Agile Software Engineering: A New System for an Expanding Business Model at SCHUFA

Rainer Burkhardt[1] and Volker Gruhn[2]

[1] SCHUFA Holding AG, Wiesbaden, Germany
[2] University of Leipzig, Germany
Rainer@Burkhardt.com
Gruhn@ebus.informatik.uni-leipzig.de

Abstract. A software development unit called to renew or replace an existing corporate system may face some special problems in an established company with a lot of well-trained behavioral patterns and thought structures fitting to the legacy system only. The challenge is not just to be on a journey and reach the destination with the development team but keeping accompanied by the stakeholders during the travel. In this article we describe how Agile Software Engineering can be introduced to an in-house development structure of a company. We describe how agile process elements and model driven approaches can be combined in order to achieve a light weight, flexible and incremental software engineering process. We also show the resulting organizational structure of a development department and have a closer look to the management actions that must be taken to introduce agility to the internal team and the stakeholders all over the company.

1 Introduction

The German company SCHUFA has a 75 years history. Reorganized effective 2002 from a set of regional companies to an all-national and centralized AG with lots of shareholders – mainly banking corporations – it is still privately held. In an expanding market environment SCHUFA is a well known key player and market leader in the credit bureau business, has a tremendous pool of data on natural persons in Germany and is competing with international positioned companies like Experian, InFoScore and others.

Nowadays, not a small set of companies face the situation, that a working IT system is processing some established business cases generating good profit. As technology and markets change, the need rises to rethink and rewrite the business model of companies in small steps. As crucial future capabilities of the IT system will be for the company's business success in the future as crucial the maintenance of the existing business presently is.

SCHUFA is one of these companies and addressed the importance of developing new markets while maintaining existing customers and shareholders by creating an in-house IT including a software development department.

The two major challenges for the SCHUFA IT have been and still are the ongoing developing and the going live of releases of the new system while maintaining the still productive and doing well legacy system – at least the not yet replaced parts. Thus, the legacy system business is focused on production demands and requirement while the new system business spans all project activities from early domain requirements to production demands as balancing optimizations and monitoring requirements from the production and operations teams.

Problems or challenges are accompanied by a lot of opportunities in our case. From the point of view of a software developer the most exciting one is to build and manage a new team for developing the new system. That newly hired team can feel relatively free of burdens that the legacy system and the long time employed staff have to carry. And these guys can use latest news and experiences from technology and science. This is not a statement like "throw away what you've learned – here is the new stuff"; because that is not what we think. Development in iterations and increments, building releases and modeling already have been a good choice earlier [2]. Now we had the chance to add lots of agility and omitted some predefined structure of processes and artifacts.

Agility is a really natural thing – at least for some of us. However, applying agile modeling and extreme programming in software development – as far as we have learned – is not a matter of laissez-faire. There is still an employment opportunity and need for managers in agile biospheres. Thus, an agile software engineering process contains activities on maintaining and selling as well as on controlling and educating the agile practices. The actions to be taken are headed toward the development team and the stakeholders. Concerning matters are the system, the architecture, and technological aspects.

The following chapter addresses the basic aspects of our approach. In the third chapter we refine the process on relationships and roles of stakeholders while we focus on the chosen organizational options as an environment in chapter four. The following two chapters explain how we applied our approach inside the development department of SCHUFA and how we lived the process in every day's business. Finally we summarize the lessons learned and left for future work.

2 The SEP Approach at SCHUFA

Software engineering processes (SEP) deal with artifacts and processes and try to give a handsome guideline to a development unit. A SEP is valuable if it is applied by the addressed unit, raises the quality of the produced software releases, helps to hit the goals of the projects and accelerates the velocity of the software production.

Introducing heavy weighted processes for about a decade, we learned that the first mentioned topic from above is crucial: the application of the process in a day by day business and by every member of the team.

We find it easy to trust a decision that already had proven to be the right one. Applying best practices is easy to argue because it uses knowledge from lessons learned earlier without paying the price again. Arranging the development in short cycles is one example: we can guarantee to have evidence available earlier and are able to

evolutionary improve the development's processes. Thus, we take benefits from small steps and short cycles.

Combinations of iterative and incremental discipline-based development processes also use that nowadays (see [4]). So we decided to follow an evolutionary approach (see [5]).

Thus, light weight processes enable both: measurements to be taken soon and organizational effects to be effective early.

System

In migration from a legacy 3270 world to a via web accessible system an XML gateway addresses one of the most important objectives: get all SCHUFA customers shifted to the new system in a delimited time frame. Providing an extensible language and a 24x7 available service platform for contract partners to access today's and future's services from SCHUFA hands the technical means to the management and the sales force to connect new and existing customers to the SCHUFA system. All customers that are live at the XML Gateway my not take care if the demanded services are processed by the legacy system or some replacing component of the new system at SCHUFA.

Written in Java and already following the new architectural concepts that also are basics for the new system the XML gateway as the departments first productive service of the new system helped us train and improve the development and deployment processes and also helped us to find, fix and establish the new organizational structure. Via our XML gateway the complete set of services of the legacy system had been brought to the contract partner. However, our main internal interest in providing such an interface was to lay out a migration path for the existing (~5.000) and new (many) customers. We consciously left some work to do for others (outside the development) to execute the plan of getting all customers exclusively to that gate.

Our next steps to replace the Legacy System "just" had to replace the use cases of the acting production system by new solutions realized by components of the new system.

Architecture

Joining the company we discovered a remarkable technical oriented thinking from the IT division to the legal department and the sales division. Unfortunately the used technical terms mainly were bound to the long time existing legacy system implementation not to the domain. To learn the terminology of a company is one thing to make it accessible to new employees – and the SCHUFA hired a lot of new employees in that time – is sometimes completely different.

We anticipated that circumstances with an architectural metaphor. Since blade servers actually entered the mind of IT people we chose a similar design in three levels where the blades to be inserted are the software components (figure 1).

The case is symbolizing the calling-type framework of the new system which is based on J2EE using *Container Managed Persistence* for *Entity Beans* and *Stateless Session Beans* for the activities in a workflow written in UML's activity diagram notation and executed by the framework's workflow engine.

The lower level is capable to hold *Access Components*. These components encapsulate the access to subsequent systems, mainly the legacy system, the search engine, the decision support system and via the database management system to the attached databases.

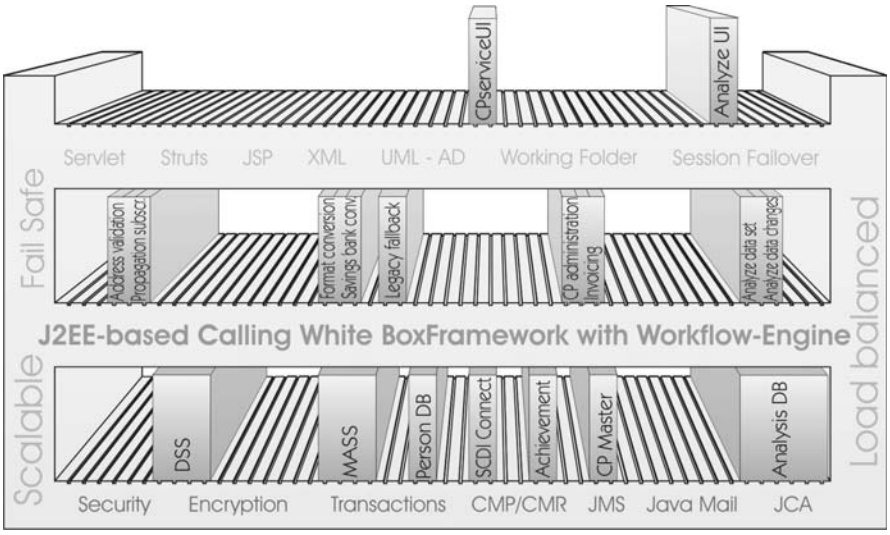

Figure 1: Architecture of the New System

The mid level is the model-level of the architecture. So called *Activity Services* implement business components. We stereotyped these components to be *Organizational Components*, *Product Components* or *Workplace Components*. These components take remarkable advantage of detailed and well refined models. The static and dynamic structure is a crucial and sometimes immanent scope of rethinking, refactoring and rearchitecturing. As a result, sometimes the framework has to be extended. In all other cases the model of the application system is modified, altered or just enriched. We mainly used UML's *Class*, *Sequence* and *Component* diagrams (for dependencies). Sometimes in discussions we experienced *Deployment* diagrams to be helpful, especially to show which items we talk about are components residing inside the containers and which are located elsewhere referring the distinct machines.

The upper level is able to hold interface components. One is the above mentioned XML gateway. Most others are user interface components.

Architecture is a term that often appears as a homonym. The reader already noticed that the above given description is concentrating on software aspects that we call component architecture inside the department, following [2].

Development Team

Mixing expectations and experiences gave us a clear understanding of what the core tasks of the development team should be. We had to develop new products for the new system and to redevelop existing ones to stepwise replace the products from the legacy system. We had to maintain and improve the architecture and to do the con-

cerning use-case spreading work. And we had to build new workplaces for internal employees.

Our work can be planned and the tasks to do can be ordered. However, there are some dependencies that rise the first time you go into the deep stuff – means: in modeling or programming sessions. Thus, *organizational* and *product components* sometimes turn out to use existing systems what pushes the appropriate *access component's* feature on top of the task stack. *Product* and *workplace components* sometimes turn out to force extensions to *interface components*, machine or user, causing the team to push the task to build a new version of that particular component to the top.

All these tasks not only cause effort in programming. Also involved is modeling, web design, database design, architecture, system technicians and production's concerns, quality aspects, requirements and of course all further communication to the stakeholders and management including reporting, best proved to be done by experienced project managers.

3 Addressing the Stakeholders

Stakeholders in XP terms are usually named just customers. While we focus on a corporate in-house development with SCHUFA here we are able to make some distinctions appropriate to that special scope. Figure 2 shows a development-centric view to stakeholders inside and outside the IT division.

None of our stakeholders forced us to introduce agility. Thus, we – as the driving force – had to "sell" agility to all stakeholders. In the following paragraphs we summarize our related activities.

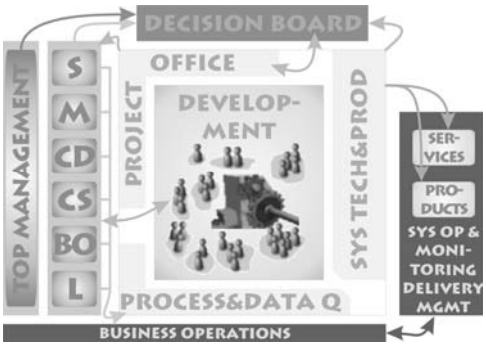

Figure 2: Stakeholders inside and outside the IT division (frame)

Stakeholder: IT Management

The management in the IT division consists of the division director and four managers, three leading a department, one leading a supplementary unit attached to the division. While we take the point of view of the development here, the *System Technique and System Production* (STSP) yet is a stakeholder. Holding the stakes of the

company for data center concerns this department contracts the hosting partner and controls all service level agreement matters as system monitoring, backup, security maintenance, load balancing and more things that are important in the production environment. Deploying a new system to production machines for these colleagues is a task that is triggered by a production-ready new system release, packaged by the development. Less new releases means less work with that task and leaves more time to anticipate intrusion attempts, network capacity problems, to be renewed certificates and so on. Rising headcount in the STSP department rapidly also had its limits. Thus, it was not easy to gain them as allies for short release cycles and early deployments.

The department *Process & Data Quality* (PDM) was easier to infect. With SCHUFA, this department is not just the quality assurance but also the domain experts for all credit bureau business processes, some kind of product development (not to be mistaken as development in the sense of software development) and revision inside IT. While agility easily can be argued to emphasize quality and bringing new products online in short cycles is also highly valued by that fraction the PDQ manager became a strong ally inside IT for agility.

One of the compromises worth to be exposed targeted the above mentioned different environments. The development was set to be in charge for two of them and PDQ took responsibility for one. Being in charge for the two production systems left enough responsibility for STSP and pulled away the thread of continuous or even daily deployments to be handled and managed, projected to be three a year.

The *Project Office* now is some kind of a department with defined responsibilities and direct strong communication paths to the IT director and all IT departments as well as to all corporate stakeholders outside IT. First established as a project itself, the *Project Office* asked for a lot of reporting data on our process, the actual projects, the consumed man-days and some more. After having learned, that the overall objective is to measure the work that we have done on projects in the department we elaborated an approach that offered a calculation basis to give numbers for workload and workload-based project dispatching and large-scale forecasts for the next two releases. At the end of the day the *Project Office* became a strong ally for objectives of agility and other contemporary improvements driven by the development.

Stakeholder: Top Management

Top managers are usually focused on visions and results and thereby need reports.

With SCHUFA some of the directors and executives from figure 1 are seated in the *Decision Board*. This is an organizational element we introduced to anticipate top managements information needs. The *Project Office* reports the over all project plan in a recurrent meeting to the *Decision Board* members. If resources are tight these reports may cause the board to terminate or shift projects as well as give more resources or concentrate resources to a distinct project.

For top managers first of all agile development is a new technique. Means: some aspects might be valuable others might not – depends on the promises given.

Thus, we tried to promise not too much and found high expectations combined with some strengthened knowledge and convictions how projects have to be conducted. And we presented the time boxing promise: “we will be in time, will not

extend the delivery date but cut features out of the deliverable if we need to; however, we will not shift the deadline".

Stakeholder: Domain Expert

The domain experts with SCHUFA were spread over all divisions (left side, figure 2) and also were strongly represented in the IT division (mainly in PDQ). From the very start, the PDQ colleagues were tightly integrated in the development process as to be questioned experts in interview and modeling sessions.

Being nearby all the time, this department shared and accompanied our way to agility. We considered that as an extraordinary opportunity to have access to on-site customers. However, the ability to join interview sessions and understand prepared models and helping it to correct or improve are not the same as modeling skills. Getting new product descriptions from *Marketing* and process improvements or erroneous functionality from *Business Operations* had to be on an interview or text basis.

Thus, we used that organizational structure with the defined over all development process. PDQ was in charge for the requirements engineering including bug reports preceding our sessions with the PDQ people. The profit for our session was that one single on-site customer is much easier to handle than a bunch of.

Stakeholder: User

There are two types of users of the new system:

Type 1 users are colleagues from the various departments and locations of *Business Operations*, sitting on an interactive user interface to the new system to answer requests or maintain the corporate data base. These users typically use workplaces of the new system. Some of the work places indirectly use SCHUFA products.

Type 2 users are other systems in which architecture the SCHUFA system usually appears as a backend system. Human communication in this case happens between the developers of the front end system and the SCHUFA system. These users are programming access to SCHUFA products.

Expected communication skills and means are completely different. We addressed type 2 users with a SCHUFA-defined and XML-based exchange language for requests and notes to the XML gateway of the new system. We prepared white papers and other developer adequate artifacts and provided a fully available system environment for the development at contract partner site. We designed the overall development process in the way that every new release with enhanced features appears live at the same time on the production environment and the contract partner test environment either. This solution completely fitted our agility-driven development needs.

We had some more to do with the type 1 users. First of all, a smooth transition was very important. And we also had to deal with terminal users that now had to use a browser interface. Thus, we noticed to have two different phases from the point of view of every individual:

1. switching to the new system with the new user interface philosophy and
2. learning additional features of a new release of the system

We addressed phase 1 with an operations team of extra-smart colleagues (pilot team) using the new system while others still used the legacy system for an interim period. Accompanied by *Human Resources* training on the new system was arranged

using the acceptance environment. The pilot team was contacted frequently by our project manager and PDQ to learn improvement suggestions and get requirements for the next releases of that workplace.

4 Organizational Options

To organize a software development unit in teams is a good idea if it becomes bigger than – to give a number – ten employees. Basically two orthogonal aspects have to be respected: project and line.

Line
Fortunately, employees tend to stay employed with the same company exceeding finished projects, sometimes even if they are external. That guarantees persistent rising of the department's know how where leaving consultants or internal employees can be seen as a disadvantage while there remains a lack of skills.

A line-based organizational structure is ideal to pass lessons learned from one project to others. It also enables a project-independent work to be allocated. Having an available budget for developing and maintaining e.g. a framework is a good way to build value inside the company – some kind of implemented and project-persistent knowledge.

Of course that position can be a threat for the top management if misused by the department manager. In addition such values tend to be uncountable because nearly no one in the market is asking for such company-specific frameworks. This is not a global valuation of domain-specific frameworks as stand-alone products. However, it's an observation of the authors at least considered to be valid for the German financial industry.

Project
At the same time, success is often aligned to projects. With a smart project organization and a good marketing of the project's results a lot of good news can be disclosed and help the development to be seen as a very useful part of the company. Since development departments use to be cost centers this topic should not be neglected. The next mighty manager who is trying to promote himself by a cost saving suggestion is sometimes just around the corner and definitely no software development department is really strong without an adequate budget.

We know that the four variables cost, quality, time and scope are interrelated in every project. In studying experiences from Kent Beck and other authors on that topic ([3], [9]) and recalling our own we decided to do what software engineers love to do: decompose. Decompose a big project in many smaller ones. Using this practice inside the department meant to get experienced and educated faster because the existing knowledge is applied and refined frequently.

Saying small project we mean that a project should not spread more than two releases of the system. Our projects usually have a project leading board of three individuals and for practical purposes it is usually set up with two managers from the

concerned domain departments and one from IT. With the help of the *Project Office* (see below) our *Project Manager* constantly communicates to the board.

Model-based Development
Applying experiences published in [1] and [12] and adding our own conviction, we saw big advantages in applying agile practices to modeling and in applying intensive modeling to our projects. Both, the internal team matters and the external communication should be positively influenced by using UML models. Thus, we think some Agile and XP publications miss an important thing. E.g. Cockburn correctly names roles like *project manager, designer/programmer, tester* and *UI expert* but does not name a role like *modeler, business modeler* or *business process designer* or something similar (see [8]).

"Think in metaphors, communicate in pictures." This thesis is widely supported by modeling with the Unified Modeling Language we used in the department.

We respect artificial paintings invented ad hoc in meetings. However, graphically refer to an existing model or add a diagram in a formal notation way (formal in the UML sense) brings a lot of meeting exceeding and context spanning value to the development. We think, the metaphor background is most valuable if you talk about a constantly refined and completed model and communicate the appropriate aspects to the distinct addressee.

In the given context of SCHUFA it turned out to be useful to grasp the application domain knowledge available at some software developed in the form of UML component and deployment diagrams. By doing so we introduced an over all building plan referred to in all phases of the projects. This model was used as some sort of fix point. All discussions about functional requirements especially with people outside the department were focused on this building plan. This building plan was maintained all the time because it just offered a set of diagrams that opened a key hole to the model for spectators. Of course, we additionally and much more intensively worked with use case, activity and class diagrams, to complete the overall model and inserted needed details for the implementation with these diagrams or with Java means in the programming sessions. Thus, we avoided never-ending discussions about diagram details, communicated the model in different depth of detail to differently skilled people.

Perhaps defining roles is the first step to discipline an XP process. However, if a team is big enough we can afford specialists without loosing agility.

SCHUFA Development

While installing an agile development philosophy on the department level we decided to replace a distinct individual based role model by a team based organizational structure. The basic idea is that teams scale up better than individuals and the bandwidth of a team's knowledge can be wider than an individual's.

Organizational Mix

Under these circumstances it is good to have a well-fitted organizational supported process that gives most possible advantages from both worlds: line and project.

We tried to apply learned lessons from project-oriented people with a background of huge projects ([7], [10], [11]), combined it with published experiences from convinced small-project people ([3], [8], [14]) and added our own.

The size of our teams varies over time and from team to team. The weakest team in headcount is the *Web Design Team*. The strongest was the *Development Team* with a headcount of 12, scaled down later on. Some may be astonished to hear that the *Project Management Team* at its high time had seven *Project Managers*. We don't want to analyze here what the main reasons for that phenomenon were. Some company based conditions as reporting expectations may have been influential factors and communication (internal and external) capacities could also be considered to be reasons. What we realized is that used terms as "project" imply expectations. Sometimes we felt a lot over-administered with company-processes suitable to monitor two or three huge projects in a period of two or three years. Having 15-30 smaller projects per year forces these huge-project expectations to scale down.

Project Office

With the *Project Office* we anticipated some major needs: the top management could not afford to sit in five or more *Project Leading Boards* but needs to keep informed especially on extraordinary events. The managers of the SCHUFA divisions sometimes felt to compete with their colleagues in occupying development resources. By shifting the responsibility from IT to top management to set priorities for the next to be conducted projects meant for some of us to escape the crossfire of "but-my-project-is-the-most-important-for-the-company" accusations. The kind of *Project Office* the SCHUFA established is far more than the fulfilling of needs described here. However, it's mainly destination and intention still is a headquarter-located information center for projects.

Living the Process

There are many books on project management out there. Some just focus on a particular sequence as [12] or write novels about virtual experiments in that domain [9]. Others have an overall understanding of projects from outside [7] or inside [2] the development team.

Inside the Software Development

From an internal point of view of the department projects realize versions of components delivered in a release of the new system (see figure 3). Components can be of stereotype *system* or *user interface component*, *workplace component*, *organizational component*, *product* and *access component*. Additional development actions have to be taken to develop and maintain the framework that realizes the architecture. Only

the development of product and workplace components is published as projects company-wide.

Projects take place in time slices called iterations. Selected stakeholders can see and feel the current development status at the end of the iteration (at least then). A pre-released version of the component will be deployed to the integration test environment inside a development release of the new system. With such previews we keep the stakeholders informed continuously. This avoided unexpected surprises.

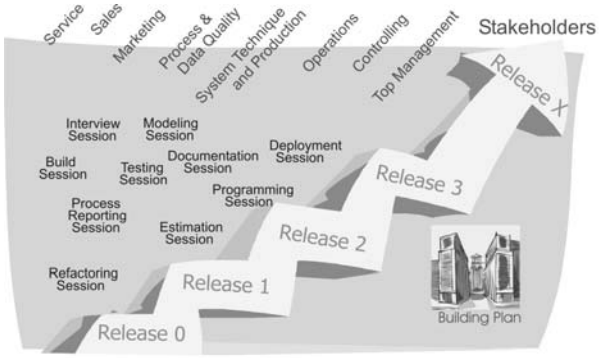

Figure 3: Building the New System in Releases

New requirements will be given full respect. However, they can only be implemented in the current release if they arrive before a so called baseline. Our baseline was a date in the middle of the release period (four iterations per release). Late arriving requirements may and usually were shifted to the next version of the component which could be the same or a following project.

The team wrote session cards to a tool to describe a bunch of work to do for realizing a demanded feature. A session card had to contain little tasks that could be taken and fulfilled in a short period of time like one or two sessions.

We classified our sessions. One worth to be mentioned is of type *Estimation Session*. At least at the beginning of each iteration to every project one session of that kind had to be planned. Another session type is called *Refactoring Session*. While easy refactorings happened in "standard" sessions as *Programming* or *Modeling Sessions* more complex ones (e.g. affecting the architecture) had to be written in a separate session card. Some similar session card is created for *Architecture Session*s but they usually were originated in architectural extensions. *Presentation Sessions* tended to be planned at the end of an iteration phase. *Documentation Sessions* were created whenever an external documentation of an implemented piece of software such as a white paper was required. *Interview Sessions* could be demanded out of a *Programming* or *Modeling Session* and could be planned by the *Project Manager*.

Modeling Sessions use UML as language of choice, *Programming Sessions* use Java. Both are usually arranged as paired sessions what means that two people work together on one computer. During the work they nearly constantly talk to each other. Main results are skilled up employees and high quality code. *Inspection Sessions* could be demanded and described by any developer to point to an observed phe-

nomenon that was under suspicion to or already had caused problems. Usually in such sessions at least one participant was from the *Architecture Team*.

Measurement

Usually project oriented organizations use to count in man days or man months. They use this method or some derived calculations as key performance indicator (KPI). The big disadvantage on calculation based on man-days is, that work time is just time not result, not work that has done or has to be done.

Our intention was to give an even better indicator for ourselves and later on to put it in a management report and publish it via the *Project Office* as capacity plan. Thus, the *KPI* has to show if more work has been mastered in the same time. Having fixed the time slices for iterations and releases these time periods seemed to be ideal for our purpose.

We decided to do some research on our existing source code over the last months to find out what changed there. Good help in such cases comes from your configuration management tool where you can step back in the time line to earlier builds. We mainly used the tools that we already applied (the UML tool and the Java IDE) to count and wrote down the numbers of use cases, activities, classes, beans and some more. The problem with counting is: whatever you count can be manipulated. Using lines of code as a calculation basis may cause programmers to change their notation style or add garbage comments. In the same way using classes could probably result in many tiny classes some times. More complicated formulas could also solve the problem but we liked to have a simple measure.

Thus, we decided to use session cards (SC) as the basic number and started without weighting different types of session cards different. We then decided that it is easier to understand for people outside the department if we avoided using more terms like session, e.g. iteration or release and simply took “month” instead as a time period indicator. We named the unit of the resulting number SDP for software development performance.

```
SDP perfMar04;
// …
perfMar04 = SD.calcPerformance(2004,3);
```

We further decided to intentionally hide the concrete formula for *calcPerformance* (work per time) because we may change it in the future from today’s *number of session cards worked on per month* to *number of session cards completed per month* or some completely different. However, the SDP as a unit should persist.

This indicator expressed in one number, how good the software development performed in a given period of time. Based on the development of that number over time further analysis of de- or increases of that unit could be taken. Having an exact number of the overall costs of the department provided by the company’s controller we were able to report how expensive a single iteration or a single release was.

With that calculation basis we looked into the past. Project stakeholders and company leaders were even more interested in the planning. So they liked to have a good estimation that meets the future reports exactly if possible.

Capacity planning

Capacity planning was a main task of a *Project Office* like ours at SCHUFA. Some of the guys there had lots of experiences with that topic based on man-day calculations. The basic idea was to have a trunk of internal employees and supplement these guys by external employees if more work has to be done in the same time. You run into a big problem with this approach if you engage external staff that is able to stretch a particular amount of work to be done over a much longer period of time. It's not easy to detect that by the *Project Office* or the management because the calculation basis is time not work (or effort).

Our SEP approach fulfilled the need to have forecasts in capacity also and we had a work-based calculation. However, you still have estimations instead of exact predictions. In addition you need to experience that kind of performance measurement a considerable time period to approximate reality with the prediction.

To start an example calculation we first had to estimate the amount of work to be done in session cards (SC): "Today we assume that realizing the component 'contract partner' is 32 SC of work to be completed."

Our experience is that this first estimated number rises by ¾. About ½ of it is discovered before the baseline is reached. That in mind we say: "We expect the organizational component 'contract partner' release 1 to be 46 SC of work."

As a further measure of experience we know that about ¼ of the features on session cards is not crucial to the success of a workplace or product component. Having that in mind we said: "34 SC have to be planned to be done if we like to go live with the contract partner component in the next release's iteration 1 and 2 of the new system."

Having our calendar-based planning in mind, social effects like holiday-time in various concerning states could be considered to make our next estimation step more realistic. A good estimation basis was the last year's months of the two iterations. E.g. we planned for February and March 2004 where Feb03 may have had a SDP of 43.7 $^{SC}/_{month}$ and Mar03 may have had a SDP of 52.7 $^{SC}/_{month}$. With this in mind we decided: "Let's plan to do 15 SC of 'contract partner' in the February iteration and 18 SC in the March iteration. That leaves 13 SC more for April and Mai for that component."

This calculation example showed a free capacity of 28.7 SC for February and 34.7 SC for March for other projects of the department.

These are the numbers we and the *Project Office* were interested in. We could then pick the next project from the priority list and go ahead the same way.

What we have learned from this easy calculation basis – to be honest – is: some people are used to calculate man days and costs for years. It's sometimes hard for them to accept the advantage of a new KPI as ours. While we strongly believe in the advantage of our KPI we keep convincing others outside step by step and go on planning this way inside the department.

Performance results

After many months of practicing the described approach we have seen that influences from early adoption phases decrease and the amount of work described in session cards does not vary as much as in the beginning any more. However, there was still a

wide range dependent from level of understanding and the individual author of the session card.

In capacity planning we are still trying to improve our predictions and forecasts in the *Project Management Team*. While *Project Managers* are individuals they tend to pack more or less work to be done in a single session card. Sometimes the first *Interview Session* completely redefines the PM's or the team's view to a feature or application that has to be realized. Since we have comparably small projects we might be able to learn quicker and to become experienced faster in that domain. We still see some more work to be done in the team especially concerning lately joined team members or external *Project Managers*. It's reassuring if you have a comprehensible calculation basis as our KPI offers. You just do not feel like a gambler in the role of a *Project Manager*.

Team building results

Some of the colleagues in the department felt like sort of lonely hackers in the beginning. A freshly started department has its own rules, of course. Accompanied by team building events the colleagues – one after the other – discovered our applied XP practices for themselves and also found their roles in the teams and the all together working process. However, that takes time. With us, about one year passed until all accepted the new way of building software and there are still some traps in the corridor in the offices. One must carefully maintain that spirit to keep the engagement, employee's satisfaction, and after all: stakeholder's satisfaction.

Conclusion

We accomplished two major projects and numerous smaller ones with agility in use as described in this article. In a period of about two years the described organizational structure was developed and helped us to arrange the processes with development participation. The team and the managers still have to "sell" agility – to the stakeholders and some team members as well. However, we have seen a huge commitment to the approach at the end of that period if we compare it to the limitations and barriers we faced in the beginning – personally and organizationally.

This approach was particularly worthwhile in the SCHUFA context, because software staff was heterogeneously qualified and because a new architecture was introduced. On the other hand, the agility approach showed its shortcomings with respect to quickly grasping the application domain knowledge and in communicating it to many developers. In order to overcome this drawback, we introduced teams of specialists in the department. Here we gained benefits from model driven approaches. That means, UML models were used to communicate the application domain knowledge and session cards to dispatch the work.

In the given situation, this combination turned out to be useful and an appropriate compromise. Future work will be devoted to describing more precisely under which circumstances a combined software engineering process as that described above can help to reconcile agile practices with organizational needs. We also intend to elabo-

rate how the approach from SCHUFA could be applied to companies with similar constellations willing to add lightweight agility to their software engineering process.

References

1. Ambler, Scott W., Agile Modeling, Wiley: New York, 2002
2. Beck, Kent, Extreme Programming Explained, Addison Wesley: Boston, 2000
3. Beck, Kent and Martin Fowler, Planning Extreme Programming, Addison-Wesley: Boston, 2001
4. Boehm, Barry and Richard Turner, Balancing Agility and Discipline, Addison-Wesley: Boston, 2004
5. Brooks, Frederick P., jr., The Mythical Man-Month, Anniversary Edition, Addison-Wesley: Boston, 1995, pp. 231-239
6. Burkhardt, Rainer, UML Unified Modeling Language, 2nd ed., Addison-Wesley: Bonn, 1999
7. Cantor, Murray R. Object-Oriented Project Management with UML, Wiley: New York, 1998
8. Cockburn, Alistair, Agile Software Development, Addison-Wesley: Boston, 2002
9. Crispin, Lisa and Tip House, Testing Extreme Programming, Addison-Wesley: Boston, 2003
10. DeMarco, Tom, The Deadline, Dorset House: New York, 1997
11. DeMarco, Tom and Timothy Lister, Peopleware, 2nd ed., Dorset House: New York, 1999
12. Erikson, H.-E. and M. Penker, Business Modeling with UML, Wiley: New York, 2000
13. Jeffries, R., A. Anderson, and C. Hendrikson, Extreme Programming Installed, Addison-Wesley: Boston, 2001
14. Wallace, Doug, Isobel Raggett and Joel Aufgang, Extreme Programming for Web Projects, Addison-Wesley: Boston, 2001