Making RUP Agile

Michael Hirsch
Zühlke Engineering AG
Wiesentrasse 10a
CH-8952 Schlieren, Switzerland
Hirsch.Michael@acm.org

Abstract

The Unified Development Process (USDP) and especially its implementation by Rational Software Corporation, the Rational Unified Process (RUP), is a comprehensive process covering almost all aspects of software development projects. However, due to the great level of detail provided by RUP, many professionals do not consider RUP practical for small, fast paced projects. This paper reports the experiences with RUP on two small projects with teams of 3 to 5 developers. RUP proved to be adaptable to the needs of small projects and was very effective in both projects. One key to the successful application of RUP in small projects is the careful selection of a proper subset of artifacts and keeping these artifacts very concise and free from unnecessary formalism. This paper goes into the details of what it takes to make RUP agile, how it was applied in the two projects, and how it was configured. Also covered is what elements of RUP contributed to the success of one project, and why RUP could not prevent that the outcome of the other project was less than optimal.

1. Introduction

This paper reports on experiences with applying RUP in small projects. A small project in this context is one with 3 to 5 developers and 6 to 9 months duration. The focus of this paper is on how we adapted RUP at Zühlke Engineering AG for this sort of project, i.e. what artifacts we used and why, the typical number and length of iterations, and the approach which we used for project planning and control.

Zühlke Engineering AG, founded in 1968, is an independant systems development services company with headquarters in Zurich and offices in Frankfurt and London. Today, we employ over 200 engineers. Our main business is custom development of software-, electronics-, mechanical- and micromechanical systems for our customers in various industries. One fundamental assumption of our business model is that the customer provides domain knowledge and we provide project mangement and technical know-how, with the consequence that systematic and unambiguous requirements engineering is of utmost importance. Typical project teams consist of 7 to 20 engineers on integrated systems[1] projects, and 3 to 10 engineers on software only projects. Software development makes up over 50% of our business volume.

For software development we used structured analysis, design, and programming up to 1991. By that time it became clear to us that structured techniques did not scale up well for complex systems and that we could not achieve the level of productivity with them we needed to stay competitive in a market with ever increasing customer expectations. In 1992 we started a transition to object oriented analysis, design and programming, which was completed by 1995. Like many other companies, we initially defined our own development process, which was rooted in the waterfall model. This turned out to be not really satisfying and by 1998 we seriously began to look for alternatives.

After a brief survey of what iterative processes were publicly available we decided to try RUP. The main reasons for this decision were that (a) RUP was the only well documented and comprehensive iterative process we could find[2], (b) RUP was very complete, thus saving us work for creating templates, guidelines and so on, (c) we got the impression that RUP was created by practitioners as opposed to methodologists assuming that we all live in a perfect world, and finally (d) documentation of RUP was available in electronic form, an important prerequisite for its applicability in real world projects by real world software engineers. The fact that we were already using the UML for object modeling routinely made it even easier to adopt RUP.

2. Adapting RUP for Small Projects

2.1 A Brief Overview of RUP

RUP covers virtually all aspects of typical software development projects. Figure 1 below shows the two most important dimensions of RUP, which are (1) the organization of a project on the time axis (shown on the horizontal axis) and (2) the areas of work in a software development project (shown on the vertical axis).

[1] An integrated systems project is one that requires skills from different engineering disciplines such as sotware-, electronics- and mechanical engineering.
[2] Extreme Programming and other Agile Methods were not widely published and easily accessible in 1998.

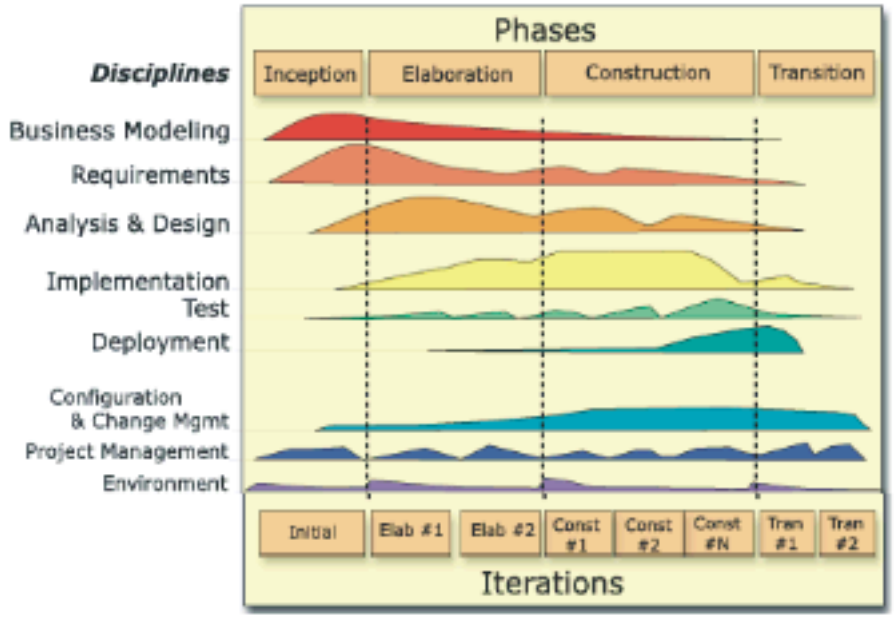


Figure 1: Overview of RUP

An area of work is called a *discipline* in RUP. These are the 9 disciplines defined by RUP:

- **Business Modeling:** Describing business processes and the internal structure of a business in order to (a) better understand the business and (b) to be able to come up with proper requirements for software systems to be built for the business at hand.
- **Requirements Management:** Eliciting, organizing, and documenting requirements.
- **Analysis and Design:** Creating the architecture and the design of the software system.
- **Implementation:** Writing and debugging source code, unit testing, and build management.
- **Test:** Integration-, system- and acceptance testing.
- **Deplyoment:** Packaging the software, creating installation scripts, writing end user documentation and other tasks needed to make the software available to its end users.
- **Project Management:** Project planning and monitoring.
- **Configuration and Change Management:** Covers all tasks concerned with (a) version and release management and (b) with change request management.
- **Environment:** Adapting the process to the needs of a project (or an organization), and selecting, introducing and supporting development tools.

The height of the bar associated with each discipline in figure 1 is an indication of how much work is spent for this discipline at a given point in time. Naturally, business modeling and requirements management need more work at early stages, whereas deplyoment needs more work towards the end of a project.

For each discipline, RUP defines a set of artifacts, activities, and roles. An *artifact* is a work product, such as a document, source code, or an object model expressed in UML. An *activity* is a detailed description of a small unit of work which creates, modifies, adds to, or reviews an artifact. A *role* is a responsibility which one or more people in a project take on, such as project manager, software architect, or test designer.

On the time axis, RUP divides a project into the following four phases:

- **Inception:** Defining the objectives of the project, including the business case.
- **Elaboration:** Creating and validating the architecture of the software system, capturing the most important and critical requirements, and planning and estimating the rest of the project.
- **Construction:** Implementing the system based on the executable architecture created in the elaboraton phase.
- **Transition:** Beta-testing the system and preparing release candidates.

Each phase is further divided into one or more iterations. Each iteration builds on the results of the previous iteration and delivers an executable (development-) release of the system. The duration and goals of an iteration are planned before an iteration starts. When an iteration is completed, a full assessment of the iteration is done in order to allow for corrective action if needed.

2.2 Considerations for Adapting RUP

In its most recent incarnation (Version 2002 of November 2001), RUP defines about 80 major artifacts, 150 activities, and 40 roles. When we first looked at RUP in 1998 there were fewer artifacts, activities and roles, but even then it was clear to us that there were too many of them for the size of projects we had in mind. So we set out to create what we called *RUP Light*, a scaled down version of RUP with as little process overhead as possible. Today, we would call this activity probably "Making RUP agile[3]" to be fully buzzword compliant, but in 1998 the term "agile methods" had not been coined yet, or at least we weren't aware of it.

We decided to adapt RUP in the following areas:

- **Artifacts:** We wanted to maintain only artifacts that were really needed and that added value, so we radically scaled down on the number of artifacts. We kept the proposed structure (i.e. the table of contents of documents) of the remaining artifacts, but made changes where we thought a template was not exactly what we wanted. This is, by the way, exactly what RUP recommends. We do not consider RUP to be at fault because we needed to customize the extent of documentation we wanted to maintain. Since RUP is a *framework* for all types and sizes of projects, some amount of customization is normal and to be expected. There is a list of all artifacts we kept in the next section.
- **Activities:** We didn't change any activities, and we decided not to use them for detailed iteration planning, contrary to what RUP suggests. The reason for this is simple: all developers on the projects were experienced in use case modeling, object modeling, the UML, and most other techniques employed by RUP. Activities and their aggregation into so-called workflow details are quite finely grained, such as "describe a use case" or "write a unit test for a class". It simply makes no sense to plan the work of experienced developers at this level of detail. Rather, we treated activities like a textbook: if a developer needed guidance on how to complete an assigned task, he would consult appropriate activity descriptions to find answers, but otherwise activity descriptions were not used.
- **Roles:** We did no formal assignments of roles to developers. We used roles only as a checklist to verify that we had all required skills in the team.
- **Project Planning:** We decided to maintain two levels of plans, as RUP suggests. The first level is a coarsely grained project plan, which basically lists start and end dates of all phases and iterations, as well as major goals for each iteration. The second level is a detailed plan for each iteration, which is prepared a few days before an iteration starts, and modified during an iteration if needed. Rather than constructing iteration plans based on a list of tasks to be done, we decided to base them on a list of results we wanted to achieve in an iteration. Typical results are the specification and implementation of a product feature, the implementation of a change request or bug fix, or the preparation of a major non-code related artifact, such as a software architecture document. Each planning item is assigned to an owner who is responsible for achieving the associated result. A typical iteration plan would call for about a dozen features to be implemented, and up to 10 change requests and bug fixes. In addition to features, change requests and bug fixes, we used a fourth category of planning items, which we called *additional work items* for a lack of a better term. Additional work items are all kinds of results which are not directly related to a single feature, such as software infrastructure components (e.g. OO/RDBMS mappings, error logging mechanisms, etc.), preparation and execution of major tests, or installation and configuration of development tools and libraries. However, the main focus of planning is on product features, because this is what the customer sees and wants (and ultimately pays for).
- **Phases:** We adopted the four phases of RUP and the associated milestones without any modificatons.
- **Iterations:** We decided to have iterations of about 4 weeks. Each iteration results in a tested software release, complete with release notes and installation scripts, which is given to the customer. To some people, 4 weeks may seem very long. However, for us an iteration is first and foremost a planning period, with a detailed list of goals we want to achieve in that period. The length of an iteration has nothing to do with how frequently we build and integrate the system. Daily builds are the rule on all our RUP projects. Customers are encouraged to frequently look at the current state of the system. We are incorporating minor change requests from customers into a running iteration. Major change requests, however, are postponed to later iterations, because they typically require replanning the overall project.
- **Project Control:** Weekly status meetings with the entire project team are our main tool for tracking progress. Since every result to be achieved is the responsibility of a single person, there are no ambiguities as to who is responsible and for what. Every week, each developer estimates the remaining amount of work to achieve his iteration goals. Thus deviations from the plan become obvious very quickly. We set up the rule that we do not prolong an iteration to achieve all results originally planned for if we run out of time. Rather, we end an iteration at the scheduled date with fewer results if we can not follow the original iteration plan. We think it is better to deliver a release with fewer features than planned on time rather than a release with all features but too late. This view is shared by most of our customers.

We summarized our thoughts and rules on how to apply RUP in an internal memo with a few pages. Other than that, no customization work was done. We deliberately did *not* modify the RUP online documentation to reflect our changes, because we did not want to repeat the work involved with every new release of RUP. To date (August

[3] See the Agile Manifesto at www.agilemanifesto.org

2002), the internal memo has evolved into a company specific RUP User Guide, which describes how to apply RUP and which is published on the SEPG (software engineering process group) homepage in our intranet.

3. Project One: Turbine Layout Tool

Armed with the guidelines and considerations outlined in section 2, we started the inception phase of our first official RUP project on October 1st, 1999. The current RUP Version then was RUP 5.1.1.

The mission of the project was to create a software tool for designing blades of steam turbines. The customer was Alstom Power Inc., one of the leading manufacturers of power generation equipment and power plants. The tool was to be used by about 20 to 30 mechanical engineers at several Alstom design centers around the globe.

3.1 Challenges

The main challenges of this project were:

- A very short development time of at most 9 months from the first idea to deployment in a production environment.
- A sophisticated user interface with 2D and 3D graphics.
- We had to integrate existing software, written by the customer in Mathlab, for some critical algorithms.
- Requirements were initially very vague, because no previous or at least similar tool existed.

We decided to meet these challenges with the following approach:

- A small team of highly experienced developers consisting of two full time developers from our company and one full time developer for Mathlab work and testing from the customer. The team was led by a part time project manager (myself).
- Application of RUP Light, as outlined in the previous section. We settled for iterations of exactly one month, starting on the first and ending on the last day of each month. In hindsight this proved to be very helpful, because it eased planning and created a steady rhythm in the project. We all knew that we were in trouble if the specifications of the features to be implemented were not clear around the 10th, or that we should be feature complete around the 25th of each iteration.
- Early and very intense involvement by the customer. In addition to the developer from the customer, who worked on our premises most of the time, the person responsible for the project from the customer attended all iteration planning and iteration assessment meetings.
- Systematic requirements management based on product features. All of the customers wishes were expressed in terms of product features. Specifications of features were kept to the minimum required and were typically from a few sentences to a few paragraphs per feature. Feature specifications were written by two engineers of the development team, one from us and one from the customer.
- Daily builds. We started with coding activities in the very first iteraration and maintained a daily build throughout the entire project.
- Implementation in Java-2. We chose Java version 1.2 standard edition as our development and target platform for the following reasons: support of 2D and 3D graphics, availability of a comprehensive GUI toolkit (Swing), higher productivity compared to development in C++, and finally the tools (Java SDK) were free.

This approach turned out to be very successful. The project was completed 2 months ahead of the initial schedule, with all required features , and most important, the customer was very satisfied.

3.2. Some Project Statistics

Table 1 below summarizes some statistics of the completed project.

Team size (headcount)	4
Team size (full time equivalents):	3.5
Number of use cases:	6
Number of features:	40
Number of change requests implemented:	53
Number of bugs found an fixed:	14
Number of iterations:	6 + 2
Project duration:	7 months
Total effort in person days:	260
Number of Java classes implemented:	approx. 180
Total LOC, including commment lines:	approx. 30,000

Table 1: Project Statistics

3.3 Project Timeline

Table 2 summarizes phases, iterations and releases of the completed project. The project started on October 1, 1999 and delivered the first production release on April 30, 2000. We spent more time and effort in the inception and elaboration phases than in construction and transition. This is typical for projects where requirements are very unclear at the beginning and need to be evolved during the project. After having productively used the software for a couple of months, the customer came back with some change requests and a few new features. We implemented these in two additional iterations. For the sake of simplicity we added these two iterations to the transition phase of the original project rather than defining a new project.

Iteration	Start Date	End Date	Milestone	Release
Inception Phase				
1	1-Oct-1999	30-Oct-1999	--	R-1.1.1 (GUI layout demonstration)
2	1-Nov-1999	7-Dec-1999	LCO	R-1.2.1 (Proof of concept)
Elaboration Phase				
3	8-Dec-1999	31-Jan-2000	--	R-1.3.1
4	1-Feb-2000	29-Feb-2000	LCA	R-1.4.1
Construction Phase				
5	1-Mar-2000	31-Mar-2000	IOC	R-1.5.2 (Beta test release)
Transition Phase				
6	1-Apr-2000	30-Apr-2000	PR	R-1.6.2 (First production release)
7	6-Nov-2000	30-Nov-2000	IOC	R-1.7.1 (Added some new features)
8	1-Dec-2000	22-Dec-2000	PR	R-1.8.1 (Second production release)

Table 2: Phases, Iterations and Releases

The milestones in table 2 are according to RUP terminology, which in turn is derived from Barry Boehm's work[4].

3.4 Project Artifacts

We decided on the artifacts we wanted to maintain at the beginning of the project and documented our decision in the so called *Development Case* document. During the project, very few artifacts were added and no artifacts were dropped. Table 3 summarizes the artifacts used by the project.

Our main criteria for the inclusion or exclusion of an artifact were the questions "what value does this artifact add for the customer?" and "what are the likely consequences if we don't have this artifact?". If we couldn't identify a convincing added value of an artifact we did not use it. As always, selecting the right artifacts involved many tradeoffs. For example, a test plan and test case descriptions would have been desirable, but we renounced them in favor of informal and ad-hoc testing. The time saved by this was invested in building more features for the customer. Fortunately we did not run into any significant quality problems despite the lack of formal testing. We attribute this mostly to the systematic software design approach used and the fact that we received frequent feedback from the customer.

3.5 Lessons Learned

Overall, this project was a big success both for our customer and for us. We think the following factors contributed strongly to the positive outcome of the project:

- Iterative and incremental development. It would have been impossible to complete this project within the same timeframe and budget with a conventional waterfall process. All fundamental assumptions of waterfall processes, i.e. that requirements can be defined upfront or that a systems software architecture can be defined before any coding work starts, did not apply to this project.
- Strong involvement of the customer in project planning and monitoring. A person from the customer was present in all iteration planning and iteration assessment meetings. The decision of what goes into an iteration in terms of features, change requests, and bug fixes was always made together with the customer.
- Fast and useful feedback from the customer. Usually we received feedback from the customer for a new release within one working week after delivering the release. In addition to the official releases, which were produced at the end of every iteration, we frequently showed the current state of our work to the customer in an informal manner. Feedback received from this was usually incorporated into the next official release.
- A small team of experienced and highly motivated developers.
- Low overhead for project planning, requirements and change management. The effort spent for project management activities was about 7% of the total effort for the project.
- A pragmatic but nevertheless effective approach to change management.
- Finally, the framework for organizing projects and the many templates and examples which come with RUP saved us a lot of time in setting up the project. Without RUP, we would have had to come up with our own definitions for project phases, work areas, artifacts and many other things.

What would we do differently today if we could start again? First of all, we would spend more time on organizing and executing the testing effort. At a minimum, we would use

[4] LCO = Lifecylce Objectives Milestone, LCA = Lifecycle Architecture Mileston, IOC = Initial Operational Capabilities Milestone, PR = Product Release Milestone.

the JUnit framework (see www.junit.org) for unit testing and a small set of documented test cases for system testing. Although we had no quality problems in this project we never felt secure about the quality of the system. Also, if the budget had a little more margin, we would collect some straightforward design metrics in order to have a quantitative view of the design quality of our system.

RUP Artifact	Comments	Format / Tools
Requirements		
Vision Document	Describes project objectives, important classes of users, and all product features. Features are uniquely identified for reference from other artifacts.	Text document, 15 pages
Use Case Model	Describes all use cases of the system in one document. Use cases were used to better understand the system, but not for project planning. Initial versions of the document included sketches of the GUI, in later versions the sketches were replaced with screen shots of the actual GUI.	Text document, 20 pages
Analysis and Design		
Software Architecture Document	Describes the high level software design of the system. Contains some UML diagrams plus brief descriptions of important mechanisms.	Text document, 12 pages
Design Model	Detailed software design of the system in terms of classes and objects and their grouping into packages. We used the CASE tool „Together / J" for object modeling. This tool keeps design information as special comments in source code.	Together / J
Implementation		
Implementation Model	In RUP terminology the implementation model is simply the collection of all artifacts required to build the system, i.e. all source files, makefiles, configuration files, etc.	Java source code files, Makefiles, JBuilder
Test		
Defect List	A list of all open and closed defects, including a brief description of each defect.	Text document, 3 pages
Deployment		
Release Notes	Were written for each release given to the customer.	Text document, 2 pages
Installation Artifacts	These are all executable files, configuration files, installation procedures, documentation etc. needed to install the system. For each release a set of installation artifacts was created and given to customer.	ZIP file
Configuration and Change Management		
Configuration Management Plan	Describes the policies for version control, release management, and change request management. We used CVS for version and release management. All artifacts of the project were placed under version control.	Text document, 8 pages
Change Request List	A list with all open and closed change requests, including a very brief description of each change request.	Text document, 4 pages
Project Management		
Software Development Plan	A coarse grained plan with a list of all planned iterations. For each iteration, the major objectives, start and end dates were specified. This plan was updated after each iteration.	Text document, 8 pages
Iteration Plan	A detailed plan for each iteration, outlining which features, change requests, bug fixes and additional work items will be done by whom.	Text document, 4-6 pages (one for each iteration)
Iteration Assessment	A summary of the results of an iteration, i.e. what objectives were reached, what features, change requests and bug fixes were actually implemented, and reasons for any deviations from the plan.	Text document, 2-5 pages (one for each iteration)
Environment		
Development Case	Describes how RUP is adapted for a project. Mostly a list of which artifacts are used and which aren't.	Text document, 6 pages
Programming Guidelines	We reused company internal Java programming guidelines.	Text document, 10 pages

Table 3: Project Artifacts

4. Project Two: Pay TV Planning System

When the turbine layout tool project was completed in April 2000, another challenging project was about to start. This time the customer was a company developing and manufacturing equipment for Pay TV operators. This company had a project under way whose goal was to develop a TV program planning system. The system should support creating program schedules for multiple Pay TV channels and controlling the equipment to actually broadcast the content according to a schedule. Technically, the system was built as a client / server application, where the server was developed by the customer and the client was contracted to us.

The challenges of this project were very similar to the previous one, i.e. unclear and changing requirements (the server part was in development, with little documentation available), a tight schedule, and the need to integrate software provided by the client.

Having been very successful with our RUP Light approach in the turbine layout tool project, we decided to try the same approach on this project. We assigned the same team plus one additional developer to the new project. Configuration of RUP was also the same, with a few small improvements. For example, we added unit testing with JUnit, a test plan, and use case storyboards. The implementation technology was the same as before, i.e. Java 2 standard edition.

Given the same team, the same process, and the same technology, we expected to repeat the success of the previous project. Unfortunately we were wrong, the difference was the customer.

4.1 Project Timeline

Here is a chronological list of how events unfolded in this project.

1-May-2000	The project starts with a scheduled completion date of 30-Oct-2000. The team is highly motivated and confident that we can meet the goals of the project.
31-May-2000	Release 1.1.1 is delivered to the customer.
June 2000	The customer provides no feedback to this release. In fact, the customer did not even install the software on their infrastructure. Many requirements are still unclear. The customer has no time to work with us on clarifying requirements. Key people from the customer are not available, because they are assigned to other urgent projects and are most of the time occupied with firefighting at *their* customer's sites all over the globe. We receive contradicting information from different people in the customer's organization. Nevertheless, we carry on developing and work with assumptions of our own where we are missing information from the customer.
30-Jun-2000	Release 1.2.1 is delivered to the customer.
July 2000	Like in June, the customer doesn't install the release delivered and provides no feedback at all. Our client is not yet integrated with the server, because a middleware layer developed by the customer which is needed to access the server is late by 2 months and will probably not be available before September 2000. We agree with the customer that we will develop a simple simulation of the middleware and the server in order to be able to test our client.
31-Jul-2000	Release 1.3.1 is delivered to the customer, including a simulation of the middleware and the server.
August 2000	Like June and July....
30-Aug-2000	Release 1.4.1 is delivered to the customer, still with a simulated middleware and simulated server.
September	For the very first time, the customer briefly looks at the software we created. As to be expected, the customer has many change request but no time to define the changes they want in a sufficient level of detail. The middleware under development by the customer's organization is cancelled. We are asked to change direction and build the project's middleware based on the simulation we developed in previous releases. The original task, i.e. developing the client, becomes second priority.
30-Sep-2000	Release 1.5.1 is develivered to the customer, with a prototype of the middleware.
October 2000	The customer hires a usability expert to create a new GUI for the client. First results from the usability expert are available by mid of October. Also, the customer changes the database system from SQL Server to Oracle and changes the database schema a couple of times. This affects the client and the middleware. Also in October, one of our developers gets seriously sick and is not available for the entire month.
31-Oct-2000	Release 1.6.1 is delivered to the customer. This is the final release for us, as the contract was time-bound and expired by end of October. The development team is completely frustrated and declines to continue to work on this project.

In our last release we could implement about 70% of the changes to the GUI requested by the usability expert, the

middleware worked but could not be completely adapted to the changed database system and database schema, many change requests from the customer were not implemented because there were no specifications for them, and some minor bugs remained in the software.

We learned later that the customer acquired a competitor and handed development of the entire system, i.e. server, client and middleware, to the acquired company.

4.2 Lessons Learned

This project was clearly no success. The customer did not get what he wanted, we were not able to complete the project as planned, and the development team was utterly demoralized. The project was not a complete failure either, at least the customer got a working piece of software, although not as complete as initially planned.

The main reasons for the unfortunate outcome of the project were the complete lack of feedback from the customer during the first 4 months of the project, and key people from the customer not being involved enough in the project. The relationship with the customer was positive and friendly during the entire project. Key people from the customer saw the necessity for user feedback and would have liked to provide more of it, but were in no position to do so because of other urgent tasks. We parted on good terms, even though the project was not a success.

The key lesson to be learned here is that no software process, regardless how sophisticated, can compensate for customer feedback. We are convinced that a different setup of RUP would not have improved results, since the root cause of the problem was not a process issue but a committment issue. In hindsight, we should have terminated the project after the second iteration when it became clear that there was no hope to receive feedback and sufficient support from the customer. Other than that, there is not much we could have done differently in order to get better results.

5. Conclusions

Since the two projects reported in this paper we have completed about a dozen more projects with RUP, all in the range of 1 to 4 person years of effort and with teams of 3 to 8 developers. The initial set of artifacts identified in the first project still forms the core of project documentation. Since then, we added unit testing with one of the xxUnit frameworks, a test plan, documented test cases for system testing, and a risk list to the set of mandatory artifacts for every project. Even so, the number of artifacts is still low and not an unneccessary burden on a project. So far we had no complaints from developers that they have to write too much documentation, which we interpret as a good sign.

Here are some suggestions to make RUP agile:

- Carefully select a small subset of artifacts and keep the level of detail of these artifacts down to a reasonable limit. Of the 80+ artifacts of RUP, only 10 to 12 are really needed on small projects.
- The list of “must have” artifacts includes a software development plan, an iteration plan and iteration assessment for each iteration, a software architecture document, a vision document with a list of required features, a change request list, a defect list, and a few others. See table 3 for further ideas.
- Iteration planning should focus on the desired results of an iteration rather than on the list of activities to be performed within an iteration. In this sense, the description of activities and workflow details in the RUP online documentation can be viewed as a textbook which is consulted when needed, rather than a template for detailed iteration planning.

The average amount of project management overhead was between 5 and 10% of the overall effort in the roughly 15 RUP projects we completed so far. This disproves claims that RUP is a management-heavy process which creates prohibitively high management costs for small projects. Apart from the second project covered in this paper, all RUP projects we did have been successful so far.

The following conclusions are applicable regardless of what concrete process is used:

- Iterative and incremental project planning are a key to success in projects with many uncertainties such as vague and frequently changing requirements, unproven technology, or an unknown customer.
- No amount of planning and project management can substitute for user and customer feedback. Iterative and incremental development can not compensate for lack of feedback.
- Even on small projects, a person from the customer's organization who can spend at least 50% of his or her time for user feedback and iteraton planning is essential.

We believe that our configuration of RUP is in the true spirit of agile methods. Ultimately, being agile is a mindset, which can be practiced with many different processes, including lightweight versions of RUP.

6. References

1. Rational Software Corporation Inc., RUP Online Documentation, Version 2002.05.00 (a commercial product of Rational Corp.)
2. Kruchten, Philippe. The Rational Unified Process / An Introduction. Addison Wesley, 2000
3. Beck, Kent. Extreme Programming Explained. Addison Wesley, 2000
4. The Agile Manifesto, www.agilemanifesto.org (as of August 2002)

zühlke
DIE DENKFABRIK.

Making RUP Agile

OOPSLA 2002 Practitioner Report

Michal Hirsch
Hirsch.Michael@acm.org

Zühlke Engineering AG
Wiesenstrasse 10a
CH-8952 Schlieren
Switzerland
www.zuehlke.com

Contents

Company Background

zühlke
DIE DENKFABRIK.

- Independent engineering and consulting company
- Founded 1968
- Offices in Switzerland, Germany and England
- Approx. 220 employees
- Custom development of
  - Software systems
  - Electronics and mechanical components
  - Devices and systems requiring skills in software, electronics, mechanical engineering and optics
- Typical project size: 1-10 person-years, 3 to 10 developers
- Additional information on our website at www.zuehlke.com

History of Software Development at Zühlke Engineering

1975: First software projects for embedded systems

1980s: Software development for minicomputers, with SA/SD/SP

1992: Decided to switch to OOT

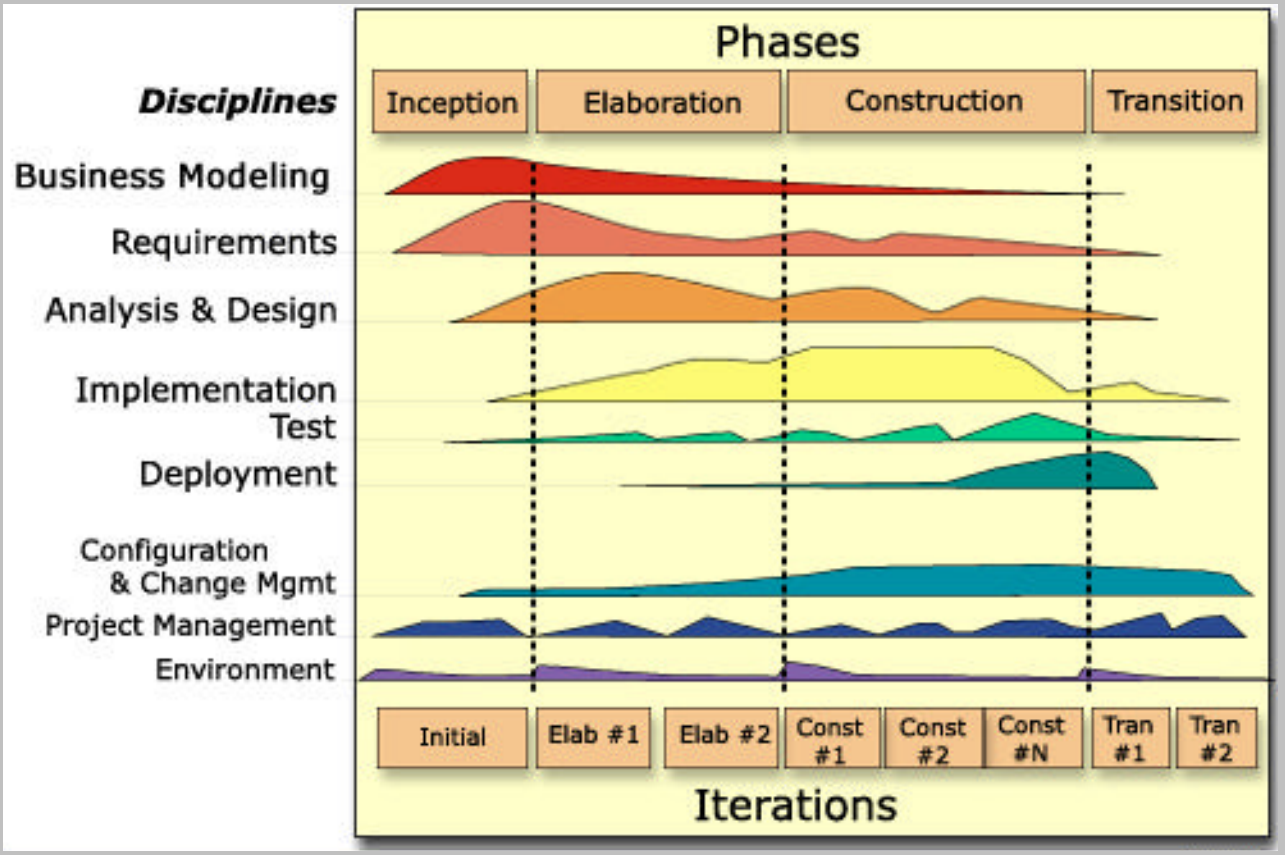
1995: Transition to OOT completed
Technology worked, but home-grown waterfall process was cumbersom

1998: Started looking for iterative / incremental processes

1999: First RUP project started

2002: 15+ RUP projects successfully completed
RUP is well established and naturally used in all software projects

RUP Overview



Some RUP numbers:

- **Current Version: RUP 2002**
- **80 major artifacts**
- **150 activities**
- **40 roles**

zühlke

How We Adapted RUP for Small Projects

- **Target project size: 3 to 8 developers, 6 to 12 months duration**
- **Artifacts: focus on 10 to 15 core artifacts, skip the rest**
- **Activities: used like a textbook, not used for planning**
- **Roles: used as a checklist of skills**
- **Project planning:**
  - **Coarse grained project plan, detailed iteration plans**
  - **Focus on results rather than tasks**
  - **Most important planning items: features**
  - **One iteration = one month**
  - **Well defined ownership of all artifacts**
- **Project control:**
  - **Weekly status meetings with estimation of remaining effort**
  - **Schedule has higher priority than functional completeness**
- **Policies documented in “RUP User Guide”**

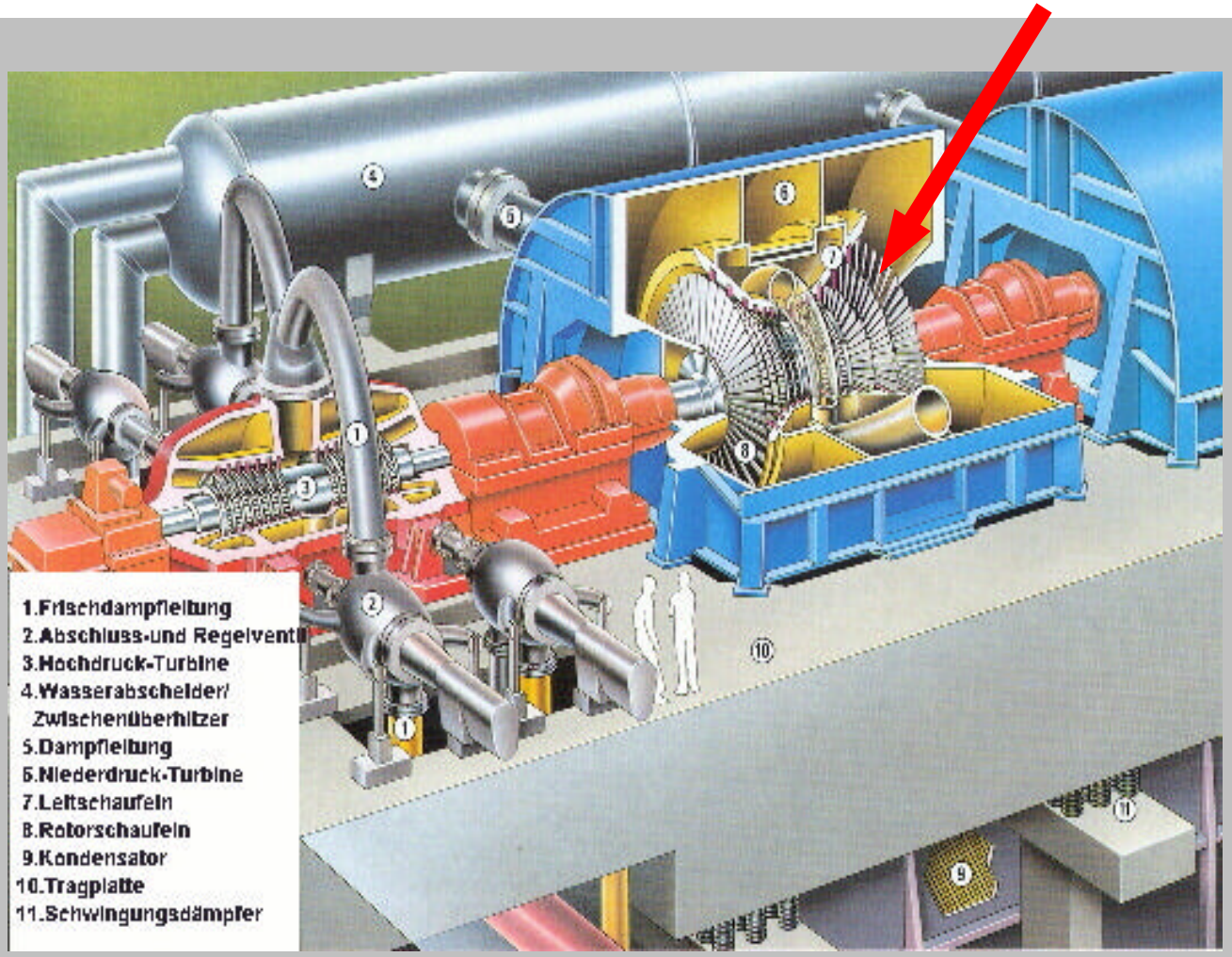
Project One: Steam Turbine Design Tool

Mission:

Build a tool to design blades of steam turbines

Customer:

Alstom Power



zühlke

Project Challenges

- Very short development time of < 9 months from first idea to first production release
- Sophisticated user interface with 2D and 3D graphics
- Integration of software developed by the customer (in Mathlab) for geometrical and thermodynamical calculations
- Initially very vague requirements, because no previous tool existed

Our Approach to meet the Challenges

- **Small team of experienced developers:**
  Two Zühlke engineers plus one engineer from the customer
- **RUP Light, with monthly iterations**
- **Early and intense involvement of the customer**
- **Systematic requirements management based on features**
- **Daily builds**
- **Implementation in Java 2 / Standard Edition**

The result: finished project two months ahead of time, with all features required and with very few defects. Big party !

Some Project Statistics

Team size (headcount):	4
Team size (full time equivalents):	3.5
Number of use cases:	6
Number of features:	40
Number of change requests implemented:	53
Number of bugs found and fixed:	14
Number of iterations:	6 + 2
Project duration:	7 months
Total effort in person days:	260
Number of Java classes implemented:	about 180
Total LOC, including comment lines:	about 30,000

Project Timeline

Iter.	Start	End	MS	Release
Inception Phase				
1	1-0ct-1999	30-Oct-1999	--	R-1-1-1 (GUI layout)
2	1-Nov-1999	7-Dec-1999	LCO	R-1-2-1 (Proof of concept)
Elaboration Phase				
3	8-Dec-1999	31-Jan-2000	--	R-1-3-1
4	1-Feb-2000	29-Feb-2000	LCA	R-1-4-1
Construction Phase				
5	1-Mar-2000	31-Mar-2000	IOC	R-1-5-2 (Beta test release)
Transition Phase				
6	1-Apr-2000	30-Apr-2000	PR	R-1-6-2 (Production release 1)
7	6-Nov-2000	30-Nov-2000	IOC	R-1-7-1 (Some new features)
8	1-Dec-2000	22-Dec-2000	PR	R-1-8-1 (Production release 2)

Project Artifacts

Requirements

- Vision Document (15 pages)
- Use Case Model (20 pages)

Analyis and Design

- Software Architecture Doc. (12 pages)
- Design Model (Together/J)

Implementation

- Implementation Model (Java Source Files, Makefiles, Jbuilder)

Test

- Defect List (3 pages)

Deployment

- Release notes (2 pages per release)
- Installation Artifacts (ZIP file)

Configuration and Change Management

- Configuration Mgmt Plan (8 pages)
- Change Request List (4 pages)

Project Management

- Software Development Plan (8 pages)
- Iteration Plans (4 to 6 pages)
- Iteration Assessments (2 to 5 pages)

Environment

- Development Case (6 pages)
- Java Programming Guidelines (10 pages)

Lessons Learned

Factors that contributed to the success of the project:

- **Iterative and incremental development (no hiding of the truth)**
- **Strong involvment of customer in project planning and monitoring**
- **Fast and useful feedback from customer**
- **Low overhead for project management (approx 7% of total effort)**
- **Pragmatic but effective change management**
- **RUP framework saved a lot of time at project setup**

Areas to improve:

- **Systematic testing effort**
- **As a minimum, add unit testing with xxUnit tools**
- **Collect some design metrics**

Project Two: Pay TV Planning System

Mission:

- **Build the client of a Pay-TV scheduling and control system**

Challenges:

- **Unclear and unstable requirements**
- **Server software simultaneously developed by customer**
- **Tight schedule**
- **No documentation of server interface**

Our Approach:

- **Same team as before, plus one additional developer**
- **Same process configuration as before**
- **Same technology as before (i.e. Java 2)**

Project Timeline

1-May-2000:	**Start of project, team is highly motivated and confident that we can repeat the previous success.**
31-May-2000:	**Release 1.1.1 delivered to customer.**
June 2000:	**No feedback from customer, customer is too busy.**
30-Jun-2000:	**Release 1.2.1 delivered to customer.**
July 2000:	**Still no feedback from customer. Middleware developed by customer is 2 months late. Customer asks us to develop a simulation of the middleware.**
31-Jul-2000:	**Release 1.3.1 delivered to customer, including simulated middleware and server.**
August 2000:	**Like June and July...**

Project Timeline (continued)

30-Aug-2000:	**Release 1.4.1 delivered, still with simulated middleware and server.**
September:	**Customer looks at the software for the first time. Many change requests concerning look and feel. Middleware developed by customer is cancelled. We are asked to change direction and extend our simulation into a real middleware. Client becomes second priority.**
30-Sep-2000:	**Release 1.5.1 delivered, with prototype of middleware.**
October:	**Customer hires usability expert to design a new GUI. Customer changes DBMS from SQL Server to Oracle.**
31-Oct-2000:	**Release 1.6.1 delivered (last release, as contract was time bound). Not all change requests implemented, a few bugs not fixed. Development team completely frustrated and declines to continue.**

Lessons Learned

- **Key lesson: No process, regardless how sophisticated, can compensate for missing feedback from customer**
- **Problem was not a process issue, but a commitment issue**
- **Being agile doesn't save you if the customer has no time for you**

Things we could have done to improve results:

- **Terminate project when it became clear that the customer could not provide feedback**
- **Maybe offload customer from other tasks to free up some time for providing feedback**

Making RUP Agile: Conclusions

- 15+ projects successfully completed since the two projects in this presentation
- “Making RUP Agile” is entirely practical and leads to an effective process if done right
- Expect 5-10% of management overhead for RUP Light
- RUP templates, guidelines, and process structure save a lot of time
- Projects become comparable, which is a big advantage in supervising project portfolios

Conclusions (continued)

Things to consider to make RUP agile:

- **Carefully select a small subset of artifacts**
- **Iteration planning should primarily focus on results (as opposed to focus on tasks to be done)**
- **An iteration is a planning period, not a single build**
- **Iterations of one month work very well**
- **Even on small projects, insist on a person from the customer with at least 50% percent of his / her time available for the project**

Ultimately, being agile is a mindset, which can be practiced with many different processes, including lightweight versions of RUP

zühlke
DIE DENKFABRIK.

Questions & Answers