The Rational Unified Process

Course PAD005 – Software Project Management 5p
Blekinge Institute of Technology, School of Engineering
Autumn 2004

By
Pär Åsfält
Asim Dedic
Samir Drincic
Markus Erlandsson
Master of Science in Computer Science and Engineering
Blekinge Institute of Technology
paas02@student.bth.se
asde02@student.bth.se
sadr01@student.bth.se
maeg02@student.bth.se

For
Patrik Berander
Blekinge Institute of Technology
Patrik.berander@bth.se

Table of content

Abstract 3
Summary 3
Workflows in the rational unified process 4
Project Management with RUP 4
Project Managers tasks and responsibilities 4
Business Modeling 4
Requirements 4
Analysis and design 4
Implementation 5
Prototypes 5
Testing 5
The configuration & change management 6
Inception 6
Elaboration 7
Construction 7
Transition 8
How does RUP fit to different types of projects? 8
Commercial (market driven) 8
In-house 9
Contract-driven 9
Disadvantages 10
Planning to death 10
Detailing too much 10
Skipping problem analysis 10
Letting end dates of iterations slip 10
Starting construction before fulfilling the exit criteria of elaboration 11
Testing only at the end of the project 11
Advantages 11
Component based design and implementation 11
Subunit integration 11
All testing 12
Iterations and its end dates 12
What about planning? 12
References 13

Abstract

The Rational Unified Process is one of the best process models in software development. It as easy as it is complex. It depends on the size and difficulty of the project. But if done right it can lead your company to great success. In the other end, if done poorly can lead your company to ruins

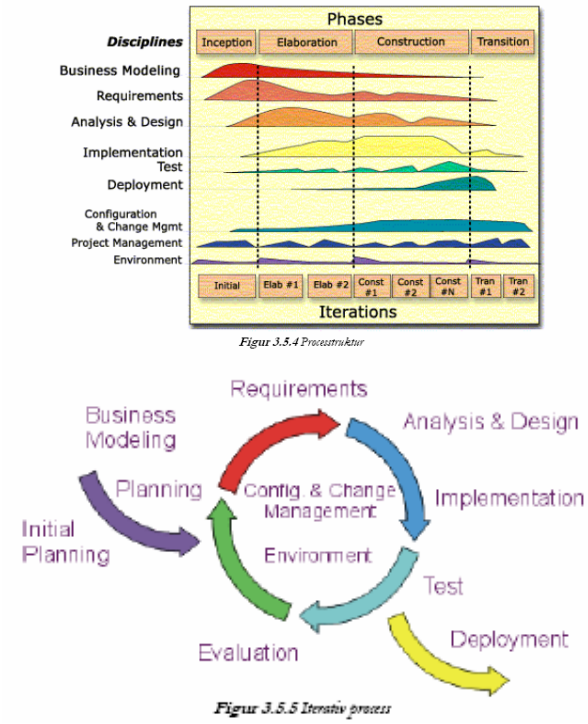
We shall try and give you a overview in how it works, so that you may lead your company to success.

Summary

RUP is an iterative process development model with cycles. That is the reason why it is easily adjusted. It is a big framework for making the process development easier. It is mainly done from 4 different phases

- Inception
- Elaboration
- Construction
- Transition

Easy phase can be divided in to more than one iteration and are edified of several workflows. Examples of workflows are Requirements, Analysis & Design, Implementation, Test and Configuration & Change management. These workflows are acting together and are their activity grade differs during the different phases and iterations.



Figur 3.5.4 Processtruktur

Figur 3.5.5 Iterativ process

There are six best practices that you are supposed to follow in RUP and these are.

- Develop software iteratively
- Manage requirements
- Use component-based architectures
- Visually model software
- Continuously verify software quality, and
- Control changes to the software.

The secrets in RUP are many and they can be summarized to these:

- Risks, RUP is risk driven and that means planning and follow-up
- Use-cases, they shall be as good written that they can be used as:
    - Formulate requirements and their behaviour.
    - Be used as test cases, user manual and design behaviour.
- Glossary, item definition.
- Iterative development
    - Little each time
    - Verification and validation
    - Frequent deliveries.

This is not new or revolutionary only a structured way of dealing with changes and their affection. Agree upon one procedure and one routine and formally structure your work. You easily discover defects in your work with your close relation to the end-user and there by also get more information about what she wants the system to do, by doing this you get a better system and a more satisfied customer.

RUP is a big framework with a lot of different methods to do different tasks. Used for smaller projects with short release cycles and to big projects that stretch over several years. RUP also sees the requirements as changeable objects and that is allowed thru the iterative process.

RUP uses the component based architecture and it is a key activity to choose which components of your system you want to integrate. Using this with the iteration means that you get a useable release in each iteration. Benefits of components are flexibility, independency and reuse of components not code.

In each iteration there is a test phase which verifies the quality for a perspective of the user stories that was the ground for the requirements. This means that you test you product against the requirements and to the real user behaviour. By doing this you increase the ability to find errors and when you use this in each iteration you find them early. The earlier you find errors the easier they are to fix.

CM has a big responsibility in this development process because there are going to be much updates and controlling this and the version handling. This is why change control is important it must be able to trace the changes in user stories and there by changes in requirements and so on.

It is important to also think about which activities are done in which order, define which components are done when, define how to measure the progress.

RUP have much similarity with XP (eXtreme Programming which has taken a lot of ideas from RUP). These are the also the strengths; see to the end-user needs (use-cases), importance of testing.

There is some weakness they might be; it takes to much time, to formal, to ceremonial. That is why it is important that you don't choose all things that RUP support because if you do, you will be buried in paperwork. The wealth of details is huge.

Workflows in the rational unified process

Project Management with RUP

Rational provides a framework for managing software-intensive projects. The framework includes guidelines for managing planning, monitoring and risks, but it doesn't guide you how to manage people, budgets or contracts.

Common planning techniques don't really work with an iterative process. It's very hard to plan if you don't know what you're going to do.

The *phase plan* is the main plan produced at the beginning of inception; it's holds dates for milestones (both minor and major) and which resources will be needed.

Each *iteration plan* consists of Gantt charts for planning and resource allocation. Kruchten (lead architect of the rational unified process product) says that these plans shouldn't be too detailed or strict because they belong in a dynamic environment, if they are; they become too hard to maintain and follow. According the guidelines you should have completed the plan for the upcoming iteration, and use the active iteration plan for monitoring the progress.

The duration of each iteration is depending on the size of the organization, it's recommended to last for two to six weeks.

The amounts of iterations vary a lot on the product, i.e. new technology tends to need more iterations. Companies that are used to work with and iterative development process can use more iterations with smaller lead-times, which according to Kruchten often results in better products.

Project Managers tasks and responsibilities

The project manager is responsible for following artifacts

- Iteration Assessment
- Iteration Plan
- Business Case
- Software Development Plan
    - Product Acceptance Plan, Risk Management Plan, Risk List, Problem Resolution Plan, Measurement Plan.
- Work order
- Status Assessment
- Project Measurements

Some daily work for the project manager in a RUP project could be:

- Planning time for approved change requests for upcoming iterations
- Monitoring risks
- Status and progress reporting
- Dealing with problems

Business Modeling

Business modeling is made in the inception phase and it's about analyzing the customers' behavior and the target organization needs.

Requirements

To every requirement we assign risks, effort and costumer priority. All the information (including prioritization) should be in the requirements document.

RUP delivers a flow schedule to follow for the requirements engineering. It helps the engineers to deal with requirements and new iterations, both on new and existing systems, analyzing problems, defining system and scope.

- Analyze the problem: identify stakeholders, boundaries and constraints.
- Understand Stakeholders Needs: Different election techniques.

One of the main benefits of an iterative development model are that requirements often change during the development and while you show the customer the prototypes.

Use cases and Scenarios are described as a very good way of finding the functional requirements.

The professionals involved in the requirements workflow are: System Analyst, Use-Case Specifier and User-Interface Designer.

Recommended tools from Rational are: Rational Rose, Rational RequisitePro and Rational SoDA (for documentation).

Analysis and design

As in all common development models the purpose of the analysis and design is to transform the requirements to a specification describing how to implement the system.

The analysis phase in the Rational Unified Process ignores many of the non functional requirements; they are handled in the design phase.

The degree of needed details in the design depends on the expertise of the implementer, the risks and the complexity of the system.

Roles that involved in the analysis and design are mainly the architect and the designer. The architect is responsible for the analysis model, the design model, interfaces and the software architecture document. The designer is responsible for the use-case-realization, the classes, packages and the subsystems. But there could also be a database designer and a capsule designer. The two most important artifacts produced in the analysis and design phases are: The design model and the software architecture document.

The design model describes the behavior of the system. To reduce complexity rational uses an approach with subsystems and packages.

The analysis model is used for over-viewing the system. It's a sketch of the system; you may see it as an abstraction or generalization of the design.

A key issue in the rational unified process is the *component based development*. Here in the design we see clearly that Rationals framework recommends the creation of subsystems and designing independent units. The independent units are connected to each other with interfaces, this leads to more flexibility, remove dependencies and increase the possibility of a higher abstraction level.

Implementation

We just told that the rational unified process follows the object oriented approach with component based development, consequently will as well the implementation go after this approach. The components are split up in different builds; this requires much integration and configuration control. The builds are used for demonstrating the system, with good configuration control different builds can be used by the developers and integrators. The integration is as well done is increments. The reason is that it's easier to locate errors and the early results boost the morale of the developers.

Prototypes

During the implementation workflow, prototypes are a common way of working. There are four common types of prototypes. First the *behavioral prototypes*: they provide a quick preview for how the system will work in a user perspective. *Structural prototypes*: The goal of the structural prototypes is to make a authentically test of the design with the actual tools that will used later on. The structural prototypes tend to be transferred to evolutionary prototypes later on. *Exploratory prototypes*: This is quick informal prototypes that are made by the developers to test the architecture of the system, mainly for personal needs or proof of concepts. *Evolutional Prototypes*: These are maybe the most important prototypes, the is large, formal, very close related to the design, rigorously tested, and often developed in more than one iteration. Often much of the code in the evolutionary prototypes is moved to the final product.

The key workers in the implementation part are of course the developers (often called implementers by Rationals framework), they are also responsible for unit testing of their contributions. As previous discussed: integrations is very central in the component based development, so another important role of the implementation phase are the integration, he construct all builds.

The structural planning for the implementation should be done early in the elaboration phase. According to Rational and Kruchten [1] this will prevent configuration management problems.

Every subset of the system should have assigned one responsible person. He's responsible for integration, sending change request for the design and control unit tests.

Testing

The obvious goal with the testing is to find all defects (and verify the system towards the requirements), but maybe the most important is to know that we created the right product (validation). In RUP you have to make sure that everything is integrated properly and the components interact well together.

The idea is that everyone is responsible for the quality. According to Kruchten [2] Quality can't be seen as a test at the end of the project, you have to start early with quality assurance.

RUP uses an iterative implementation processes, therefore the test process also have to be iterative. Rationals framework is based upon many professional software engineering experts' experiences, something that Rational call "Software Engineering Best practices", and the experiences are used to find common quality oriented problems. There are something called quality dimension, here the product are tested for common quality problems like; reliability, functionality and performance.

The Testing follows almost same steps as the integration. As told in the implementation part, the small units are tested by the implementer during the implementation. The next step is the integration test, here the integration are tested. Third step are the system test, where the whole system are tested. The forth and last stage are the acceptance test.

There are many different types of tests, like benchmark, configuration, function, installation, integrity and etc.

There are three important artifacts: The test plan, the test model and the results. The plan contains the goals and the strategies, and the test model describes what kind of test that should be executed.

The roles in the testing are primary the test designer (responsible for planning and designing tests) and the tester (responsible for executing and evaluating).

The configuration & change management

When working with this component based approach you have to track and maintain your different parts. Developers have to be able to select different versions (often newest branch) of a component to fit their needs. They should also be able to know who is responsible for a component, look at the history and the reasons for a change. Another very important part is the ability to work concurrent with the same modules; often called the *sandbox* approach. The sandbox's vision is that you can elaborate private with any shared module at your local computer without affecting others work.

Configuration management is about versions, dependencies and the concurrent version handling system (sandbox).

Change management is the structured (sometimes called bureaucratically) way of approving, handling, analyzing and implementing of changes. Every worker in the Rational Unified Process is allowed to fill in a change request of something that he/she found and think that should be changed. The change management plan defines if a change request should contain a reason or solution besides the requested change area. The change request is given to the CCB (change control board), consist often of the configuration manager, the architect and sometimes even the customer, they are responsible to decide if the change should be implemented or not. They are also responsible to analyze with other artifacts that could be affected by the proposal change, if the change are approved, they are responsible of follow up the implementation progress.

Rational provides an illustration of change request management, configuration management and status and measurements called the CCM-cube. The goal is to show how CR, CM and S&M are connected to each other and how they intertwined and responds with iterations.

Inception

The main purpose of the first phase in this process model is to decide if it is profitable to carry out the project. This decision lies on the stakeholders that are involved in the project. To their help they have several documents. The projects purpose, goals and the business case are formulated. The business case contains information to make a successful delivery, risk management, resource estimation and an project plan that shows the milestones that separate the iterations and phases apart.
Approximately half of the use-cases for the project should be identified. Description of them doesn't have to be very elaborate, but enough to give an overview of what needs to be done. That is why the business case is very important artefact during this phase.

An architecture-prototype is developed during this phase. The prototype isn't predefined in RUP but can consist of an executable code or perhaps some models. The goal of the prototype is to show that it follows the business case, but also that it is doable.

So why is this phase so important, well it is in this phase that the basis for which RUP is chosen for the project, meaning that the process is adapted to satisfy the conditions of the organization.

There are three common groups of tasks that are executed during this phase. Iteration planning, discussed earlier, the five workflows and establishing a developing-environment.

The five workflows include requirement management, analysis, design, implementation and test. Gathering requirements is done in four tasks. The first one is identifying the requirements that are in the heart of the system either from technical customers point-of-view. In the second task the team should put them selves in the system environment in order to see he system in context. Also a vocabulary development should commence in this task. The third task is to find, prioritise and specify the use-cases. This task is also divided into sub tasks but generally you can say that goal of it is to have a complete use-case with different actors and how important they are to the system. The final task of requirement management is to gather in all the non-technical requirements.

Analysis is the second activity and is divided in to two tasks. Analysing the architecture and analysing the use-cases from a more technical point-of-view, for instance what kind of resources the use-cases need. The purpose of design is to start the development of a design-model in order to make the technical aspects of the system as clear as possible for the stakeholders. The implementation is quite small in this phase. But if a prototype is build an implementation flow should be present with the activities of the implementation. Testing is almost unheard of, but the test-leader should be present in gathering of requirements so that he can get familiar with the system and its functions.

In the beginning of the project the project manager decides which goals are to be fulfilled on order to move to the next phase. He also appoints a group that is going to evaluate the work, so that it can be decided if the project should move on to the next phase.

Elaboration

The goal of the second phase is to identify as many requirements as possible in order to create a baseline for the architecture. To do that the inception phase is refined, so that the basis for the system is complete. The models from the inception phase don't work as blueprints but rather as guidelines during the elaboration phase. Architectural issues such as analysis and design are determined. Also the team should give a thought to the technical risks so that they are eliminated early in the project.

During this phase there are four groups of tasks that need to be executed, as always the five workflows, iteration-planning, evaluation and further development of development-environment.

The workflows in this phase are more elaborate and should specify the inception-phase's requirements. After the inception phase there may be approximately 50 % of the use-cases, where in this phase that number should be around 80%, so that prioritising and specifying of the use-cases is done more accurately. To help the team out the use-cases should be structured to give better understanding of the use-case-model, which gets more complex with every new use-case that is identified. Another part of requirement management is to create the user interface prototypes. Where the stakeholders can se how the system might look in the end.

During the analysis the team should evolve the primitive analysis that was constructed during the inception-phase. In the inception-phase the architectural analysis was rather incomplete, where here the analysis should lead to almost complete architecture. Also the use-cases should be analysed, in those cases that are more complex in technical sense to complete.

In the design phase they may be roughly a tenth of the whole project that is designed, and later on implemented, regarding the use-cases. Here the team is going to detail the use-cases more elaborately.

The implementation is split in three parts. The first one is of the architecture. There should be identification of the subsystems which are then tied to the specific parts of the design-model, making it work properly. The second implementation is on the classes and subsystems. And in relation to that unit testing is preformed. The final part of the implementation is the integration of the system. Even though this can be a small part of the whole system-functionality it does help in detailed planning of the future integration.

The testing is taken more seriously in the elaboration phase. Purpose is to verify the requirements that have been identified are the ones that have been designed and implemented. There are three stages that are a part of testing, and they are planning tests, designing test and executing tests. Since they are very self-explanatory there is no need for more explanation.

After each iteration there should be an evaluation. That way the system is going to have an executable release which can be verified against the requirements. The goal of the evaluation is to eliminate the biggest risks or having a contingency plan for them.

At the end of the phase the project manager needs to start planning for the first iteration of the construction phase and have an idea on how the rest of iterations are going to proceed. Also the PM should have an understanding in which order the use-cases are to be implemented.

Construction

The main purpose of the construction phase is to release a so called beta release, which can be tested by the user in a real environment, not in a development environment. Both the inception- and elaboration phase have eliminated de biggest risks or created contingency plans for them. Also a baseline is created, where the basic architecture with the important technical decisions decided. Also in this phase new use-cases must be identified. Although the most of the use-cases are identified there is some still missing and should be identified and specified during this phase.

This phase also contains the five workflows, but also iteration planning, completion of the project's business case and evaluation. During the requirement management the team should identify the rest of the use-cases. The phase can't be completed without having identified all use-cases. That's why the PM should have set the criteria for when all use-cases are found. Creation of user interface prototypes is also done in this phase. Prioritising and specification of the use-cases found is relatively easy due to the fact that the most important use-cases should have been found in the previous two phases.

The analysis should be left alone from now on in case of the architecture. But it could be used in connecting the different models, use-case-model and design-model, with the analysis-model.

In this phase the design and implementation are done as one work-flow. Until this phase there should have been approximately 80% of the use-cases identified, but only 10% of them implemented. So now the rest of them are implemented in the system. There is often problem here and that is that after each iteration belief in that the task left are easy and boring to do. So a good PM should have planed the iterations so that they slowly, logically and surely lead to a complete product. In order to integrate the system a detailed plan is needed, and that's why it uses the prioritised use-cases as reference to plan the progress. The goal is to put together the use-cases in order that the different subsystems, when implemented, create logical, functional and testable configuration.

The testing is very important during this phase. Every release is to be and the requirements implemented should be verified. The unit-tests are done on the spot by the people that have implemented them. But the integration-, function-, performance- and system-tests are done by specific test-employees.

Evaluation is done continuously throughout the testing, every iteration. But the evaluators should consider some guidelines also. And they are, if the iteration has achieved its goals, if every release is approved so that the next iteration can commence, if risk- and iteration-documents are updated and if the system-tests approve that the product is ready for the next phase.

It is very difficult for the team to plan the next phase due to the fact that the product is on its way to the customer for final implementation. The one thing that can be planed is the preliminary schedule for beta testing, appoint beta-testers and plan the test activities. Even though it's impossible to know the outcome of these tests the team should have a hunch on what the risks are and try to create a contingency plan for them.

Transition

In this phase the system is thought to be complete in sense of operating ability. PM and evaluation group have passed the system for leaving the development and installing in the customers environment. As previously mentioned some risks can have stayed hidden during the testing and are discovered in the customer-environment, but the customer can also come with new requirements. If they are not to complex they could be added to the system, otherwise a feature-list can be created where all the new requests are added and thus having a new requirement-document available for a new project.

There are two main goals of this phase and they are, that the requirements should be filled to customer's approval and to take care of any eventual problems that can arise in order to get the product fully installed.

Different from the other phases transition doesn't contain the five workflows in the same extent. Instead the focus lies on preparation of beta- and acceptance-tests, installation, managing the test-results, completing all of the projects artefacts and ending of the project.

The difference between beta- and acceptance-tests is that during beta testing the project members are rarely with the testers, while during acceptance tests should always have members from project present. That is way the beta tester should be provided with all the manuals that they could need in order to proceed with the testing. The acceptance tests are a bit more formal. If the customer accepts the test-results he then takes over the control over the product and the teams work is finished. That is way it's very important to agree on acceptance criteria before the testing begins.

Regardless of what kind of testing it is the faults discovered are divided into two groups, defects and serious problems. A defect is often related to a component fault, which luckily should be corrected without iterating a whole new phase. Luckily, because of the component based architecture. After dealing with the problem the team should ask itself following questions.

- Is the bug related, yet undiscovered, bugs?
- Can it be corrected without affecting the architecture and/or design of the system?
- Has the correction brought up new faults?

The more serious problems can lead to an new development iteration. In this case it is very important to have created change management and configuration management, because new components can be introduced and the consequences can be unmanageable.

The termination of a project can proceed when all the artefacts are completed. This is important so that the continued development of the product can proceed without complications. But the most important thing, obviously, is that the customer is pleased.

Evaluation is divided into two categories, evaluation of the phase itself and evaluation of the whole project. Evaluation of this phase is little bit different; first of all there aren't any other phases that are after this one and other projects that start could use the experience from these projects, so it should be documented. If the previous phases are done correctly there shouldn't be any problems in delivering the product.

The evaluation of the project takes time and should contain the analysis the mistakes that were done but also the things that were done well. Purpose is to create some help to future project so that they don't make the same mistakes.

How does RUP fit to different types of projects?

Here we are going to investigate how big ability to adapt RUP has.

Commercial (market driven)

Market driven project such as games or programs for companies or personal use are not that specialized for RUP. It may seem strange because not really any model have support for this kind of project. Every company has to develop the wheel over and over again.

But of course you can change the model to fit into your project. Fist you have to change the definition of the requirement and most of all you have to change the way you get them. Because market driven process is not about meeting the customers' needs it is all about guessing the markets needs. This is why you have to change the way

of looking at requirements. To not be a condition you have to meet but to look for the needs.

You have to have good knowledge about the area you will to release you product in. There is unfortunately no way to know how the market is going to act on your product. And you have little contact with the end user, how are you then going to test your product? You simply can not test it with the real end user but there is something you can do.

By adding new roles Product Manager and Requirements Database Management and adding new work flows you can make it to MRUP (Market-Driven Rational Unified Process) See [1] for more information. (Only available in Swedish, short abstract in English)

If you don't want to do this whole process that change to the MRUP you can more simpler add a department that act as customer and have fake end users, such as beta tester. This is used at some gaming development companies and they hire or borrow real players and let them try it out, by doing this they actually get to reach at least some end-users.

Advantages with using MRUP with commercial projects are that you still can use the big frameworks of tolls that RUP offers. Do you use RUP before then you don't have to change as much as you would have done otherwise.

Disadvantages with using MRUP are that it is not really tested and you might to change some from the MRUP to make it fit your project.

In-house

In-house projects are when projects are done within the same company and it is mostly done with support team. This leads to the same points that are important for the RUP and that is the close connection with the customer. In this case it is the same company but that does not really matter. What a matter is the close connection with the customer because this will result in; highly iterative approach with lots of interaction with the users and a quick feedback loop with prototypes and designs that will eliminate the fear of not giving them what they want. Some problems may occur if the company is huge and there is no god communication with the customer. This problem might be as great problems as if you have poor communication with customer maybe even worse. Because you might not recognize it as a problem in the beginning, you think hey it is only within our own company we have good communication here. Else you might be more foresighted and do something about it in an earlier stage.

Contract-driven

The RUP was create to do all type of projects according to Rational them self. But there are several points that you can't neglect that indicate that it's made for the contract driven projects. That first of all the requirements that you get from the customer and how you with you relation to the customer get changed requirements as the time goes by. RUP is based upon these requirements as simple user stories that you can use to a lot of things and as you can change during the whole process. And by its iterative development process you can deliver the releases to the customer and there by encourage testing and new changes in the next iteration.

Furthermore the contract driven project is good for RUP because you know how will be the end-users and you can test you product at the customers' users. But of course the customer might not have any end-users because she might use it in commercial purpose then we are back to the problems mentioned in that part. We can't simply only say well that is not our problem, maybe it is not on this project. But if you give the customer a product that will revolutionize the market you might have a long and prosper relationship to look forward to. But if you give your customer a product that will bankrupt the customer, fist you won't get any more contract from that customer that is obvious. Your reputations might have been defiled.

But RUP still has the problem with how to get the contract and how do you pay for contracts that you don't get?

Disadvantages

In this part we will describe few serious and large disadvantages of a Rational Unified Process.

Planning to death

This is the first and perhaps the biggest disadvantage of Rational Unified Process. Planning to death means just what it stands for and that is to plan too much and not to stop when you are supposed to.

This is described as every project manager's fear in the book "Adopting the Rational Unified Process". The problem appears as a manager has to fulfill the responsibilities of his role. Among those responsibilities is to hire the right people and to make sure that project is costly-effective. To fulfill the second part of the sentence above a manager has to define some milestones and delivery dates in order to be able to measure progress and to make sure that project is not slipping. It works to define immediate releases and final delivery date and then to measure the team performance against those.

This works for rather small project where project manager can have a big clear picture of what is to be done and by when. In a larger project though this is not possible. Larger project require more planning to make sure that resources are used in an optimal way. Also it is not enough to have some milestones and to measure progress against those. In a larger project there can be a whole year between two milestones.

It is here the mistake is done, especially by the managers using the RUP for the first time. They try to plan the project to detail for a long time ahead. The book talks about managers who try to plan the project for a 6 months period. This is not possible! We do not even know what will be developed at that time.

Many of those managers feel that they are loosing control over the project if they do not make a detailed plan but it is not the fact, it is a reality if a iterative process.

Solution to avoid this trap is to make a high level plan over the entire project and then as iteration come go more in detail and plan each iteration. This is called "just in time planning". Planning to death problem appears as the manager even though all recommendations try to make a detailed plan. This way no development work will ever be done, people will be busy planning. Another problem that will appear is that this detailed plan from the beginning of the project will be out of date just some time into the project if it is not updated.

All the people ever involved in a software project have experienced too much planning. Planning is good but as you do too much you will get the worst possible effect from it. I a project we did we planned too detailed for a too long time and the effect was that we spend 30% of out time later in the project updating the plan. If we had done just in time planning we would have saved this time and could have used it for something else.

Detailing too much

This problem is often done by the people that are moving to the iterative process from a waterfall based way of thinking. In a waterfall model you should define requirements pretty much in a lowest detail in order to move on.

This is not the fact of Rational Unified Process. In RUP you should write you requirements only so detailed so you can move on to the next iteration. After moving on everything will be evaluated and the scope and priorities of the project will be updated. If you have written to detailed requirements right at the beginning all this work will be wasted.

Skipping problem analysis

This problem is something that we have experienced.
The definition of the problem is rather simple. The team delivers what is not asked. Many times a project team will go on and define requirements, choose technology and so on without really looking at the problem that shall be solved. They simply do not know what they should deliver. If you do not know what to deliver, it does not matter how good you make it. Nobody wants the perfect product if it is not the product ordered.

In order to be sure this does not happens you have to make a proper problem analysis. This problem is written as a RUP specific but we believe that it is a problem of every process model.

As written we have experienced this kind of a project. We went on and worked towards nothing. Nobody saw the problem of a end product. Luckily this turned out good as we in the last moment actually understood what shall be done.

Letting end dates of iterations slip

This problem is really RUP-specific. Because of the huge number of iterations RUP is specially exposed to this problem.

We discussed before that it is not good to plan to death. What you should do though is to define major milestones in the overall planning. As this is done you have a global goal to go for. Project manager should now define several iterations for each milestone. If one of the phases/iterations is late you can be pretty sure that the milestone will be late too. This gives you a overview of how the project is progressing. In RUP it is recommended not to ever let an end date of a iteration slip.

To deal with this problem there are several ways. One of the solutions is to move some of the work to the next iteration and then make sure that next iteration has more

resources than previously planned so this extra work can be done in time. This will cause you to meet the end date of current iteration and enough of time to deal with the next iteration and the extra work moved to it.

The only thing to be careful about here is the so called "snowplow" effect. Pushing the work ahead of you whole the time will make you miss the overall target time. At this moment you will be forced to add more iterations and the overall end time will slip.

Starting construction before fulfilling the exit criteria of elaboration

This problem appears as the most important milestone of the Rational Unified Process if not followed. This milestone is called "completion of the elaboration phase". To pass this mile stone there are several guidelines or you can almost see them as requirements, to do. Among those there is a guideline that says that "the vision and requirements are stable". We are not going to write all the guidelines here but the problem of not fulfilling them.

If the guidelines are not fulfilled there will be problems. One of the problems will be that you will probably need time redoing the planning and take care of changed requirements. This will make the financial risks of the project to grow. Another thing that will strike you is that if you have planned for some parallel work it will not be possible. Without stable architecture it is not possible to do any parallel work.

So if there is any parallel work planned even more time should be given to this milestone.

Testing only at the end of the project

One of the clearly defined moments of Rational Unified Process is that testing should be done at end of each iteration. Even if this is clearly specified there if often no testing done until the end of the project. Many organizations skip this part from the iterations and they do all the testing in the end.

This if often done because the testing is found hard to integrate with the rest of the iterations. The result of iteration can also be so small that the company feels there is no point of testing it.

This should really be seen as a huge problem. Using RUP often means that it is a rather large project. In this kind of project you need to test all the time in order to reveal the problems while there is still time to fix them.

Advantages

Now it is time for some advantages with the Rational Unified Process.

Component based design and implementation

This is a first good thing with the Rational Unified Process that is worth describing. The RUP uses UML to fulfill its goals, and at the same time is it strongly object oriented. Those two are the foundation of a component based design and implementation.

Wherever possible the RUP tries to separate components from each other and make them independent part of the project. The result will be many small units that will be designed and implemented without dependencies on the other parts of the project. This is a very good thing for several reasons. One of them is that you now really can apply parallel work. Components are as we write independent and therefore you can apply parallel work. You do not have to wait that component X is finished to go on and work with component Y.

Next thing that makes this handy is the fact that is something goes wrong it is easy to fix. In a more traditional process model such as waterfall model you should change the whole design, update and estimate once again.

In RUP you only need to update one component. Once again this component is independent and is not affection any other components. So the only thing that you have to do is to update this components design and then implement it.

Same thing goes for totally changing a whole part of the project. If this part/component is independent this should not be a problem. Replace, design and then implement.

Subunit integration

In Rational Unified Process everything that can is divided into subunits. Every programmer in the project will receive their own code to manage. The difference between RUP and other models is that in RUP it is the programmer (for example) that is responsible to implement and then make sure that it works. Then they start on a new iteration, implementing a new component of the system.

What are the advantages some people might think? Well first of all it is easier to look into the project and see the progress. It is easier to locate where we will run late in out project and can plan regarding that.

The most important thing with this kind of integration is that failures are easy to locate. If something is not working this will be detected before the code is smashed together. So you will have perhaps 1000 rows of code to search trough rather than 60000. BIG ADVANTAGE!

Another good thing is that workers can go on and program without having to wait for something other to be done. If you have made good subunits this will be true at least. This in order leads to another thing, they can finish when they have completed the product and tested it. If

they are done with their code they can finish without having to wait for all the other parts to see the result. To see the result this fast will most likely raise the morale. It is always uplifting to see what you have accomplished.

All testing

The Rational Unified Process is rather alone to have this kind of testing. In all the models there is testing but in RUP there is this kind of testing three times!

In RUP as you have understood this far we have several iterations. In each iteration we now also know that there are several subunits. So what is the result of this?

When a product is to be developed we first make a large plan how to do it. Then we go on and make the iterations. Every iteration has its own problems to solve and parts to develop. In order to manage this we make several subunits that will be responsible to make those different parts. So now we have from one huge project made small projects on almost atomic size level. As those subunits now work they have to as we wrote before to make sure that their part is working and that there is no bugs. This will probably be well done, every programmer wants to deliver working code.

So now we have done unit-testing. To go on we now integrate all this subunits into one big chunk, and of course we have to test it again to make sure it now works as it is put together. This makes the second testing round.

Now these steps are applied in all iterations. Most often the result of previous iteration is moving into the next iteration so it is important to make sure everything is ok. Other ways next iteration will get huge problems.

After all iterations are done we have to test the entire product as well! So first we have tens or hundreds of tests (depending on the number of iterations) and then we test it all again.

We believe that this make RUP one o the most “secure” models to use. What is the chance that a bug will slip trough? It is not big, unless it is some hidden bug that is hard to find. But is we talk about usual bugs such as spelling wrong while you are writing the code there will be none.

Sure this kind of testing takes long time but it is compensated and rewarded in the end with a fault free product.

Iterations and its end dates

As we wrote before it is a death sin to let end date of an iteration slip. And nothing has changed 2 pages down, it is still a sin.

But if you do not do that sin the Rational Unified process provides a great advantage to the project. Control! We find it very handy to be able to see the progress by just looking at one single iteration. As we wrote before you only need to look at one iteration and if it is not slipping in time you do not have anything to worry about.

In other models you have to make sure too look at the much bigger picture in order to see if project will be done in time. Not needed in RUP.

And even if you lose some time and have to take action to make it back on track again the RUP will make this easy for you. Push the work that can not be done to the next iteration. As every finished iteration leads to a new iteration you can plan for this as next iteration is to be done. The RUP and its iterative nature make this kind of action very easy.

What about planning?

This far we have almost only criticized the planning in the Rational Unified Process, but it is not that bad. Or better to say it is really good. If you think about for example and a huge project together you get a headache. To plan a entire project that is very large you must have some kind of hyper threading in you head.

What happens in RUP is that you do not have to plan for a two year period. You can just make a pretty surface scratching plan for the entire period and take care of details as you do the iterations. So if you avoid the “planning to death” problem you will have a good overall plan and at the same time a good detailed plan as you need it.

Another thing that will not happen is those constant updates that we all have experienced. Because we do not use detailed two year plan we do not have to update it whole the time to make sure it is accurate, and this saves time.

References

Philippe Kruchten, *The Rational Unified Process An Introduction Second Edition*, Page 189, Addison Wesley, 2000.

Adopting the Rational Unified Process, Lotta Råberg and Stefan Bergström, Addison Wesley 2003

I. Jacobson, G. Booch, J. Rumbaugh, *The Unified Software Development Process*, Addison Wesley, 1999.

Lotta Strand, *UML & RUP – Att lyckas med oo-projekt, Docendo, 2001*

Philippe Kruchten, *The Rational Unified Process An Introduction Second Edition*, Addison Wesley, 2000.