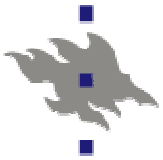
THESIS:

MODERN SOFTWARE PROJECT MANAGEMENT

STUDENT: JOAN ESCAMILLA FUSTER
DIRECTOR: JUHA TAINA
CODIRECTOR: JUAN CARLOS RUIZ GARCÍA
2009/2010



CONTENTS

1. Introduction

The thesis is about modern software engineering project management. I do some analytical comparison of methods and management styles to see if the chosen models and management styles make a difference in software development.
The thesis has three topics. First, the beginning of the thesis is about current management methods. Second, the methods principles, practices, and methodologies are studied. Finally, the thesis has a comparison of the techniques.

This research could be interesting to some software design companies because every company needs manage projects in an efficient manner. The companies find it important to get decent planning for good projects. In the beginning, it could help to give an estimate and know how much resources are needed. For example, a client asks for a budget from your company and a few competitors. Your company needs to make a quick estimate if it wants to be competitive. If this estimate is not good enough, the company may lose money because of a low estimate, or the client can reject the project because of its expenses. For this reason, a good plan work is necessary in the beginning if you want do competitive projects against other companies. Later it is useful to control the evolution of the project and to know that we are doing the necessary work and getting the objective firstly defined. Other important reason is the control of our project because we can know when we can finish the project with our resources and money.

Several iterative and incremental methods exist to control projects but nowadays agile technologies are increasing popular. It is possible that some companies may be a little scary to use it, because they have been working for many years with some old method and maybe is hard to change all of way of working. Another possibility is that the company owners think that them company is too small to implement these methods, but maybe these methods are perfect for this kind of work groups.

This thesis could be interesting even if you have or work in a company or only if you want to learn what the agile technologies are, their principles and practices. A good reason to do it is because the way of work could be applied to other sector of development not only in software. Because the main idea of these technologies is to try to connect a work team or some work teams with the client and try to develop a project.

Other sector interested in this thesis could be students because this might be a good opportunity to introduce or know something about software project management. The knowledge of these technologies may open new doors in the future.

But maybe the question is not what person can be interested in reading the thesis. Maybe the question is: why? I think that because nowadays software processes, change quickly and for this reason you need to use technologies quickly. If you are working in volatile technologies like internet or actual software development, you may need to do some change in the middle of the development because it is required for market demand. For this reason you cannot use technologies that do a change suppose almost start again or obligate the client to chose the most important decisions in the beginning of the project. If we are working in non-agile technologies the responsibility in the beginning is for the client because he must to take difficult decisions that it will lead the project to a branch or other. This responsibility is hard for the client because he maybe does not know exactly what he want or he cannot describe it. When we have the first prototype, in the final of the development, the client can look some things that he doesn't like, repair or change it could required more time and resources for this reason now the problem is for our company because this project could be not profitable.

In conclusion, if you are working in an agile market, when things are changing every day, we need to use agile technologies. We want to adapt to the client and market because it is simpler than they adapt to us.

2. Modern Software Processes

Agile technologies were born around 1995 as a reaction to structured methods that evolved from the waterfall model [Coc02]. The structured methods were heavy and slow. With agile methods are developers to erase tortuous and bureaucratic ways of traditional methodologies, because they don't want to wait to test phase to do code changes or detect errors.

However, in agile technology, developers are much more prepared to take change and correct mistakes, even those that occur near the end of the project.

There is also another advantage. The agile methodologies motivated more work teams because the programmers have our preferences when working. Generally they prefer design and programming to test and document. The agile technologies are focused on the most relevant activities for the programming, which usually coincide with the most interest to programmers.

All this, together with the fact that most agile methodologies value human relationships more than written procedures, make it much more pleasant to work with these methodologies. We cannot forget that the most motivated teams are the most productive [FoH01].

In the traditional techniques, errors and changes are detected in the test phase. For this reason, to do these changes correctly implies a return to a step of the beginning or try to change the code. This solution is not safe because it could cause other derived problems.

For all these reasons, the agile technologies have some characteristics that do the project development more quickly and easy to take (Table 2.0).

Characteristic of Agile technologies
The most important is the client
We can do a change when we want
We have Deliver working software between every 2 weeks or months
All group work together every day
Work team motivated
Meetings and face-to-face conversation

Working software marks the evolution of the project
All the work team need to work with the same constant pace
A good attention improves the agile
Is necessary simplicity
The work teams need to stay organized
The team tries to be more effective at certain intervals

Table 2.0: Characteristic of agile technologies [FoH01].

Agile technologies try to avoid methods that are used in traditional plan-driven processes (Table 2.1). Agile methods use techniques more quickly and give more weight to people and results.

Agile methodology	Traditional methodology
No very big groups	Big groups
Client is active in the process development	client doesn't form part of the process development
Requirements volatile, uncertain and that change frequently	Changing requirements occasionally
There is no fixed contract (to be adapts)	There is a contract with a price set
Group and roles to play and number of resources	Group seen as each the people who compose

Table 2.1: Comparison between agile and traditional methodology [Lar07].

The first impression of agile methods may be disorder and missing working structure. As in the traditional methods you are following some specific steps and in the agile methods these steps are not fixed. You can feel more freedom to do change or solution errors.
However, agile development is based on a set of principles as listed in Table 2.2.

Agile principles
Adaptive planning the project plan can be modified during the development
Evolutionary development. Timeboxed iterative
Customer satisfaction by rapid and continuous delivery of software
Promote evolutionary delivery
Self-organizing teams
Regular adaption to changing circumstances
Continuous attention to technical excellence and good design
Projects are built around motivated individuals, who should be trusted
Daily cooperation between business people and developers is necessary
Changes in requirements are welcomed
Values and practice that encourage agility

Table 2.2: Agile principles [Lar07]

An agile technology may have rapid and flexible response to change. For example, if a client working active with the work team have a new idea or he or she wants to change something, agile methods can adapt to these changes. Also if the project being created evolves, a client can do modifications when he wants but never without forgetting that the project has a delivery date. And the most important is that the work team must never forget working with agility and its principles (Table 2.2).

In the next points three methodologies are explained. All of them follow the principles (Table 2.2) and have some characteristics in common (Table 2.0). Scrum, XP, and RUP are the methodologies selected to explain and compare.

2.1 Scrum

Scrum is a method that regularly applies a set of best practices to work together and get the best possible results of a project. The most important characteristic is that is very simple to explain and understand, so it is simple to implement it.

In Scrum, partial deliveries are made regularly and the final results of the project are prioritized by the benefit they bring to the rest of the project. Scrum is particularly suitable for projects in complex environments where the work team need to get results

soon, requirements are changing or poorly defined, and innovation, competitively and productivity are essential.
Scrum is used to resolve situations where the client does not get what he needs, when deliveries are stretched, costs are high and the quality is not acceptable. It is useful when you need ability to react to the competition, when the moral equipment is low and the rotation high, when it is necessary to identify and solve inefficiencies systematically or want to work using a process that specializes in product development.

The Scrum development is done in an iterative and incremental manner. An iteration is a short cycle of repetitive construction. Each cycle or iteration ends with an executable piece of software that incorporates new functionality, because the members of the team work with the product and in each iteration they are incorporating new features. Iterations typically have a duration between two and four weeks [Kni09], it is named Sprint. The requirements and priorities are reviewed and adjusted during the project in very short intervals and regularly. This can be adapted in real time the product being built to customer needs. It seeks to deliver software that truly meets the needs, increasing customer satisfaction.

In Scrum, the team is focused on one thing: to build quality software. On the other hand, the Scrum project management focuses on defining the characteristics that the product must have to build and removing any obstacles that could hinder the team's task of development. The work team has to be as effective and productive as possible.

Scrum has a very small and simple set of rules. They are based on the principles of continuous inspection, adaptation, self management, and innovation [Kni09]. Developments are free to get his goals and the most important work is tried to get this objectives, without worrying about anything else. Of course every person of the work team can review or get the work of the other member because the project is visible for everyone. For this reason, Meetings are important in Scrum development because is where or the work team explained their work. Before to start every sprint the work team has a meeting and another when it is finished, also every day the work team have a meeting when every member answer three questions what is he doing?, what is wrong?, and what is the next step?. Daily scrum meeting are good to know what is doing every member of the team in a real time.

The client is excited and committed to the project because the product growing the iteration to iteration and the tools to align development with business goals are aligned to the client’s company. On the other hand, developers find an enabling environment to develop their skills this results in an increase in motivation of the team.

2.2 Extreme Programming "XP"

Extreme programming is based on communication, simplicity, feedback, courage and respect [StR03].

The simplicity helps software developers to find simpler solutions to problems. The developers also create design features that could help to solve future problems.
The communication is the most important in all practices of extreme programming. A direct communication between developers and customers is the best form of communication. Thanks to this, a team can make changes that a client does not like. Agility supports the extension of tacit knowledge within the development team, reducing the need for paper documentation. The continuous customer feedback enables developers to carry and manage the project in the right direction to where the client wants.

The courage requires developers to go hand in hand with the change. We know that change is inevitable, but being prepared with a methodology to helps to face that change. The members of the team respect work from the others and try not to depress the others. He tries to encourage them to create a good working environment.

Extreme programming is like a puzzle [Kni09]. There are many small pieces. The individual pieces do not make sense, but when combined they form a complete picture. This is a significant departure from traditional software development methods and a change in how we schedule. Every piece is explained before to program in a story card [Bec01]. Story card is a target where is written a user story. User story is defined by customer and in it is written software system requirement in two or three lines in a language understandable for customer. User stories are a quick way to manage user requirements without having to do a lot of formal documents and without using much time to administer it.

Extreme programming sets four variables for any software project: cost, time, quality and scope [Kni09]. Only three of them may be fixed or specified by the customer or project manager, while one variable will be free. Because if project manager want that his team get some software requirements with a specifics resources and he wants job finish in a determinate date, and of course with quality software. Quality will be the variable ignored, that no one is able to work well when under stress. For this reason, the project manager may to fix only three variables.

Variables among themselves bear no obvious relationship. Extreme programming emphasis on small development teams from ten to twelve people.
The project cost is increased when faster machines are needed, or more technical specialists in certain areas are required or offices for the best development team need to be present. Quality can be part of a great change because less time id required to develop higher quality for the project. Therefore, the development team is charged with the task of testing the best possible results so that they have an idea of what the problem is and how it will be resolved in a simple and efficient manner. The quality of

the project is maintained at 100% and have a facility to adapt to changes in the code that makes this process faster. Not taking into account that people work best when allowed to do quality work without pressure and can reach the project is not developed in time and quality of soils.

The methodology is designed to deliver software according to the customer needs and when necessary. The objective of XP is make quality software quickly. It is based on an iterative and incremental development with little improvements one after another. For this reason, the work team does frequent tests. In XP, it is recommended to program in pairs because the code is written and reviewed at the same time and the programming team have to work with the client because he is who know the real functionality of the application. The code must be simple and understandable and it will be rewritten to increase the legibility. The most important factor is that the code is common to all in the work team. Every member can modify or test some fragment code. It is in contrast that divide the work for everyone. The work team does tests before they add new functionality. The code is tested in each iteration [StR03]. It means that you are revising the code at the same time that you are writing it. Work team share all the code for this reason every pairs can be used the code piece that they want. But the code must to be re-implemented in system general again. In XP don't exist documentation the most similar to it is the user story but in the end of the project it will rule out, for this reason user story can be considered documentation. For this reason, XP could be perfect to develop small projects with a small work team or in project without a lot of implementation risks.

2.3 Unified Process

Unified Process is a popular incremental and iterative software development. The best known and extensively documented refinement of the Unified Process is the Rational Unified Process (RUP). For this reason, we give full attention to RUP that is a variant of UP created by IBM.

Sometimes RUP is classified as an iterative and incremental method but agile. But it is certain that it is a software development process and with the Unified Modelling Language UML it is the most widely used standard methodology for the analysis, implementation, and documentation of object-oriented systems.

The Unified Process is a generic software process that can be used for many types of software systems for different application areas, different types of organizations, different levels of competition and different sizes of projects [Kru04]. Because it define a general structure to follow and it is complete for all the kinds of projects.
Developing a commercial software product is an enormous task that can continue for several months or years. It is practical to divide the work into small pieces or mini-projects. Each mini-project is an iteration that ends in an increase. Iterations refer to steps in the workflow, increases relate to growth in the product. To be most effective,

the iterations must be controlled. They must be selected and carried out in a planned way.
A software system is created to serve users. Therefore, to build a successful system we must know what users want and need prospects. The term user refers not only to human users but to other systems. In this context, the term user represents something or someone who interacts with the system to be developed. For this reason UP is based in use cases [Kru04].

RUP Provides a disciplined approach in allocating tasks and do tasks within a development organization. His goal is to ensure production of high quality software that meets the needs of end users within a predictable schedule and budget.

The Unified Process has two dimensions [JRB99]. A horizontal axis represents time and shows the lifecycle aspects of the process throughout its development and a vertical axis representing the disciplines, which group activities in a logical manner according to their nature.
The first dimension represents the dynamic aspect of the process as it unfolds, is expressed in terms of phases, iterations, and milestones. The second dimension represents the static aspect of the process: how is described in terms of process components, disciplines, activities, workflows, artefacts and roles [JRB99].
Artefacts have a lot of important in the documentation part. A lot of artefacts are doing in the different parts of the development for example in the use case diagram in elaboration part or class diagram and sequence diagram in the construction part. The importance of the documentation is because it is used to set the way that the project will take.
An activity is a work that a person playing a role. But a process is defined by one or more workflow. A workflow is when some activities are done by a role and in this work are obtained observable results. RUP is a technology when the way to follow is stricter because it is indicate by the documentation.

3. Project Management

3.1 Background

The term software engineering was introduced around 1968 in a conference organized by NATO on software development, which was formally a branch of software engineering [Ran96]. The term is awarded to F. L. Bauer, although it had previously been used by Edsger Dijkstra in his book The Humble Programmer.

The reason that it appeared was the software crisis. It refers to the difficulty of creating easily understandable and verifiable defect-free programs. The causes are the complexity of the task of programming, the changes required in order to satisfy demanding users and problems to estimate times and resources.

There are no tools to estimate the effort needed to develop a program before starting the project. Usually it is not possible to estimate how long the project will take and how many workers will be needed. Due to this, projects often miss their deadlines. Similarly, on many occasions the staff assigned to a project increases due to a false hope of reducing the project length.

Modern software is very complex and not approachable by a single person. In the early days this assessed as well due to the immaturity of software engineering. Even today, it is not possible to make accurate estimates of the cost and time required by a software project.
Encompasses a series of events that had been observed in software development projects (Table 3.1.0)

Projects not finish on time
The project did not fit the initial budget
Low quality of software generated
Software that did not meet specifications
Unmaintainable code made it difficult to manage and develop the project

Table 3.1.0: Problems in software crisis [DFS88].

3.2 Principles and Practices

Management is available for all types of projects, of course for software projects as well. In the beginning of software development was used the traditional methods because the developers did not have special technical expertise for software projects. But nowadays the work teams could have a plan to do a project because all projects have common characteristics (Table 3.2.0).

General Characteristics	
Objective	The project must be real, sustainable and measurable
Schedule	Must have a work schedule or work plan
Complex	Is not simple and is composed of multiple elements
Demand resources	Requires skill, knowledge, capital and human effort in various areas of an organization or community
Organizational structure	Have roles and responsibilities
Monitoring and information system	Is necessary a system to record documents and information relating to the project

Table 3.2.0: General characteristics of projects [Hor09].

With this characteristics could be taken some step to start to work a project (Figure 3.2.0). In the beginning is necessary to have the first step, initiating, when the proposal was drafting in this one are describes the project objective these decisions was taken with the client. The second step is a planning work where the entire task said will be organized and assigned with its necessary resources. We are in a loop until the end of the project, when the work team and client need to do an execution of the things planned and later control that all of things are correctly done and checked. In this moment they repeat the sequence planning the next client needs executing it and controlling. This loop finishes when all the client needs are implemented. Finally, the work team starts to perform the closing of the project [Som07].

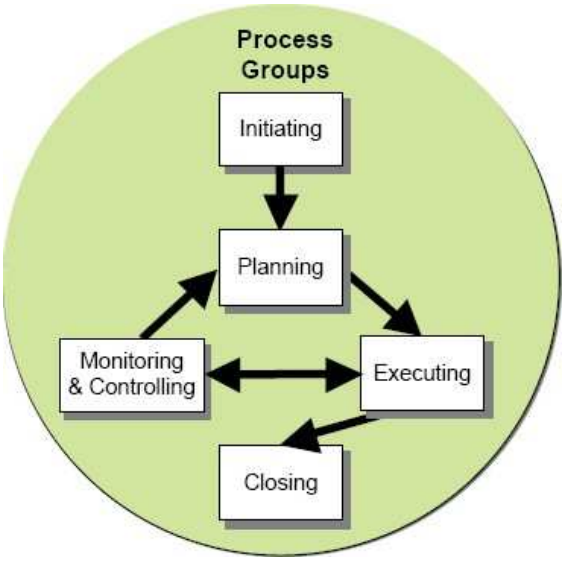


Figure 3.2.0: Steps project management [Pmg05].

4. Project Management in Modern Processes

4.1 Scrum project management

Scrum defines some roles to divide the work and the responsibilities [Sch04]. Product owner knows and fixes project priorities. Scrum Master is the responsible to save the methodology guiding the meetings and helping the team in the problems that they can to do. Scrum team is the responsible to implement the functionality chosen by the product owner. And finally the customer, he must to implicate into the project because he is the final beneficiary to project and he looking the progress has to say new ideas, suggestions or necessities.

This technology is separated in three phases [Sch04]. The first one is the planning phase when the work team takes a general vision of the project and tries to do general estimations about the price, resources and project viability. This step is quickled because as contrast to the other technologies Scrum does not need anything documented to start a project. This does not imply that some document can be done. This phase is a simple introduction to the project.

The next step is the development phase. It is the most important part of the project. In development there are three tasks: Product Backlog, Sprint Backlog and Daily Scrum Meeting (Figure 4.1.0). Product backlog is the set of all task, functionalities and requirements. The product owner defines these things. Sprint backlog is a set of task,

from the product backlog, these tasks can be changed before a sprint has started like a Sprint backlog but once the next sprint has started it cannot be modified. Every sprint backlog must be implemented in two to four weeks and it has to have realist tasks. When a sprint is finished something deliverable must to be created. When a sprint backlog is finished, a new sprint can start and in this moment some change can be done. Before to start the next sprint it is necessary have a sprint planning meeting where the task defined in the product backlog will be moved to the sprint backlog and a document named sprint goal is created it explains the sprint goals. This document will be revised when the sprint will be finished in the sprint review to make sure that all the objectives have been achieved.

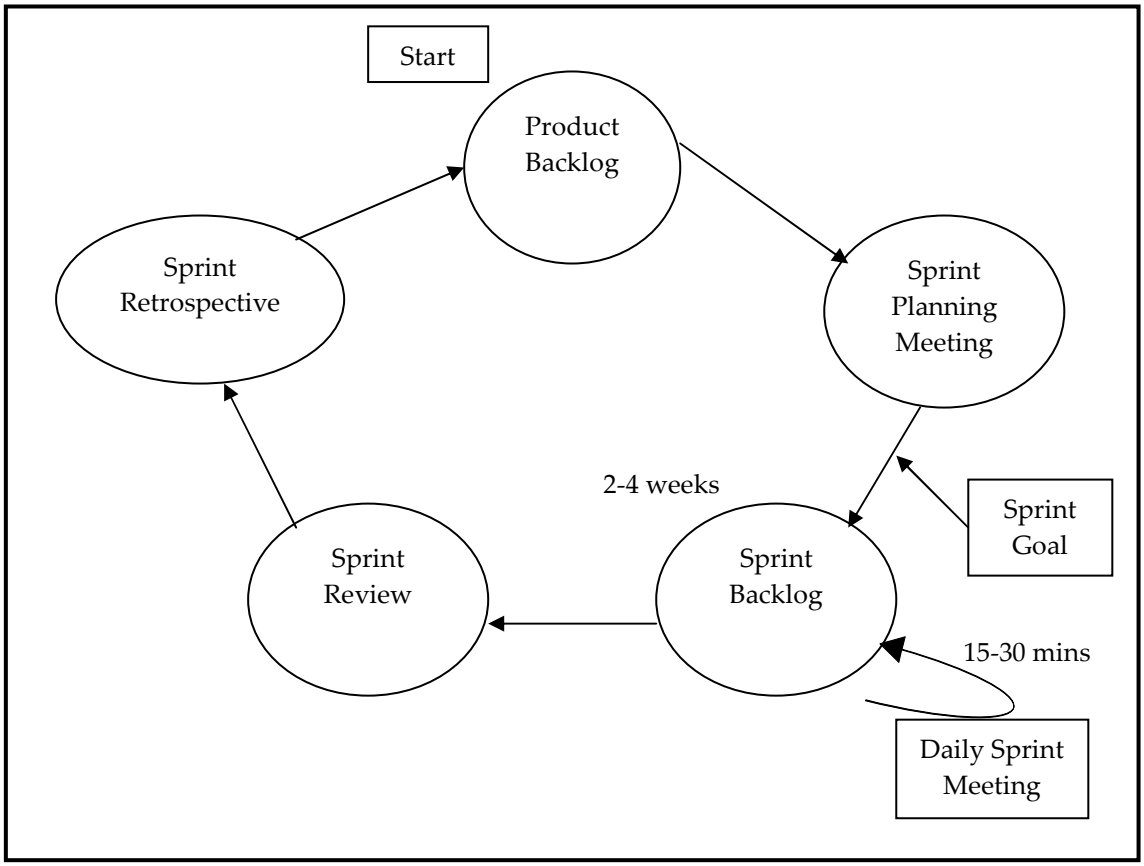


Figure 4.1.0: Scrum diagram [Sch04].

The last task of the development consist in do a meetings every day [Sch04], thirty minutes maximum, to know what has done everyone, what is doing now and say if he have some problems, the work team but especially scrum master must to delete all the obstacles to everyone.

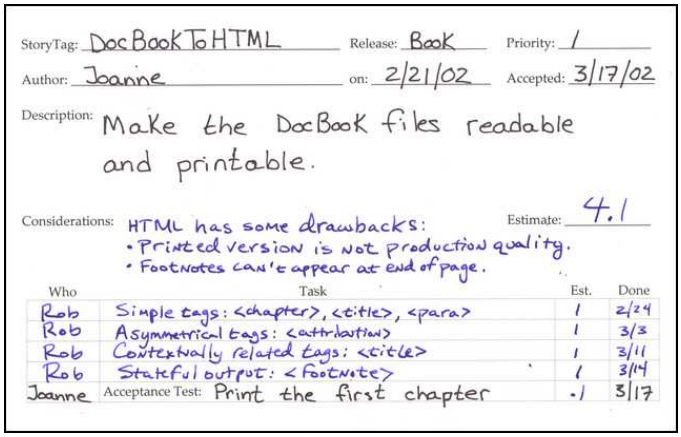
The last phase is the final delivery and assessment of the archived successes and failures. In this part, the final program is checked to know if all the goals have been

accomplished [Sch04]. This is an easy task because when every sprint is finished it is checked if the goals have been achieved.

4.2 Xp project management

Extreme programming works according some rules and principles as mentioned in point 2.3. This technology specifies also some group roles to define what everybody must do [Bec01]. The most important role is the programmer. He has to decide what short-term and long-term priorities are. He is responsible for technical decisions and for building the system. The tester helps the client with functional tests and ensures that then pass successfully. The tracker observes the process and provides feedback for future estimates. The coach is responsible for the process. He remains calm when everyone else is panicking. The consultant is not part of the team, but he helps to solve specific problems. The big boss organizes and guides the meetings and ensures appropriate conditions for the project.

The first step in development is exploration. In this part you must know if, with your team and your tools, you can finish the program [Bec01]. In exploration the programmers will use every piece of technology that is used in the production system. They work in couples and every pair tries to do every piece. Later they will compare the results, and chose the best alternative. If in a week they do not agree, this piece will be classified as a risk. The programmer should experiment with architectural ideas and performance limits of the technology. He should estimate every programming task. While the team is practicing with the technology, the customer is practicing writing stories (Figure 4.2.0).


StoryTag: DocBookToHTML Release: Book Priority: 1
Author: Joanne on: 2/21/02 Accepted: 3/17/02
Description: Make the DocBook files readable and printable.
Considerations: HTML has some drawbacks: Estimate: 4.1
• Printed version is not production quality.
• Footnotes can't appear at end of page.

Who	Task	Est.	Done
Rob	Simple tags: <chapter>, <title>, <para>	1	2/24
Rob	Asymmetrical tags: <attribution>	1	3/3
Rob	Contextually related tags: <title>	1	3/11
Rob	Stateful output: <footnote>	1	3/14
Joanne	Acceptance Test: Print the first chapter	.1	3/17

Figure 4.2.0: Story card [Nag05].

The programmer must review the first stories to teach others quickly what is needed in the story cards. This phase can continue around a few weeks or a few months. It depends on team experience because if the work team already knows their technologies they need less time for this part of development.

The second step is the planning, where the customer and the programmers must agree on a deadline for the stories [Bec01]. We must define the plan for the first release. It should be between two and six months. This phase might be quickly, a few days at most, if you did a good work in the exploration phase.

The next step is iterations to the first release. In this phase, the work is broken into one to four week iterations. Each iteration will produce a set of functional test cases for each selected stories. The selection criteria are defined by the customer because he has classified the most valuable stories [Bec01]. During the iteration, it is necessary to revise the plans. If some deviation of the plan is detected, the plan may have to change. In the end of each iteration all functional tests must be ready and running. The work team should have a ceremony to celebrate the end of the iteration.

The last step of a release is the productionizing. In this phase the iterations last one week. The team has Daily stand-up meetings so that everybody knows what others are working on [Bec01]. The work team must implement tests show that the program works because usually testing is applied parallel to this stage. During the productionizing phase software does not have to evolve because it is very dangerous for the project. When the software goes into production, a big party could be the best solution to blow off excess tension.

Maintenance is really a normal state of an XP project [Bec01]. Because new functionality must be produced and at the same time the system must be keep running, new people should incorporate into the team. Every release begins with an exploration phase where the technology may to be changed or improved. A change could be dangerous because the system is in production and some change could produce delays or bad work. If a new member is introduced to the team, the first two or three iterations his job will be to ask a lot of things and to read a lot of tests and code.

Death is the last state to the development. There are some possibilities to arrive to this phase like the customer cannot write more stories because he thinks that all the functionalities are accomplished or the customer wants to add new features but they cannot be added economically. Another possibility is that the work team persuaded the customer to realize that he project is completed [Bec01]. When the project has been finished, all the people that worked in it should celebrate with a party.

4.3 RUP project management

The Rational Unified Process unifies the entire software development team. It optimizes team communication by providing each member of the team an approach to develop an on-line customizable knowledge base according to specific project needs [JRB99]. Using online browser navigation, each team member has instant access to the RUP's knowledge base and process guidelines in their desktop. The knowledge base further unifies the team by identifying and assigning responsibilities, artifacts, and tasks so each member of the team understands his contribution to the project. By unifying your team, you can streamline communication and ensure that you efficiently allocate resources, deliver the right artifacts, and meet deadlines.

The Rational Unified Process product keeps your entire team focused on incrementally producing working software on time with required features and with required quality [JRB99]. Industry-proven best practices, contained within RUP, incorporate the lessons learned from hundreds of industry leaders and thousands of projects. You will no longer need to re-invent solutions to well-known software engineering challenges. By following RUP's iterative development approach, you will confidently deliver software on time.

RUP is divided in four phases: inception, elaboration, construction, and transition (Figure 4.3.0).

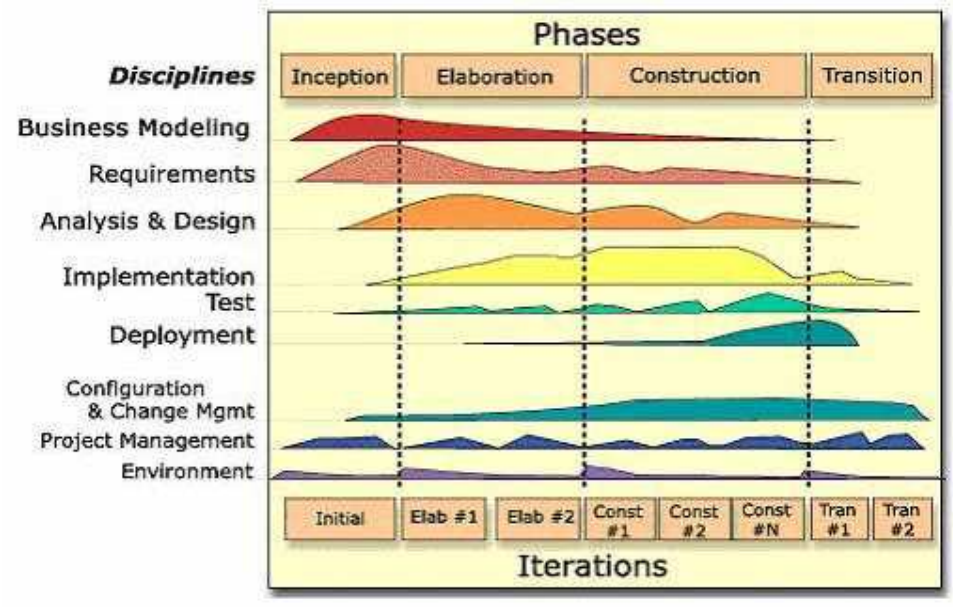


Figure 4.3.0: Phases RUP [Kru04].

Before a project has started the work team must be certain that they know the objectives, if it is feasible, how much will cost, etc. It is not necessary to have a perfect

estimation. This phase give a general idea as to the whether or not to continue with the project. Generally it should not last much longer than a week. In this step, we have an objective like to establish project scope and limits, find critical use cases of the system, show at least one candidate architecture for the main scenarios, estimate the cost in resources and time for the entire project or estimate the risks [Kru04]. This phase must generate outputs like the Business Overview that describes the objectives and constraints at high level, Use Case Model, an additional Specification like non-functional requirements, a glossary that is Terminology domain, a list of risks and contingency plans, the business case, exploratory prototypes to test concepts and architecture candidate, or an iteration plan for the first iteration of the design phase and plan phase [Kru04].
When the inception part is over all gathered requirements must be understood with use cases, cost and time must be logical and the money invested is equal to the estimated money [Kru04]. If all these items are fine we can go to the next step. If not maybe the project may be abandoned.

The next phase is elaboration. The purpose of the design phase is to analyze the problem domain, establish the foundations of architecture to develop project plan to eliminate the greatest risks [Kru04]. When you finish this phase, reaches your project point of no return. From we move from the relatively light and low-risk first two phases to address the construction phase which is costly and risky. That is why the design phase is of great importance. At this stage, we build a prototype of the architecture that must evolve in successive iterations into the final system. This prototype must include the critical use cases identified in the inception phase. You must demonstrate that it has avoided the most serious risks, either with this prototype, or with another disposable [Kru04].
The objectives of this phase are to define and validate the architecture, and to create a credible plan for the construction phase. This plan may evolve in successive iterations demonstrating that the proposed architecture will support the vision with a reasonable cost and in a reasonable time.
At the conclusion the following products must be obtained a use case model complete at least 80%: all identified cases and actors, most of the cases developed, a description of the software architecture, an executable prototype architecture, a list of risk and revised business case, a Development Plan for the project, an updated development case specifying the process to follow, and a preliminary user manual [Kru04].
To evaluate if this phase was good and the continuity with the project does not take dangers, the work team has to know that main elements of risk have been addressed and resolved, the plan for the construction phase is detailed and accurate the estimates are credible, all parties agree that the current view will be achieved if current plans are

still in the context of current architecture, the costs so far are acceptable, compared with those expected. If the evaluation criteria are not met, it may be necessary to abandon the project.

The next step is construction. The main purpose of this phase is to achieve the operational capability of the product incrementally through successive iterations. During this phase, all the components, characteristics, and requirements which have not been done so far must be implemented, integrated, and tested. The results product version can be put in the hands of users [Kru04]. The emphasis in this phase is controlling transactions, managing resources efficiently, and to optimize costs, schedules and quality.
The objective of this step is to minimize development costs by optimizing resources and avoiding having to redo work or even disposal it, and of course to obtain a quality product as fast as is practical with functional versions. This phase should be done with the UML models like use cases, implement models and design models [Kru04]. It is necessary to take attention with risks presented before and know that they are managed. The last product you need to close this phase is a transition plan.

The last phase is the transition. The purpose of this phase is to put the product into the hands of end users to develop new product updates, complete documentation, user training in the handling of the product, and general adjustment-related tasks, configuration, installation and usability of the product [Kru04]. In this part the software must to be tested and the customer has to learn the function of the program. He can use it for himself and of course the software must meet the expected requirements.
The iterations of this phase will be directed normally to get a new version. The activities undertaken during the iterations will depend on its purpose. To correct a detected error, or to carry out the workflow implementation and test. However, whether to add new features or not, this iteration will be similar to the iterations in the construction phase. The complexity of this phase depends entirely on the nature of the project, its scope and organization the implementing.

5. Analysis of project management principles

Each method has a single notation and a point of view. Nevertheless, all methods of analysis are related by a set of core principles. The domain of information and the functional domain of a problem should be represented and understood. The problem must be subdivided so as to discover progressive details and logical and physical representations of the system should be developed. In the technology explained there are different methods used but the three try to divide the work in little tasks. In Scrum, first all the tasks are defined and later they executed in sprints. In XP, the work is represented like a story and this story is selected to be a program. RUP is perhaps the most complicated model and could be difficult to look at feature. The work is divided but it is an iteration process where a prototype is updated introducing new resources and functionalities. Another common principle is that in all technologies the customer is included in the development. In XP and RUP, this is perhaps most clear. This is very important because the software is designed for the customer and if he is involved in the development the final product will be more similar to what the client wanted. The customer also exists in other roles. Everyone has his work and functions. This is an important principle to follow because if everybody knows what their tasks are and what are the tasks of rest of the work team, all the tasks will be done. If something is wrong, it is simple to know the responsible person. In RUP the work is most guided because there is more documentation and is simple follow the work done and the work not done. In XP to know the work that you have to do is necessary to look the story cards, but this can be dangerous because the work depends directly to the customer and it is not professional developing software and he can make mistakes. For this reason, the customer has to work together with the developers. In Scrum is more difficult know the work done and the work that you have to do because in a lot of cases the final product is not defined clearly and if the goals are get is evaluated by the customer. For this reason, could be simple forget some functionality if it is no requested by the customer. But the most important principle is that all of work team must be convinced to develop quality software and try to get customer needs.

I think that software project management is an interesting topic to learn and experiment, because in my opinion it is impossible to define a perfect method to software development because every work team is unique and the way of working is different for everyone. For this reason, to choose and domain a method could be the most difficult task. Of course everyone could have her preferences when choosing her technology because it is simple to read some actual conference papers like Future of scrum by Jeff Shuterland defending the use of Scrum or other experts like Ron Jeffries defending the use of XP. This is because a perfect technology does not exist but maybe there may be a method that comes close to your ideal method.

Looking for these technologies I think that there can be a change very soon because the software market development changes rapidly and the technologies that now have the most news in some years will be overtaken by others. I think that new technologies will be born within a few years because some groups of developers will not agree with the exist technologies and they will need her personal technologies. And I think that it is the key. First chose the correct technology and later change it if it is necessary to adapt to your necessities. It is not necessary to be a big company or a big work team to do this is simpler if there is a small group because to chose what is the good technologies the leader of the group must to know her team and it is simpler if the team is small. But this change can produce a change in the company and do that it begins to grow.

In conclusion, I think that the most difficult for a work team is to start and do change, because when a group team is well-working and the results are good starting a new project is simple because a try to repeat the previous project. But choose or change something about the work form could be difficult and hard.

6. Conclusion

The technologies studied have a lot of similarities because all of them try to group a team to do a project with the available resources.
In the three technology studies there are some differences. For example, in my opinion Scrum is the fastest technology because it is not necessary to make documents and the methodology is very simple to explain and understand. Scrum just has three phases and the first and the last phase are short. The development phase does not have a complicated branch. It just works in the iterative form by sprints. This can be a problem because it is necessary to have more experience in this technology because the course to follow is not as marked. The contrast is RUP where the work team must to do a lot of document and deliveries. It could be a good option for a younger team because in all moment they could know what they need to do, since he line to work is very marked. I think that it is difficult to do competitive project working with RUP if the other teams work with technologies more agile and technologies that permit more liberty. In a position in the middle is XP when the line is marked but with some flexibility. For this reason, I considerer that it could be the best technology for the most workgroups because they have a line to follow but with some liberty to work strictly. For an experimented team, I think that Scrum could be the most efficient technology because they can work in a freedom ambient and adapt the project to their necessities of them, customer or market.

7. References

[Bec01] Kent Beck, Extreme Programming explained: embrace change. Addison Wesley, 2001.

[Coc02] Cockburn, Alistair, Agile Software Development. Addison Wesley, 2002.

[DFS88] E.W. Dijkstra, W.H.J. Feijen, J. Sterringa, A Method of Programming. Addison Wesley, 1988.

[FoH01] Martin Fowler and Jim Highsmith, The agile manifesto. Software Development SketchBook, 2001.

[Hor09] Greg Horine, Absolute Beginner's Guide to Project Management. QUE Publishing, 2009.

[JRB99] Jacobon, Ivar, Rumbaugh, James, Booch, Gredy, The Unified Software Development Process. Addison Wesley, 1999.

[Kni09] Kniberg, Henrik. Scrum and XP from the Trenches. InfoQ, 2009.

[Kru04] Kruchten, Philippe. The Rational Unified Process: An Introduction (3rd Ed.). Addison Wesley, 2004.

[Lar07] Craig Larman, Agile & iterative development, A manager's guide. Addison Wesley, 2007.

[Nag05] Robert Nagler, Extreme programming in Perl. O'Reilly Media, 2005.

[PoP03] Mary Poppendieck and Tom Poppendieck, Lean Software Development: An Agile Toolkit for Software Development Managers. Addison Wesley, 2003.

[Pmg05] US DEPARTMENT OF VETERANS AFFAIRS, Project Management Guide, 2005.

[Ran96] Brian Randell, The 1968/69 NATO Software Engineering Reports. Dagstuhl, 1996, Schloss Dagstuhl.

[Sch04] Ken Schwaber, Agile software development with Scrum. Microsoft Press, 2004.

[Som07] Ian Sommerville, Software Engineering. Addison Wesley, 2007.

[StR03] Matt Stephens and Doug Rosenberg, Extreme Programming Refactored: The Case Against XP. Apress, 2003.