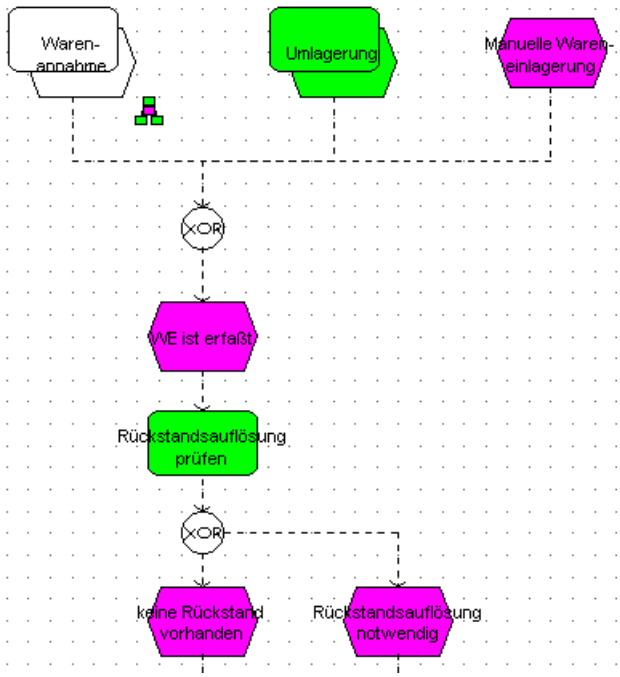

Figure 4: An example business process

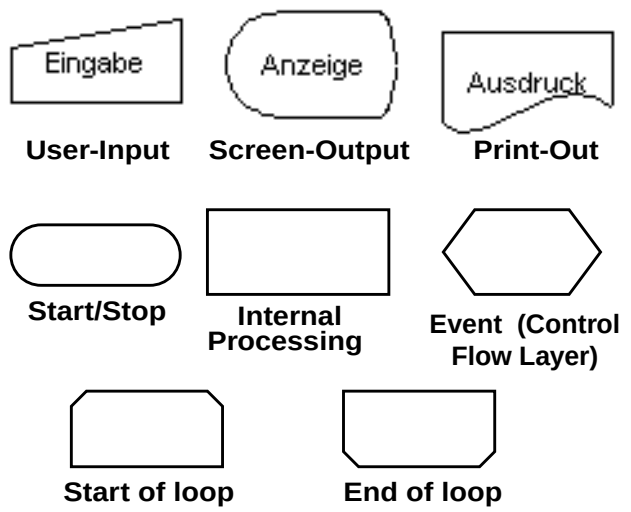

Figure 5: Symbols added to the basic notation

it would allow us to support the Business Process Layer out-of-the-box, in a way that directly contributed to our goal of providing a notation that would also be easy to understand for our customers. Figure 3 gives an overview of the notation used by ARIS for business process modeling. A simple example of a business processes described using the ARIS-notation is shown in Figure 4. The additional notational elements that were used for the requirements and control-flow layer can be seen in Figure 5.

A further benefit of using ARIS and its standard notation is that it is widely known among the customers of MSuD. Thus, documentation using ARIS eases communication with customers.

Additionally, ARIS supports the customization of the modeling notation. Thus, we could augment the notation with symbols to represent the other layers. The tool also supports linking other documents to diagrams, so that, e.g., text or HTML documents could be referenced as needed. We do not want to go here in the details of the modeling notation that was developed, as it would go beyond the scope of this paper. However, there is one aspect of the modeling notation we want to highlight here: All systems that were expected to be developed share a lot of commonalities. Thus, it was a requirement that the modeling notation should be able to represent the similarities among the systems. This was done both to reduce the amount of duplication among documents and thus to simplify the maintenance of the documents and to identify opportunities for reuse for future systems.

In this context, experience from domain engineering work and in particular the knowledge on domain analysis notations that exists at Fraunhofer IESE could be successfully used [5, 6, 7].

The following requirements needed to be fulfilled by an approach for the modeling of variants:

- The approach needs to be simple (i.e., a small number of notational symbols)
- It must be possible to apply uniformly on all levels (from business processes to control flows)
- It must be possible to represent the variations in terms of differences in the domain as opposed to differences among the systems

The last requirement may sound strange, however, we regard it actually as crucial in building a successful reuse program. This idea of explicitly representing system variation as domain decisions is also core to PuLSE-CDA, the domain analysis component of PuLSE [6], the reuse approach developed at IESE. A domain decision can be represented as follows:

Name: Geschäftsart
Description: defines whether goods are delivered directly from the provider to the client (brokerage) or whether goods are first delivered to a central storage area.
Values:
zentral — goods are delivered to a central storage area
strecke— goods are delivered directly to a client
Selection: 1–2:2
Constraints: —
Systems:
A: {zentral}
B: {zentral, strecke}

The name provides a unique identifier (here Geschäftsart = business type), the description further explains the decision. The values clause defines values for the decision. The selection than identifies what sets of values are correct values for the decision (i.e., here {strecke, lager} is a correct result.

Constraints identifies restrictions to the overall set of values based on the values for other decisions. Systems finally identifies the values the decision has for different systems. A specific system can thus be described by a list of decisions that are taken.

The decision description by itself does not show how the individual variants differ. For this the concept of variation needed to be introduced into the various notations described above. As systematic variation was found to be a very difficult and new concept to the personnel, we decided to develop a simple notation that could be coherently used on the different layers.

We were able to come up with only three different types of decisions that allowed to easily represent the variations that needed to be described. These are called:

- Unique variant decision
- General variant decision
- Optional variant decision

An overview of these decisions (and the variant discriminator) is shown in Figure 6.

The *unique variant decision* is used in situations were a decision can be made during development time among several paths such that for each system exactly one path is chosen.Thus, no more than one path is relevant, no matter

what the value of the domain decision is. Thus, the instantiated diagram for this model can be constructed by removing the decision symbol, the variant discriminator, and all non-selected paths. An example of a unique variant decision is shown in Figure 7. This example shows a business process that will vary depending on the type of delivery ('Geschäftsart') that the business using the MIS supports. This can be either "Zentral" (i.e., the customer orders goods straight from the inventory) or "Strecke" (i.e., goods will be delivered to the customer via carrier directly from the producer).

A bit more difficult is the *general variant decision*. This is actually a generalization of the unique variant decision. In this case more than one path might be relevant. If exactly one path is chosen, then it behaves like the unique variant decision. Otherwise, it is converted into a run-time decision that may select among the various paths compatible with the domain decision choice.

The *optional variant decision* is used in situation where a certain path only exists for some variants. This is especially useful for describing user interfaces on the requirements level that contain a certain button (and thus allow a certain action) only for specific variants. Depending on the value of the domain decision the path is contained in the system instance or not. Further, if it is present it will be selected, if the corresponding runtime choice is made.

This rather restricted set of notational elements has been found sufficient to describe all relevant situations.

Technology Transfer Approach

It is a well-known fact that the success of a technology transfer project does not only depend on the quality of the technical solution. Rather, the transfer of these technical ideas to the industrial environment needs to be adequately managed and supported in order to be permanently used in the target organization. Consequently, we came up with what could be called an *organizational solution* in addition to the technical solution that was described above. This organizational solution consisted of four steps:

- Up-front training of the employees
- Rapid transition to the new tool environment
- Ongoing consulting
- Guidelines / checklists

A key decision we made was that the tool should completely replace the tool that was previously in use for control-flow modeling. Further, this should be done without any transition period, thus transition had to happen from one day to the next. This required that by the transition date the new documentation structure had to be in place and all existing

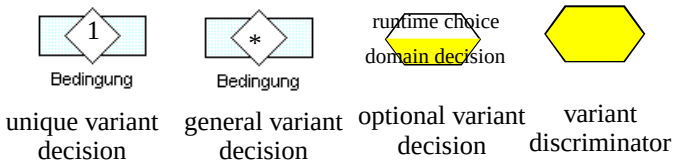

Figure 6: Symbols added for variants

control flows had to be accessible in the new environment, so that modeling could continue without any disruption. Further, people had to be already trained at that point in time wrt. the new tool environment, the modeling approach, and the new documentation structure.

As all of this had to happen without impeding ongoing development work at MSuD, Fraunhofer IESE as the consulting partner took care of this. For training of the employees a one-day training course was developed that included theoretical and practical training with the tool environment, the new document structure, the modeling notation (especially differences to the one previously used), and in particular on the way chosen for representing variants. As an one-day course can obviously not be very detailed, it was clear that further support was needed. It was decided to have additional modeling guidelines and checklists. These were made available to MSuD staff in the form of small booklets, provided by IESE members. Additionally, people from IESE would be available for further questions and the clarification of any upcoming issues.

While the first sub-project focused on the new documentation structure, a second sub-project, performed in parallel, focused on the introduction of new practices. In the second sub-project the problem of having different ways in which documentation was performed was addressed by supporting the development of other analysis and design work-products by templates and checklists. Originally, this restructuring did not directly aim at the core documentation approach, but centered around the improvement of project manageability. However, as the groups of people in the two projects did overlap, the opportunity was seized and the new templates and check-lists for this improvement project were

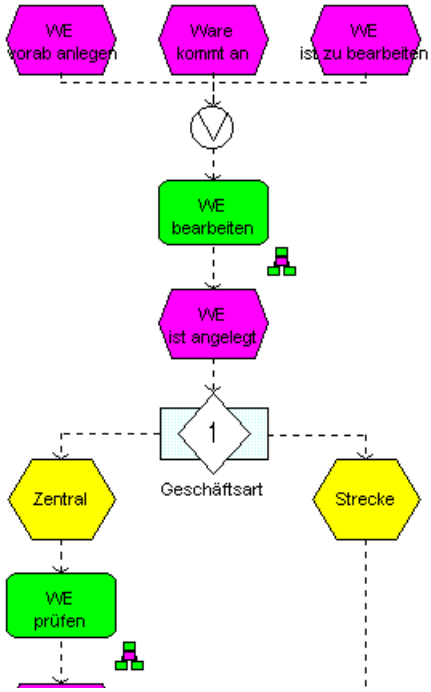

Figure 7: Example for the description of variants

developed in a way as to ensure a more repeatable approach to analysis and design documentation.

3.4 Perform

In performing the shift to the new documentation approach three phases could be distinguished:

- Preparation phase,
- Introduction phase, and
- Application phase.

During the *preparation phase*, the prerequisites (i.e., training, check-lists) were developed as discussed in the preceding section. The new core document structure was set up by people from Fraunhofer IESE. Student workers and some non-software development people of MSuD helped in transferring existing documentation from the previous tool environment to the new one. This process was organized and supervised by people from Fraunhofer IESE. Additionally, they performed reviews of the transferred material. As this rapidly transferred material still adhered to the old guidelines it still exhibited many of the old problems. Consequently, it was expected that this material would have to be reworked later on, but the aim in the rapid transfer was to have completely up-to-date documentation available in a single tool environment. During this phase already first problems with the tool could be discovered and recorded so that training could already take them into account.

Also during this time the introduction courses were given, so that people would be adequately trained in time for the transition. Due to restrictions on personnel availability, these courses had to be repeated several times to ensure that everybody had received the adequate training.

At a certain, generally agreed date all documentation switched to the new tool-environment. With this the true *introduction phase* began. Pretty soon people discovered that they felt unhappy with some aspects of modeling, especially on the requirements level. Thus, adaptations had to be incorporated into the documentation approach. During the introduction phase people from Fraunhofer IESE were regularly on-site and provided hotline services. As both sites are only one minute walking distance apart, in case of problems somebody could be immediately on-site in order to solve any problems.

While one of the reasons for choosing the tool environment was to have a mature tool base, still problems were discovered during the introduction phase and ways to deal with them had to be found. We found it important that tool knowledge would be built up inside MSuD. Thus, somebody from MSuD was selected to become step by step an expert for the tool environment.

Often, when performing a QIP-project the execution is monitored with the help of a measurement program [2]. In our case we did not do this in an explicit manner. This had various reasons. One of them being that the regular site-visits of IESE employees, and the phone calls to our ‘hot-line’ allowed to constantly poll for problems and difficulties with the new approach. This proved to be a working way for detecting major problems during the introduction period.

After some time the situation stabilized, requests for changes and the identification of problems abated and finally ceased. This we regard as the *application phase*. It started a few months ago. By now, the modeling approach (including checklists, templates, and guidelines) is in regular use. After the situation stabilized, the technology experts could phase out of the project and turn to other projects. New people could be brought in to take care of further sub-projects within the overall consulting relationship between IESE and MSuD.

3.5 Analyze Experience

As described in the previous section, feedback could be gathered during the execution of the improvement project mostly via the frequent interaction of IESE and MSuD personnel. However, while this helps to identify pressing problems, it is insufficient for getting an overall picture of the results of the improvement project (especially the positive parts). In the described project we had the additional benefit that this was only one project within an ongoing cooperation. One of the next steps within this cooperation was to perform assessments of the software development process for several reasons:

- First, results of the assessments would be used as a baseline for the next QIP-cycle, were the emphasis would be on process improvement.
- Second, it was used to assess in detail the interaction of the analysis group with the design and the quality assurance groups.
- Third, it was used to introduce new people who would be involved in performing the next QIP-cycle.

This was a good possibility to get additional insight into the detailed results of the described improvement project, and required very little additional effort.

The assessment was performed in the form of structured interviews. The questions were taken from templates that are commonly used by Fraunhofer IESE when assessing small and medium-sized companies, where development processes are not too well (or at least not explicitly) defined and the people interviewed have very different background and qualification. We had two rounds of interview sessions:

- In the first iteration we interviewed the head of software development and the group leaders of that department (groups correspond to the development phases: analysis, design, implementation, quality assurance) about the overall development process. An already existing high-level process description was used as basis for the discussion. This description was refined and (slightly) changed as a result of the interviews.
- In a second step we interviewed the members of the analysis group about their detailed process: when and which documents are produced, reviewed, and forwarded, and read by whom. Of special interest here were the interactions of the group members with members of other groups, especially from design and quality assurance.

Through the assessments and the previous personal interaction the following picture arose of the results of the overall improvement approach:

Despite the fact that a tool was used as basis for the documentation approach, that is in active use in various industrial environments, still an unexpectedly large number of problems were encountered. For example, the tool-set contains functionality to publish the documentation on the Intra-Net. This was supposed to be the interface for those development groups that did not have to actively model any information (implementation and quality assurance groups). However, it soon became visible that the number of diagrams needed for software documentation caused severe performance problems both on the client and the server side. Thus, a work-around had to be found by sub-dividing the documents into groups. This would have been very hard to detect prior to the tool introduction, as the Intra-Net interface was reported to work well even with large numbers of documents and the generation of sufficiently large number of documents for testing the performance would have caused a prohibitive amount of effort.

In daily use also some minor problems with using the tool surfaced (e.g., the length of entries in comment box was limited). While these restrictions had only very limited impact, they still caused severe annoyance in the early introduction phase. Further, some developers found the user interface of the tool somewhat clumsy. But opinions diverged here, and after some time people got used to the user interface. While it will be always the case that people's opinions will diverge on the subject of tool interfaces, here early exposure on tool usage and stronger involvement of the development team during the tool selection process could have helped.

Also the notation caused minor problems. The exact semantics of the notation caused some misunderstandings in the first place, but this could be sorted out after a while. It was found that these were notational elements that had received less attention during the training, as they are of only restricted relevance. This shows clearly the need for more extensive training, with detailed coverage of all notational elements.

A problem which was unexpectedly big, was the fact that people tended to avoid to explicitly model variants. This was particularly disappointing as considerable effort had gone into the development of the variant notation. Our analysis showed several reasons for it:

- The concept of modeling various variants within a single (domain) model was found to be extremely hard to get across. This was especially the case for people who had to focus on the new requirements for a single system, as they are just not used to think in terms of multiple systems.
- The notation gave the impression to be rather formally defined, as some care had been given to its ability to cover all required variant modeling capability, but this actually drove some people off.

Only some minor issues were found wrt. the document structure. Here, especially the strong focus on hierarchical relationships among diagrams (see levels in Figure 2) was found to be too restrictive. Thus some relaxation of the level definitions had to be performed.

However, despite the aforementioned problems, the overall transition was very smooth and no problems were found that caused severe difficulties for daily work. Thus one of the major goals of the effort was clearly achieved, i.e., to transition to a new tool environment without causing major difficulties in daily software development. An additional benefit of this effort was that already during the initial transfer (i.e., not only during the later redocumentation effort) some improvement of the existing documentation could happen (e.g., making the description and layout more coherent).

In total, one can summarize by stating that the following results have been achieved:

- The start-up phase was brief and documentation did actually improve (e.g., more coherent usage of notation, more comprehensive documentation).
- Usability of the tool environment and accessibility of the information was lower than originally expected, due to the aforementioned problems, but especially accessibility was higher than the using the previous approach.
- Ease of documentation has – after some adaptations – reached the hoped for level, although with the restriction that still resistance to the usage of variant modeling exists.
- Communication with the customers is facilitated. The new documentation structure allows to provide the right level of detail for negotiations with customers. In addition, customers are familiar with the notation used.

3.6 Prepare Experience for Reuse

What could be reused in the next project? Immediately into mind come the following:

- IESE members acquired detailed tool knowledge (ARIS) which may come in useful in other projects.
- IESE has experience with a customized tool and notation. This experience will be beneficial in future projects.
- We have developed training material for various levels of modeling.

However, all this is very context-specific and thus the chances for reuse are very small. Therefore, we consider the major results to be the experience, the lessons learned. We will concentrate in this section on them. We will first discuss the lessons learned directly from the specific project described here, then we will also discuss more general lessons learned, that are actually due to previous improvement projects within this cooperation. These lessons have already been validated in this project.

Lesson learned: Successful introduction of new practices is only possible with the active cooperation of the people in the target organization. It is not enough if a consulting company only shows how things are to be done in the future. To make sure that new practices will be performed, it is necessary, that these practices are tailored to the organization. For this task, the active cooperation of people in the target

organization is crucial. They have to adapt the new practices as their own and they have to help adapt these practices. In such situations it is important that all of them know how important it is to participate in order to get a 'tailored' solution. Consequently, people within the target organization have to be introduced early on in the improvement effort. Thus, finding a champion for the change within the team is particularly important for establishing the new practices.

Lesson learned: Organize a detailed evaluation of the tools. This should include already considerable trial usage of the tool applied to examples of realistic size and structure. In this phase also the future users of the tool should be strongly involved.

Lesson learned: Training is extremely important to get right, right from the start. Thus, a one-day training course is just too little for a change of this magnitude. Training should also emphasize those aspects that will not be so often used. They are still too important to be misunderstood.

Lesson learned: Domain analysis concepts are very hard for people to understand. This is particularly true, if people focus in their daily work on a per-project basis. Making the entrance barrier low, e.g., by using informal modeling or by using a tool that also supports domain instantiation, so that people directly see on a per-system basis what the effect of model changes will be might help (cf. [5]).

Lesson learned: Having comprehensive examples for the future documentation right from the start helps a lot, both in terms of validating the approach, as well as in helping people to get a full understanding of the concepts and their implementation in daily work.

Lesson learned: Detailed guidance and support in the early introduction stages (right after training) is very important. In the future, an approach of doing their work with them should be tried in order to help the personnel start with a new modeling approach. Waiting for them to ask for assistance was sometimes too late.

Besides the above lessons learned that were directly derived from the specific improvement effort described in this paper, there are also some lessons learned, that have been derived from earlier phases of the improvement project and have been ascertained by the project described here.

Lesson learned: A long-term relationship simplifies cooperation. It builds trust and understanding, and many of the improvement activities performed would not have been possible without the special relationship that had evolved.

Lesson learned: The fact that both organizations are in direct neighborhood of each other facilitates communication. Many problems could be solved in direct face-to-face talks very easily, whereas it would often have been difficult to settle those problems via phone, fax, or letter.

Lesson learned: We found that working on-site helps little. This applies especially, if the consultant gets too strongly involved in everyday activities, such as coding or documentation. There are two reasons for this observation: First, a consultant should be a coach, not a player on the field. Once consultants – whose task it is to give advice in critical situations – focus too strongly on everyday work, they will loose too much of their objectivity to find appropriate solutions for the problems of the company. Second, we found that just being there does not give the necessary understanding: Although we worked on-site a lot in earlier phases of the project, we remained 'outsiders' to the organization. Thus, while being in close contact with the organization helps a lot, it does not automatically mean that you will become a 'member' of the organization, it may even aggravate the situation. Since we had been on-site, it was generally assumed that people from Fraunhofer IESE were aware of how things were done in the organization. However, this was often not the case as IESE members were not introduced to general practices in the same way as MSuD personnel.

Lesson learned: Working on site does not necessarily give a picture of the practices in an organization. We had suspected that working on site would yield a good understanding of the general practices in the organization. However, it turned out the understanding gained was restricted to the tasks we ourselves had performed on site. Especially the broader context of these tasks was missing. For a good overall picture of the practices performed, focused interviews would have been much more efficient. From our experience this type of participant observation is only helpful to supplement other techniques to capture process information, e.g., it could be used to get a thorough understanding of detailed activities and tasks when an overall understanding of the process is already there.

Lesson learned: While companies like to have a single point of contact over several years, rotation needs to occur regularly and in a planned manner. So that the right people for a problem area were introduced in time – and taken out, when their time was over. This is especially important for a long-term consulting project, where over time a wide range of different problems are addressed and different techniques are introduced. Even though it is helpful to have people involved who are familiar with the environment, it seems to be better to have people involved in the project who are not familiar with the environment but have a particular strong background in the currently needed area of software development practices.

Lesson learned: New people that shall be introduced as consultants in an environment often need to be trained in some way on the environment specifics. The level of this training will always depend on the specific type of improvement effort they are supposed to help in. For example, somebody coming in for an assessment will not need a lot of up-front training, while somebody aiming at helping in improving the software design approach in detail will need considerable exposure to the system, the existing implementation, etc.

4 CONCLUSION

In this paper, we described how a documentation improvement project was conducted in a medium-sized company. The project amounted to a complete replacement of the modeling and documentation approach. It also

included changes to the modeling notation and the tool platform. With respect to the modeling notation perhaps the strongest shift happened as a new notation had to be developed that integrated aspects of business process modeling, requirements modeling, and control flow modeling in a domain-specific manner for multiple variants. Overall, the major results aimed for in the transition were achieved. However, some problems were encountered that lead to the lessons learned we described in this paper. We believe, that with the backing of these experiences we will be in a much better position to address future technology transfer projects. Throughout the project we benefited from the embedding of the improvement effort it in a long-term cooperation on many levels.

ACKNOWLEDGEMENTS

We would like to thank all the people who contributed at certain phases to this project. Special thanks go to Oliver Flege, Stefan Dittrich, Jürgen Kämmerer, Werner Schirp, Oliver Vering, and Lars Ehlers.

REFERENCES

1. Jörg Becker and Reinhard Schütte. *Handelsinformationssysteme*. Verlag Moderne Industrie, 1996.
2. Victor R. Basili. *The Experience Factory and its relationship to other improvement paradigms*. Fourth European Software Engineering Conference, Ian Sommerville and Manfred Paul (eds.), Lecture Notes on Computer Science Nr. 717, pp. 68–83. Springer Verlag, 1993.
3. Victor R. Basili, Michael K. Daskalantonakis, and Robert H. Yacobellis. *Technology transfer at Motorola*. IEEE Software, Vol. 11, No. 2, pp. 70–76, 1994.
4. Information on the ARIS-toolset can be found at http://www.ids-scheer.de/produkte.htm.
5. J. Bayer, D. Muthig, and T. Widen. *Customizable Domain Analysis.* First International Symposium on Generative and Component-Based Software Engineering (GCSE ’99), Erfurt, Germany, 1999.
6. J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, and J.-M. DeBaud. *PuLSE: A methodology to develop software product lines*, in Proceedings of Symposium on Software Reusability’99 (SSR’99), pp. 122–131, May 1999.
7. J.-M. DeBaud and K. Schmid. *A Practical Comparison of Major Domain Analysis Approaches — Towards a Customizable Domain Analysis Framework.* The 10th International Conference on Software Engineering & Knowledge Engineering, SEKE ‘98, pp. 128–131, 1998.