Institutionen för
INFORMATIK

Handelshögskolan
Göteborgs universitet

Managing Changing Requirements with Tools Supporting the Unified Process

November 1998 – January 1999

Examensarbete I

Author: Katarina Werkström
Supervisor: Bengt Hagebratt

ABSTRACT

Several modern software engineering processes advocate an iterative life-cycle approach. This involves that the set of requirements are refined throughout the system life-cycle. An ongoing refinement demands an active control of the requirements, which can be enabled with the support of a requirements management tool. This tool should provide capabilities to document, organize, track, and query requirements. It is important that the requirements management tool can be integrated with a process and tools used to support other activities of the development. It must be possible to transfer data between the different tools in order to ensure consistency. This thesis deals with the management of changing requirements according to the Unified Process, with the support of the requirements management tool RequisitePro. The requirements document of an existing system was used as a starting point. This document was imported to RequisitePro, and then altered to comply with the guidelines of the Unified Process. The results showed that the combination of the Unified Process with RequisitePro can be very useful, but that there are some problems. If requirements are organized as recommended in the Unified Process, RequisitePro makes it possible to actively control requirements by queries, traceability, and change history. The problems found concerned: lack of guidelines for documentation of use cases in RequisitePro, and difficulties with synchronization with the visual modeling tool Rational Rose. The conclusion drawn was that the solution to the problems of requirements management is found in the guidelines of a process and the experience of a skilled requirements engineer. The tool is the assistant that ensures that developers can perform their job more efficiently, and alleviates cumbersome tasks.

CONTENTS

1 INTRODUCTION

Managing the evolution of requirements for information systems is a major problem to software engineers. Problems with handling changing requirements are large contributing factors to software project failure. No matter how carefully the requirements are designed there are always changes to the original set of requirements. Requirements change because stakeholders change their minds, because the external environments change, and because developers fail to find the right requirements at the right time. The problem with changing requirements goes beyond spending time at implementing new features. It is important to know what impact a change will have on other requirements and artifacts. The effects of one small change can become far more fundamental than it was possible to imagine. Therefore it is important to structure and organize the requirements in a way that makes them resilient to changes, and easy to track.

Development of software system has evolved from chaotic and unstructured (and still is chaotic and unstructured in many ways), to a process intensive practice. Processes are needed to create order and structure, to organize the projects. There are many books and articles on the market that concerns software development processes. They vary from traditional waterfall models, to modern use case driven and iterative approaches. Even though dealt with very differently, a common factor among these processes is that requirements management is of major concern. Further, there is also consensus on the need of supporting tools to ensure compliance with certain normative criteria, such as completeness, traceability, verifiability, and reusability.

1.1 Problem Statement

Two of the essential factors for successful management of requirements are assistance from a process and tool support. The process provides guidelines for activities and artifacts such as documents and models, whereas tools facilitate active control of the requirements. Different tools concentrate on different aspects of system development, therefore it is needed that information stored in different tools can be exchanged and related to maintain consistency throughout the project.

1.1.1 Intention

The intention of this study was to investigate if a process with tool support can facilitate management of requirements, and how the process and tool support is combined. The emphasis was placed on tool support:

- How should requirements be managed with the tool to be conducted in accordance with the process?
- How is the tool support used in the most efficient way?
- Are there any possibilities of information interchange between tools supporting the process?

1.1.2 Hypothesis

The assumption made was that there is an existing process with tool support that is capable of efficient enhancement of requirements management. There are several companies that claim that their products can achieve this. One of those is the Rational Software Corporation. They propose the combination of the Unified Process and the tools RequisitePro, Rational Rose, and SQA Suite*. The hypothesis was that the Rational products could provide a solution to the problem of managing changing requirements.

1.1.3 Scope and Limits

This thesis deals with the problem of how to document, classify and structure requirements in order to manage change. This thesis is concerned with managing requirements and not how to capture them.

The Unified Process is a Software Engineering Process, which describes a family of related processes sharing a common structure, and a common architecture. A member of this family is requirements management. The guidelines conveyed by the Unified Process encompass far more than attended to in this thesis. The features of the Unified Process of interest for the current investigation were

- Guidelines for documenting and organizing requirements, in order to manage change.
- How the guidelines can be combined with the supporting tool RequisitePro.
- How RequisitePro can be synchronized with other software engineering tools, such as Rational Rose, in order to exchange information.

* The Unified Process, RequisitePro, Rose, and SQA Suite are all products of the Rational Corporation, but for simplicity the Rational prefix will be dropped in most of the future descriptions of the process and the tools.

2 PROCESSES OVERVIEW

A process can be defined as a set of ordered steps intended to reach a goal. In the concept of software engineering the goal is to build, or to change, an existing system. Even with good architecture and methods present, the software development is quite unproductive without a sophisticated development process. It is important to know what information should be put into the repository or generated out of it at each development stage, and who is responsible for each activity. The order of the steps to be followed vary from project to project, depending on which process is used. In the traditional Waterfall model the steps are sequentially ordered, whereas in the Unified Process the development is iterative.

2.1 Traditional -Waterfall Model

The process used by most software projects in the past is the Waterfall model that was first presented in 1970, in a paper by Winston Royce [13]. In this model, depicted in *Figure 1,* an activity is pursued until a document is reviewed and approved, and then this document becomes the input of the following activity. This procedure is conducted from requirements specification until the final delivery of the system.

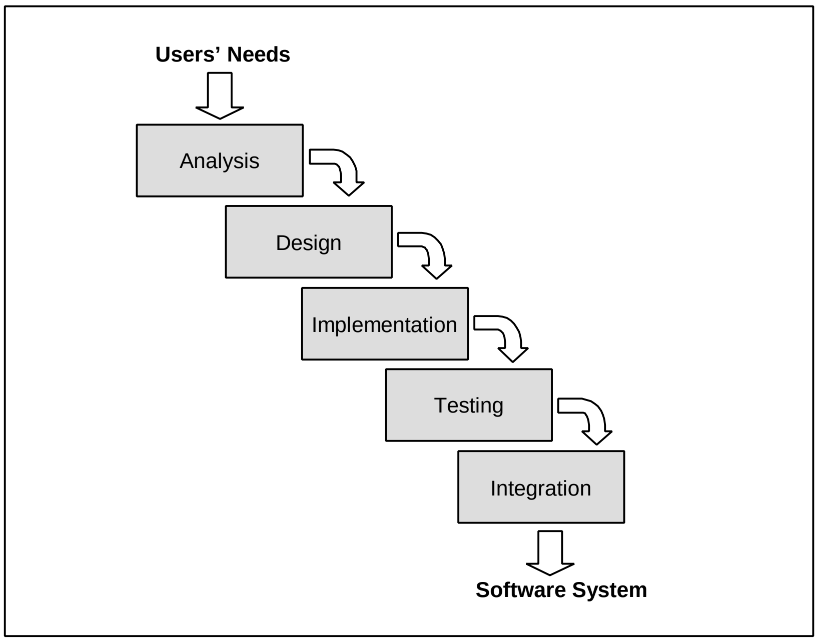


Figure 1: The Waterfall Model

The fundamental steps of the Waterfall model are [13, 14]:

1. **Analysis** – Completely understand the problem to be solved, its requirements and its constraints. Capture them in writing and get all interested parties to agree that this is what they need to achieve.
2. **Design** – Design a solution that satisfies all requirements and constraints. Examine this design carefully and make sure that all interested parties agree that it is the right solution.

3. **Implementation/Coding –** Implement the solution using your best engineering techniques.
4. **Testing –** Verify that the implementation satisfies the stated requirements.
5. **Integration** – Deliver the system. Problem solved!

In principle this model is very reasonable, as a matter of fact these are the steps used by other engineering disciplines, as e.g., when building bridges and skyscrapers. But engineers of bridges and skyscrapers have learned their "lesson" from hundreds of years of experience, whereas software engineering has been in existence only for a few decades. One of the main complaints about the Waterfall model is that it does not allow any feedback. When one activity is completed it is final and there is no return. Another complaint about the Waterfall model is that it leads up to a *big bang* [13, 14, 17]: Many errors or problems are not discovered until the end of the development when testing is performed. This result in projects being delayed and over budgeted [6].

2.2 Modern - Unified Process

The Unified Process is a controlled iterative and use-case driven software engineering process. It consists of guidelines dealing with technical as well as organizational aspects of software development.

2.2.1 Iterative Approach

Working iteratively means that the system is developed in several individual steps, where each step is a complete life cycle including analysis, design, implementation and test. In other words, an iterative process can be viewed as being composed of many small waterfall models. In order to work iteratively it is important that the iterations are controlled and that each iteration should result in a release of a subset of the final product, as e.g. a piece of executable code.

2.2.2 Use Case Driven

Use cases represents the behavior of the system. A use-case is a sequence of actions the system performs. To be defined as a use-case the actions must "…yield a result of observable value to a particular actor" [10]. This is an important factor that centers the development on users' needs. Use cases are documented both in visual models and textual descriptions. When a use case is identified it is defined briefly in a textual description called the brief description. Further on, the use case is described more thoroughly in flow of events.

The development process is driven by use-cases, from start to end, see *Figure 2*, on page 5. Requirements are expressed as use cases. In the use case descriptions classes are discovered, and the goal is that the classes should realize the complete use case model. Classes and their objects are visualized in a design model. The design model is used to structure the implementation model. The implementation model describes how classes are mapped to subsystems and their correspondent components. Components include both deliverable components, such as executables, and components from which the deliverables are produced, such as source code files. The different system sequences described by the use cases influence the structure of the implementation model. During testing, use cases constitutes the basis for identifying test cases and test procedures. Each use case is performed to verify the system behavior.

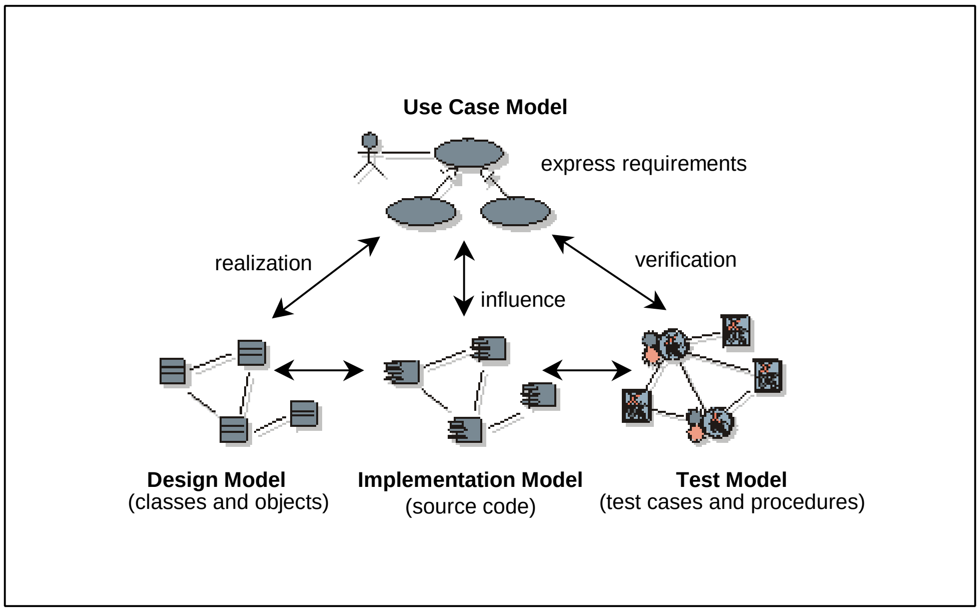


Figure 2: The Use Case Driven Approach.

2.2.3 Two Dimensions

The Unified Process is organized around two dimensions: Time (phases) and content (workflows). The time dimension is dynamic and represents the life-cycle aspect. It consists of the four phases inception, elaboration, construction and transition. For a more thorough description of these phases see Kruchten [14]. The content dimension, on the other hand, is static and consists of workflows and activities. The workflows are divided into core process workflows and supporting workflows. The core workflows are business modeling, requirements, analysis & design, test, and deployment, while the supporting workflows are configuration & change management, project management and environment, see *Figure 3*, on page 6.

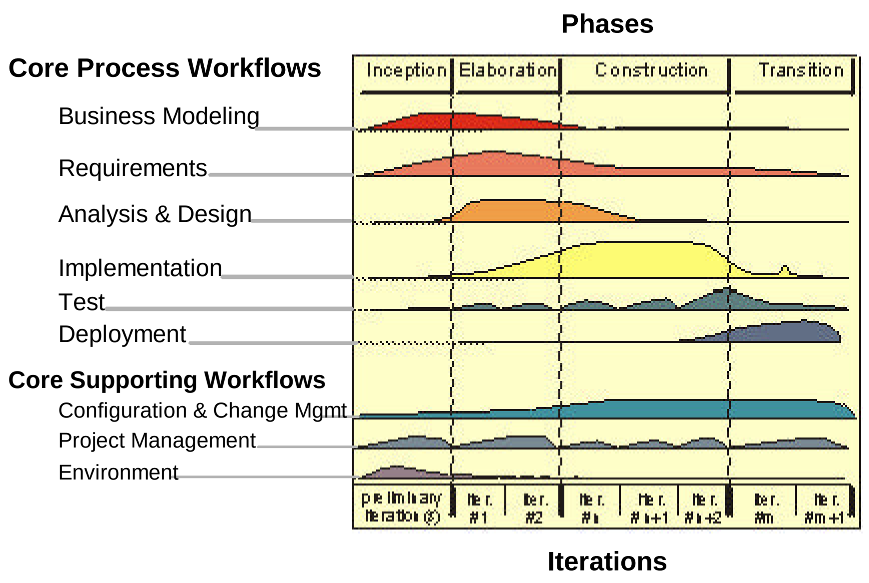


Figure 3: The Two Dimensions of The Unified Process: Phases and Workflows. The Humps Describes an Estimation of the Time Needed on Each Workflow, Distributed on Iterations.

2.2.4 Tool Support

It is possible to use the process independent of tools, but there are supporting tools that can facilitate development by automating part of the work. Some of the tools that support the Unified Process are: RequisitePro for requirements management, Rose for visual modeling, SQA Suite for testing, and SoDA for documentation.

3 REQUIREMENTS MANAGEMENT OVERVIEW

Requirements Management* is a systematic approach to establishing an agreement between the customer and the development team on the changing requirements of the system. Requirements Management consists of activities aimed at finding, organizing, documenting, and tracking requirements.

What are requirements?

Requirements are descriptions of the customers' and users' needs, their visions of the intended system. These descriptions define what the system should do, and can be used to measure the success of the implemented system. Requirements can be specified by visual models, and by textual descriptions. The requirements are the foundation of the system. When a new system is developed it is created from the requirements, any further development means that the system has to conform to the new or modified requirements. To produce a high quality product it is therefore essential to have well defined and organized requirements.

Problems with Managing Requirements

There are many problems that arise when managing requirements. Some of the major problems perceived are that requirements do not reflect the real needs of the customer, they are incomplete and inconsistent, it is expensive to make changes, and there are misunderstandings between customers and those developing the system [15]. Common causes for those problems involves the following [5, 7, 10, 11]:

- Requirements are difficult to find, because they are not always obvious and have many sources.
- Requirements change and the changes can not be tracked.
- Requirements are badly organized.
- Requirements are difficult to write.
- There are many different types of requirements at different levels of detail.

3.1 Features and Activities of Requirements Management

In order to deal with the problems of requirements management the following features and activities of requirements management can be applied: guidelines for documentation, requirement types and requirement attributes, traceability, and change history.

3.1.1 Guidelines for Documenting Requirements

Sommerville and Sawyer [15] defines the *requirements document* as:

> **…an official statement of the system requirements for customers, end-users and software developers. Depending on the organization,**

* There is an ongoing debate whether the term Requirements Engineering, or Requirements Management should be used: Are they different names for the same practice, or do they actually convey a meaningful distinction. This will not be debated here. As far as this study is concerned the terms could be used interchangeably to describe a practice that includes elicitation, analysis, specification, verification, and management of requirements.

the requirements document may have different names such as 'functional specification', 'the requirements specification (SRS), 'the safety/reliability plan', etc. These documents are all basically similar. They specify what services the system should provide, system properties such as reliability, efficiency, etc. and the constraints on the operation and (…) the development of the system.

At the start of a project, before eliciting the requirements, guidelines must be drawn that provide instructions about how to document the requirements. This should be done to assure that the requirements are documented in a consistent manner, which will make them easier to review and manage. The decision is not whether requirements should be documented, but how to document them.

The guidelines for documenting requirements should contain information about [5, 10, 15, 16]:

- What document types to use, and a definition of the structure of those document types.
- Instructions about how to capture requirements in the document, i.e., in textual descriptions, visual models, representation language (e.g. UML), etc.
- What requirement types should be used, and what attributes should the different types possess.
- How much time should be spent on documenting

3.1.2 Requirement Types

There are many kinds of requirements, which makes it convenient to arrange them into different types and subtypes. A requirement type is a class or group of requirements that have a common set of attributes. By organizing requirements by types they can be distributed to smaller, more manageable units. This is especially useful when dealing with large sets of requirements.

Different Levels of Requirement Types

Requirement types can be decomposed into varying levels of specificity [7, 10, 11, 15, 16]. All requirements can be categorized as either functional or non-functional. A functional requirement defines the behavior of the system in terms of the required inputs and outputs. Non-functional requirements, on the other hand, define the attributes and constraints on the system. This is a very general classification of requirements, which can be refined in an infinite number of ways. In the Unified Process high level requirements are documented in vision statements as product requirement types. More detailed functional software requirements are expressed as use cases, which in turn are used to derive test requirements. The non-functional requirements are documented either in a special section of the use case specification, or in a global supplementary specification. The allocation of requirement types to different documents in the Unified Process is depicted in *Figure 4,* below.

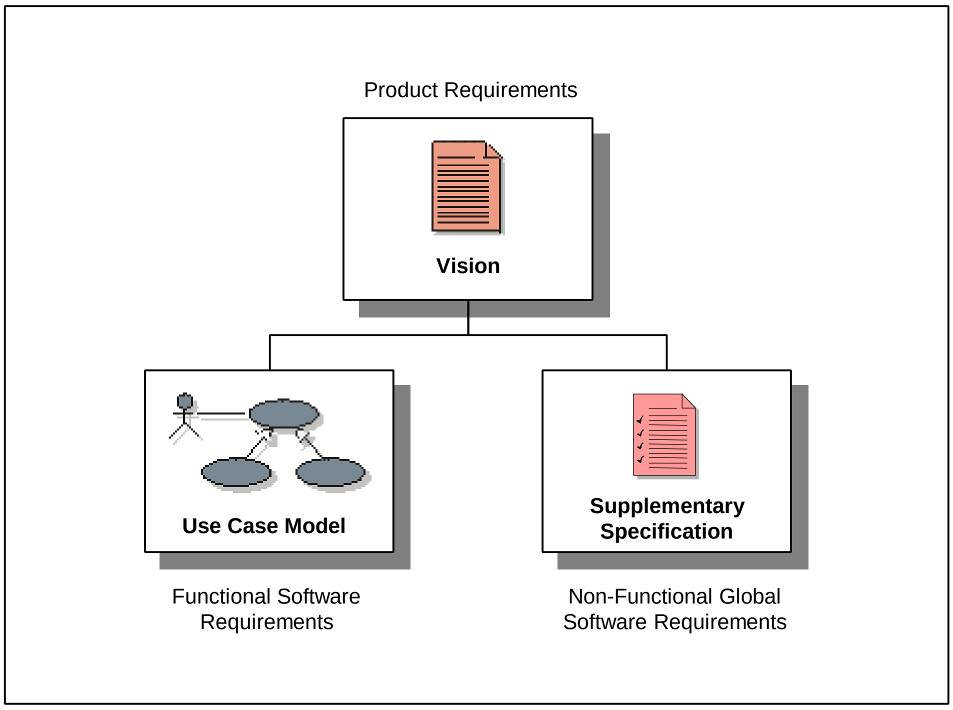

Product Requirements
Vision
Use Case Model
Supplementary Specification
Functional Software Requirements
Non-Functional Global Software Requirements

Figure 4: Allocation of Requirements of Different Types to Documents

Number of Requirement Types to Use

The number of requirement types used vary with different projects, the larger and more intricate the system, the more requirements types are needed. Moreover, different experts do not only name their requirement types differently, but they also suggest different numbers of requirement types. In the GREP Handbook [11] as many as thirteen different requirement types are recommended, while Rational experts recommends a maximum of five to seven requirement types. The benefit of having many requirements types is that they help verify that all aspects of the system are considered. The negative effect of having many requirement types is that too much effort is spent on assigning requirements to different types.

3.1.3 Requirement Attributes

Attributes are used to provide information that can be used to evaluate, track, prioritize and manage requirements. Each type of requirement has attributes, and each individual requirement has different attribute values [7, 16]. Attributes differ depending on the type of element that needs to be tracked, i.e., what questions they are intended to answer. Attribute values should be decided in the initial stage of the development. They should be able to answer questions that are pertinent to both stakeholders and developers. The Unified Process recommends using attributes to help in:

- Assigning resources
- Assessing status
- Calculating software metrics
- Managing project risk
- Estimating costs
- Assuring user safety
- Managing project scope

Some commonly used attributes are described here below in *Table 1*:

Name of Attribute	**Explanation**
Rationale	Reason for the requirement
Development Priority	Order/priority of development
Status	Proposed, approved, incorporated, validated, rejected
Risk	Probability of adverse project impact (schedule, budget, technical)
Safety/Criticality	Ability to affect user health, welfare, or economic consequence of failure
Responsible Party	Who is responsible for the requirement
Origin	Source of the requirement
Stability	Probability whether the requirement will change

Table 1: List of Common Attributes

3.1.4 Traceability

To manage requirements, traceability information is needed. A requirement is considered to be traceable if it is possible to discover [15]:

- Who suggested the requirement.
- Why the requirement exists.
- What requirements are related to it.
- How the requirement relates to other information.

What is Requirements Traceability?

Requirements traceability is a technique that is used to follow requirements from their origin, through development and ongoing iterations of refinement, to subsequent implementation and use, in intermediate as well as final products. Further, traceability provides the ability to discover the history of system features and supports maintenance of the system. Traceability should be a bi-directional path, with both forward and backward recordings [3, 4, 11, 15, 16]. Forward traceability is needed in order to demonstrate how a requirement is manifested in a system. It supports the software engineer in keeping track of the requirements, and ensures that all requirements are properly transformed at each level of development. Backward traceability is required to maintain the integrity of the requirements, when changes of the design and the environment occur. It can be used to determine which requirement was the origin of a certain piece of code.

Requirements traceability has been in practice for more than two decades [4], but still there is little consensus on how to apply it. The practices and usefulness of traceability vary considerably. Different stakeholders have different views of traceability, and standards of requirements traceability are too vague in their definitions.

Why Use Requirements Traceability?

The main stated purposes for establishing traceability are [10, 15]:

- To verify that all customer requirements are fulfilled in the implemented system.
- To verify that the application does what it was intended to do.
- To manage change.

Depending on which stakeholder is asked why he/she uses requirements traceability the answer varies. In a case study by Ramesh et al. [4], the upper management viewed the use of requirements traceability as a must for survival, essential to keep customers happy. The Project managers, who needed to trace requirements to a more detailed level, believed that traceability provided a means to show that they were in full control of the project. However, the most significant use of traceability was achieved by the system engineer, who through traceability could trace proposed changes down to the computer software units, thus identifying which entities would be affected by a change.

Different Levels of Traceability

Different standards mandate varying degrees of requirements traceability, therefore each specific situation must decide which level of traceability to use. In the strictest sense, to achieve complete traceability means that all individual customer requirements are traced to each related specification, test procedure, model element, and ultimately the source code. Never the less, it is not always pertinent to have complete traceability. It is often recommended that traceability should only be established between requirements. The main disadvantage of traceability is that it requires a considerable investment to set up and maintain. It is important to evaluate the costs and benefits, to establish the level of traceability that a project calls for. Requirements traceability is much more difficult to achieve the larger the system, but also more important the larger the system.

Establish Traceability Paths

Irrespective of the level of traceability chosen, it is important to use traceability consistently. According to The International Council on Systems Engineering [2] a consistent use of traceability should be able to answer the following questions:

- What is the impact of changing requirements?
- Where is a requirement implemented?
- Are all requirements allocated?
- What mission need is addressed by a requirement?
- Why is this requirement here?
- Is this requirement necessary?
- What design decisions affect the implementation of a requirement?
- Why is the design implemented in this way and what were the other alternatives?

- Is the implementation compliant with the requirements?
- What acceptance test will be used to verify a requirement?
- Is this design necessary?
- How do I interpret this requirement?
- Are we done?

Consistent use of traceability can be achieved by establishing traceability paths. The paths describe how the requirements are traced to different elements, e.g., other requirements, use cases, classes etc. *Figure 5* show an example of how traceability paths can be established.

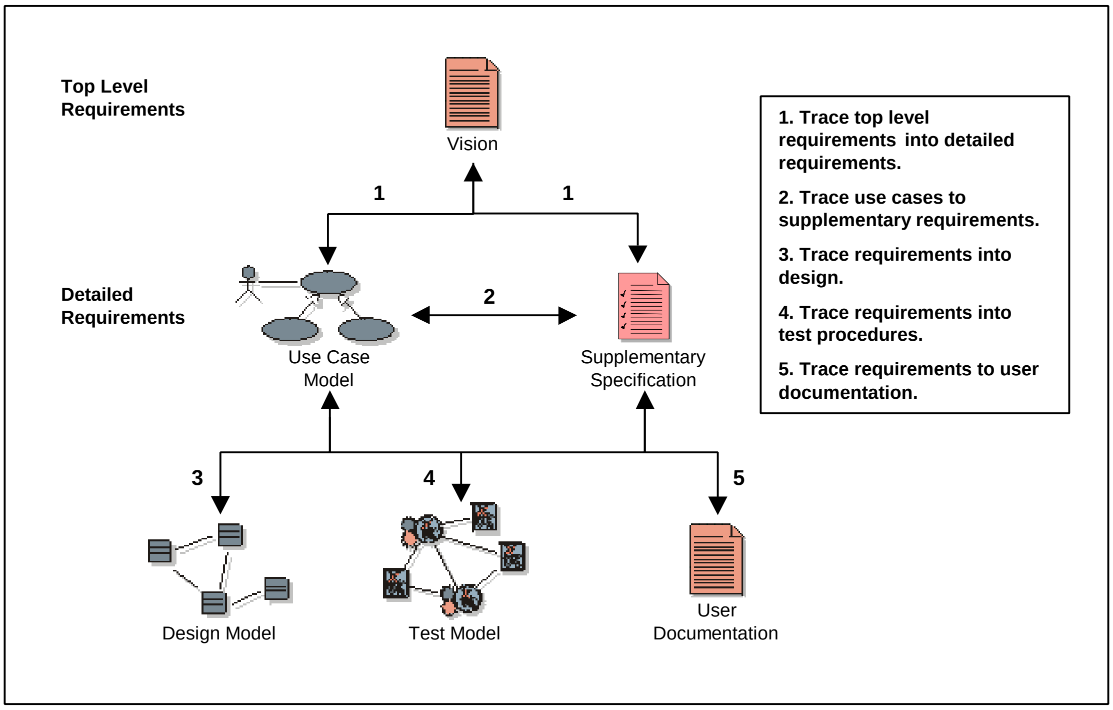

Top Level Requirements
Vision
1
1
Detailed Requirements
2
Use Case Model
Supplementary Specification
3
4
5
Design Model
Test Model
User Documentation
1. Trace top level requirements into detailed requirements.
2. Trace use cases to supplementary requirements.
3. Trace requirements into design.
4. Trace requirements into test procedures.
5. Trace requirements to user documentation.

Figure 5: Traceability Paths

3.1.5 Change History

As changes are made to a requirement, it is important that a record of all of the changes is maintained. There is a need to understand and record the history of requirements as they evolve because:

- Capturing reasons for change will avoid making the same mistakes over again.
- Change history enables developers to retreat to a certain version of the requirement, e.g., if a requirement proves impossible to implement.
- Find the reasons for why and how a requirement evolved.

In order to meet these demands the data collected in the change history record must be able to answer questions such as: What changed and when did it change, why did it change, and who authorized the change? An example of the kind of change data to maintain for the change history record is shown in *Table 2*, on page 13.

Version	Modifier	Date	Change	Reason
1.1	Donald Duck	98.05.01	Put up new fence	Pluto thrashed the old one
1.2	Mickey Mouse	98.06.02	Bought a Leash	Pluto don't obey the order: Stay Home!

Table 2: Modification History (Requirement: Keep Pluto out of Donald's Garden)

3.2 Tools for Requirements Management

Not very long time ago traceability was maintained with no more support than a paper and pen [2]. Then engineers started to use simple word processors and spread sheets to document the requirements and their dependencies. Even with the help of word processors and spreadsheets requirements management is a cumbersome task. Today the market is starting to realize the need of tools that supports traceability. Tools for requirements management, modeling (analysis and design), test and documentation, that can save effort and lighten the workload. Unfortunately there are few possibilities to relate the information stored within one tool to information stored in another tool. Developers know that it is near impossible to achieve traceability without tools, but the question is does the benefits justify the investment?

3.3 Traditional versus Iterative Approach to Requirements Management

In the traditional waterfall model requirements management is restricted to occur at the beginning of the development cycle. In the initial phase an attempt is made to precisely define all requirements to be implemented. This approach treats requirements as equally important and depends on that the requirements remain constant throughout the development life.

However, requirements do not remain constant, they change. As a matter of fact it is not always desirable that requirements remain constant. In the initial stage of a project the customer has a vision about the desired system. This vision functions as a base for the original set of requirements. As the project continues the original set changes: New requirements are created, and some requirements are modified or rejected.

It is now generally accepted that requirements must be treated as dynamic entities. They should not be confined to a certain stage of the development, but they should be maintained and refined during the entire life cycle. This dynamic view of requirements calls for a different approach to software development. Software systems need to be developed in an iterative manner that allows requirements to evolve with the system.

4 METHODS

This study was conducted during ten weeks as part of a thesis course at the Institute of Informatics, at the University of Gothenburg. Since time was one of the major limiting factors methods and tools had to be chosen with consideration to this, and the work had to be planned accordingly. At the initial stage of the study an outline of the overall planning was created. The workload of each week was planned in detail at the beginning of the week and summarized at the end of the week.

The process chosen was the Rational Unified Process, and the main tool used was Rational RequisitePro. Complimentary tools were supposed to be both Rational Rose and Rational SQA Suite. The choice of this process and the tools was due to the need of an evaluation. Another influencing factor was earlier familiarity with both the process and some of the tools, primarily RequisitePro. This reduced some of the time that had to be spent on learning.

4.1 Tools

This section contains a short presentation of the tools that were intended to be used in this investigation, i.e., RequisitePro, Rose and SQA Suite*. For a more complete description of the tools see *References* [9, 10, 12], on page 37.

4.1.1 RequisitePro

RequisitePro is a Windows-based tool that supports Requirements Management. Regardless of which process is used it helps organize, document and manage change. With RequisitePro it is possible to query, track, and trace requirements as they evolve throughout the project life cycle. Requirements are organized by linking Microsoft Word to an integrated database, which stores and manages the requirements.

RequisitePro is comprised of three different workplaces where interrelated work is done. These workplaces are the Tools Palette Workplace (*Figure 6*), the Views Workplace and the Word Workplace. When RequisitePro is started the, the Tools Palette Workplace, and the Views Workplace are displayed. The Tools Palette is used to work with requirements in documents, and to manipulate/modify projects, while the Views Workplace is an interface to the database. The Word Workplace is a window to Microsoft Word, which is not started until a document is created or opened.

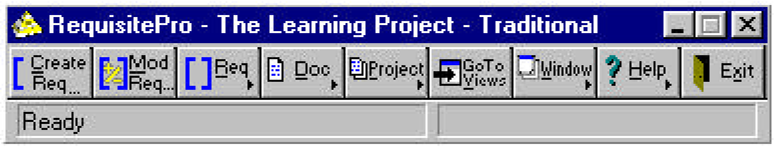


Figure 6: The RequisitePro Tools Palette

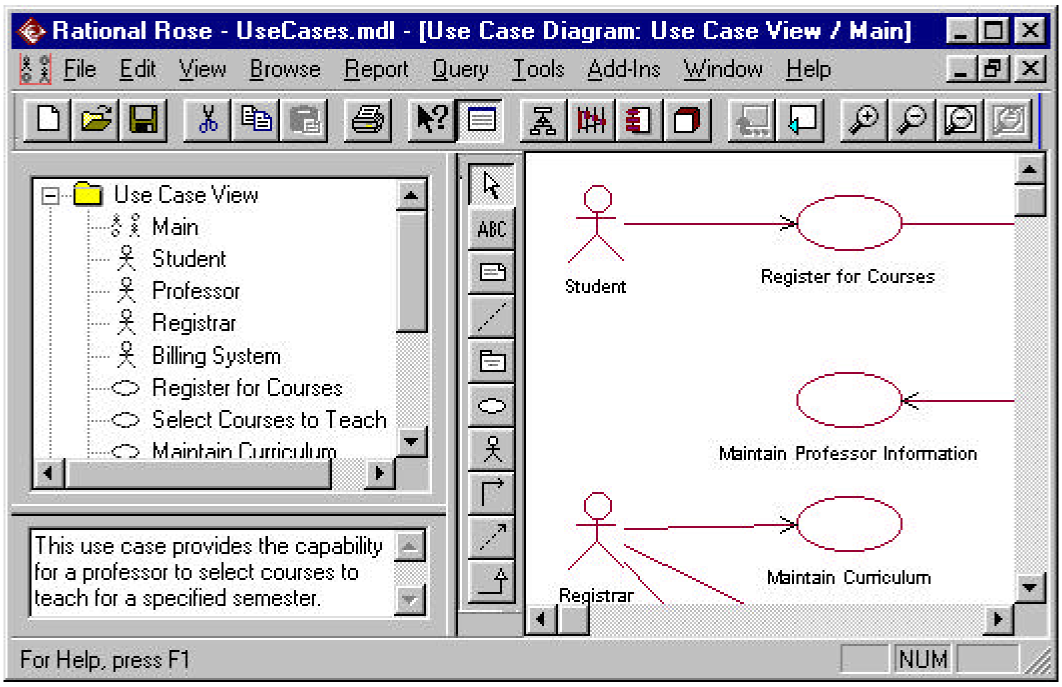
* The versions of the tools used were: RequisitePro 3.1, Rational Rose 98 (Enterprise Edition), while SQA Suite never was investigated, which is discussed later.

4.1.2 Rational Rose

Rational Rose is a visual modeling tool, which provides the capability to represent different perspectives of a system. In Rose, model components can be created, viewed, modified and manipulated. Models are abstractions of real world ideas and are used to show the essentials of complex problems. In Rose a model is a representation of the problem domain and the system software. Each model contains views, diagrams and specifications (detailed descriptions of specific entities), which are conveyed with the UML (Unified Modeling Language) notation.

The User Interface is Windows based and it consists of the following:

- A standard toolbar, which is independent of the diagram window currently open.
- A diagram toolbar, that can be customized differently for each view.
- A Browser, which provides the capability to textually view and navigate between components of the different views.
- A documentation window, where the documentation of the selected item can be entered or edited.
- A diagram window, where diagrams are created and manipulated.
- Documentation Window that shows a description of the selected item.



This use case provides the capability for a professor to select courses to teach for a specified semester.

Figure 7: Rational Rose with a Use Case Diagram, in the Use Case View

4.1.3 SQA Suite 6.0

SQA Suite is an integrated product suite for the automated testing of cross-Windows client/server applications.

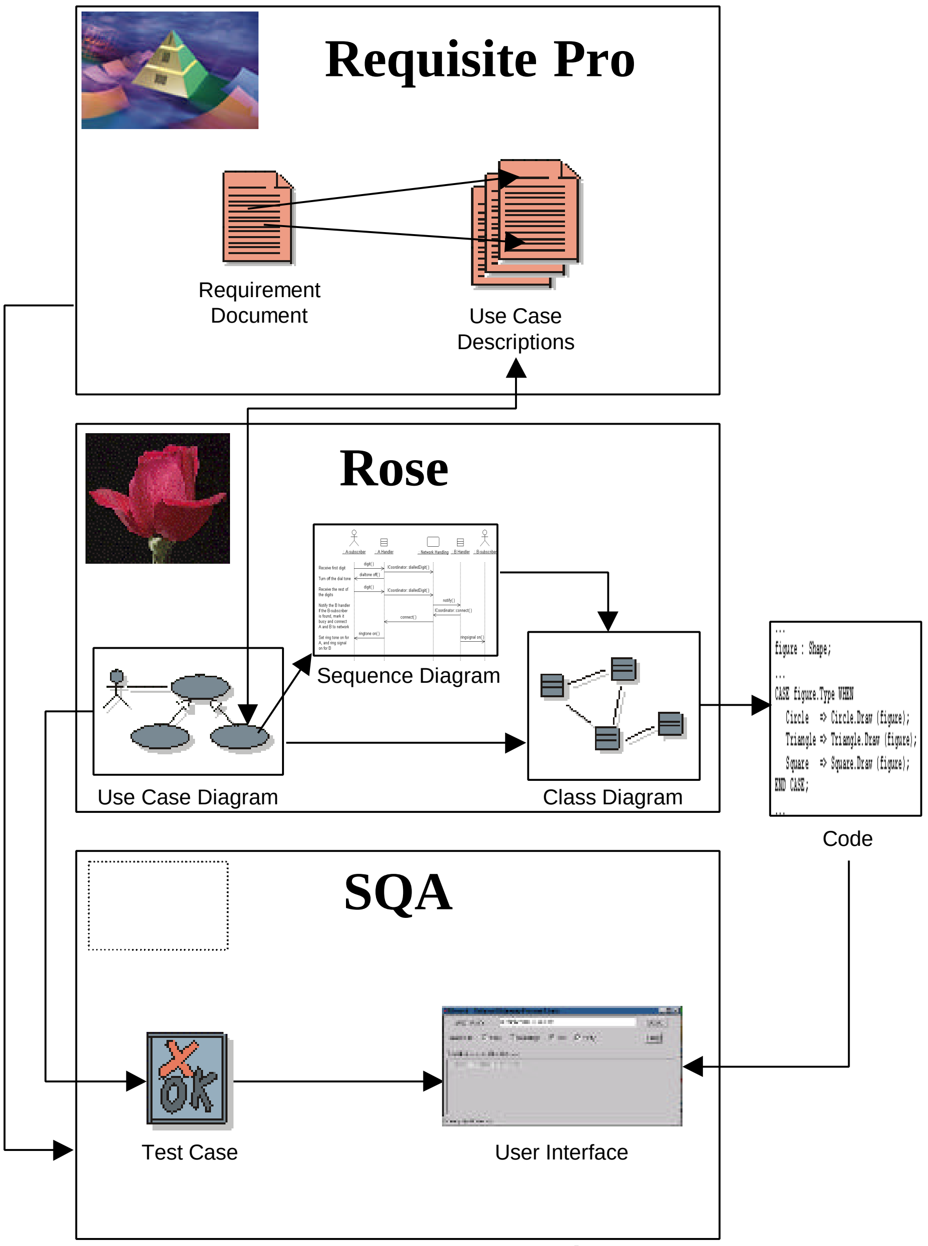
There are two versions of SQA Suite:

TeamTest Edition can be used to thoroughly test your code and determine if your software meets requirements and performs as expected.

LoadTest Edition provides integrated testing of structure, function, and performance of web-based applications.

4.1.4 Tools Integrated with RequisitePro

To enhance the requirements management capabilities RequisitePro can be integrated with other Rational tools, where Rose and SQA are two of them (see *Figure 8*). This makes it possible to exchange information between the different tools and thereby automate parts of the development process.



```
...
figure : Shape;

...
CASE figure.Type WHEN
   Circle   => Circle.Draw (figure);
   Triangle => Triangle.Draw (figure);
   Square   => Square.Draw (figure);
END CASE;

...
```

Figure 8: Integration Between RequisitePro, Rose and SQA

In RequisitePro requirements are traced to use cases, which can be synchronized with use cases in Rose. The use cases are refined to classes and objects. Code can be generated in Rose, and is then further elaborated by other means. When this code is tested the use cases are the foundation for the test cases. This can be accomplished through the synchronization between RequisitePro and SQA, or Rose and SQA.

4.2 Theoretical Studies

For the purpose of achieving both a comprehensive and complete insight to the concepts, features, and problems of Requirements Management: books, articles, reports and other documents where studied thoroughly. The Unified Process and the integrated tools, were treated similarly, but were supplemented with tools newsgroups on the Internet, and interviews and discussions with experts from the Rational corporation. In addition to this seminars were attended.

4.3 Case Study

As a foundation for this study the original requirements documentation of an existing system was used. The first step to take was to get fairly acquainted with the documentation. Then a project was created with the requirements management tool RequisitePro. The documentation was altered in order to treat the requirements according to the Unified Process. This meant that different document types and requirement types were used, and that use cases had to be defined. Further hierarchies and traceability links were established between the requirements.

With a synchronization wizard the use cases created in RequisitePro were generated in Rose. The next step consisted of creating different views and diagrams in Rose. The use cases served as starting points to create, first use case diagrams, and then sequence and collaboration diagrams (see *Visual Modeling with Rational Rose and UML* [12], for detailed descriptions of these diagram types). The use case descriptions and the sequence diagrams served as guidelines when class diagrams were created. The classes created in Rose were exported to RequisitePro in order to establish traceability links between requirements and classes. Then the synchronization capabilities were tested thoroughly by adding, deleting and modifying both requirements and items of the Rose model.

Finally the process and the tools were evaluated with respect to:

- Synchronization between the tools
- Support between process and tools
- Ease of use and facilitating capabilities
- Documentation and structuring capabilities
- Usability in requirements management

The original intention of this study was that RequisitePro also should be synchronized with SQA. However, the time that had to be spent on examining RequisitePro and Rose proved to be more extensive than estimated. Therefore the decision was made that a thorough research of RequisitePro and its integration with Rose was more purposeful than a cursory study that encompassed all tools.

4.3.1 Organization of Requirements in RequisitePro

A project was prepared in RequisitePro, which was performed as follows:

- Document types were chosen according to the Unified Process. The document types were Use Case Description, and Vision Document (Product Requirement Document).
- It was decided that the requirement types to use in the project were to be Use Case requirement type, Software requirement type and Product requirement type. Defining requirements types included defining their respective attribute sets.
- A project was created based on a RequisitePro project template, which was later adjusted to fit this particular study.

Vision

When the project outline had been created the original requirements document was imported to RequisitePro, and served as a Vision document. Two different requirement types were used: Product requirement types, and software requirement types. Further, the requirements were organized in hierarchies of appropriate levels.

Use Cases

Use cases were captured through examination of the original document and in cooperation with one of the developers of the system. The use cases were documented in RequisitePro in Use Case Description Documents, with one document for each use case. The use cases were then organized as hierarchical requirements in the following structure:

- The name of the use case served as the parent requirement.
- Basic flow of events, and alternative flow of events were given names in order to use them as the next level in the hierarchy.
- The details of the flow of events were then ordered into descending levels of hierarchies.

In each use case document influenced actors, and pre- and post conditions, were also documented.

4.3.2 Modeling in Rose

The use cases created in RequisitePro were generated in Rose through a synchronization wizard. Then diagrams were created and organized in different views.

Use Case View

In this view a use case diagram was constructed, to show the relationships between the use cases and actors that interacts with the system. For each use case specific sequence- and collaboration diagrams were elicited. The purpose of these diagrams was to achieve a more thorough understanding of the system.

Logical View

In this view class diagrams were created. The use case descriptions and the sequence diagrams were used to find the classes. Moreover, the sequence diagrams were used to find class operations and relationships between the classes.

Synchronization with RequisitePro

Through a synchronization wizard the classes were exported to the RequisitePro project.

4.3.3 Establishing and Testing Traceability Links in RequisitePro

In RequisitePro, traceability links were established between product requirements and use cases. Classes were linked to relevant use cases. Then changes were done to some requirements, and the requirements and classes affected by these changes were traced.

4.3.4 Testing Synchronization Capabilities

The synchronization capabilities between RequisitePro and Rose were tested several times by:

- Synchronizing a RequisitePro project with a blank Rose model, to export use cases, actors, and classes from the project to the model.
- Synchronizing a Rose model with a RequisitePro project, to export use cases, actors, and classes from the model to the project.
- Adding, removing and modifying use cases, actors, and classes in a RequisitePro project, which had already been synchronized with a Rose Model. Then the wizard was run to update the Rose Model with the new items in the RequisitePro project.
- Adding, removing and modifying use cases, actors, and classes in a Rose model, which had already been synchronized with a RequisitePro project. Then the wizard was run to update the RequisitePro project with the new items in the Rose model.
- Adding, removing and modifying use cases, actors, and classes in both a RequisitePro project and a Rose model, and then performing a bi-directional synchronization.

5 RESULTS

The results of this study showed that it is fairly simple to learn and use RequisitePro. The integration with Microsoft Word provides a familiar environment for requirements documentation of the system. The requirements can easily be manipulated and displayed in different views according to varying filtering criteria. Further, RequisitePro also contains wizards and templates for projects, and synchronization with other tools, e.g. Rose and SQA. Some of the problems perceived with RequisitePro were that creating requirements, hierarchies and traceability links was very time consuming, and that the synchronization with Rose was rather poor. Another problem of major concern was that the issue of documenting use cases in RequisitePro according to the Unified Process could not be solved with contentment.

5.1 RequisitePro

5.1.1 Project

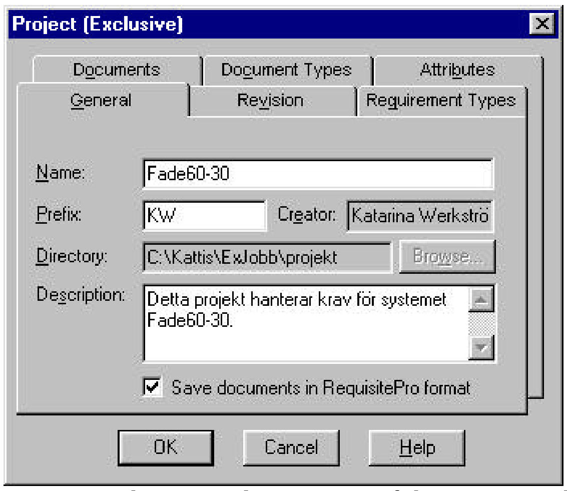
In the initial stage of this study a project was created in RequisitePro. There are three different approaches to create a project in RequisitePro:

1. Create a new project from scratch.
2. Use the project wizard.
3. Use an existing project as template.

All these approaches are quite easy to follow, but the simplest way to do it is to use a template provided by RequisitePro, and then gradually customize the template to fit the project in question. This was the approach used by the current study (see *Figure 9*). Some aspects of customizing the project were done with minimal effort, whereas other aspects proved to be unnecessarily complicated. The changes to a project template that can be performed with ease are:

- Modify, create, and delete requirement types.
- Modify, create, and delete requirement attributes.

The most complicated part appears when new document types are needed, which is described below, in *section 5.1.2*, on page 22.



Name: Fade60-30
Prefix: KW
Creator: Katarina Werkströ
Directory: C:\Kattis\ExJobb\projekt
Description: Detta projekt hanterar krav för systemet Fade60-30.
☑ Save documents in RequisitePro format

Figure 9: The General Properties of the Customized Project in RequisitePro

5.1.2 Documentation

Guidelines in the Unified Process

In the Unified Process there are Tool Mentors that describe how different Rational tools can be used in accordance with the process. Unfortunately there is no Tool Mentor for RequisitePro. This is a major problem since the guidelines[*] for documenting use cases, which are essential concepts of the Unified Process, were difficult to apply in RequisitePro. In other respects the guidelines proved to be useful to manage requirements.

The Unified Process recommends that requirements should be documented in a Stakeholder Needs document, a Vision document, a Use case model, and Supplementary specifications. The Stakeholder Needs document, is used to document the requests a stakeholder (customer, end user, marketing person, and so on) might have on the system to be developed. It also contains references to any type of external sources to which the system must comply. The Stakeholder Needs document was not used during this study, because no such document was available, and it was considered that an attempt of a reconstruction would not have been of any interest. The Vision document corresponds to the traditional Product Requirements Document. It contains a general vision of the core project's requirements, and provides the contractual basis for the more detailed technical requirements. The Use case model is a model of the system's intended functions and its environment, and serves as a contract between the customer and the developers. It consists of use cases, actors and their relationships. The use cases are documented both in textual descriptions (use case specification) and in a visual model consisting of different diagrams. The supplementary specifications capture the global non-functional requirements, which can not be captured by the use cases.

The benefits that were perceived by organizing requirements as described above were:

- Requirements are first described at a higher level in the Vision document, which gives an overview of the system. The requirements are then refined in more detail in the use case specification and the supplementary specification, which conveys more specific and concrete definitions of the end-users’ needs.
- The use case specifications made it easier to understand the system. The use case specifications also proved to be essential as a basis for more detailed modeling, e.g., they served as input to find classes and objects.

Problems with Documentation of Use Cases

In RequisitePro requirements can be documented either in documents or directly in the database. The former is recommended, since this puts context around the requirements. Both RequisitePro and the Unified Process have templates for different document types, including Vision, Use case, and Supplementary documents. Documenting requirements in RequisitePro

[*] Guidelines refer to how something is accomplished in general, whereas the Tool Mentors describes how the guidelines can be applied in specific tools.

according to the Unified Process is not always straightforward, since there are no direct guidelines for this purpose. The Vision document and Supplementary specification did not cause any major problems. The use case specifications, on the other hand, could not be documented in a satisfying manner. The origin of the problems can be defined as follows: Use cases describe different sequences performed by the system. These should be conveyed in words that the end user is comfortable with. A sequence consists of one or several functional requirements. If each individual requirement in a sequence is to be stored as a requirement in RequisitePro two dilemmas arise:

1. The document becomes difficult to read and understand, due to each requirement in RequisitePro is tagged with a unique identifier. Never the less, this can be remedied by hiding the identifiers.
2. One sentence may consist of several requirements, which in many cases makes the requirements incomprehensible when viewed separately.

A possible alternate solution to the problems mentioned above, is that sequences should not be decomposed. This is not an ultimate solution though, because it limits the advantages of RequisitePro. In this case a requirement is a composite of many individual requirements. This means that requirements included in a sequence cannot be given individual attribute values, e.g., priority and risk.

Import of Requirements

Sometimes requirements documented outside of RequisitePro need to be imported. If the requirements have been documented in Word they can easily be imported to RequisitePro. In the present study requirements documentation was both imported and written directly in RequisitePro. However, there is a problem with import of documents: The original document formats can not be successfully imported without certain preparations. If it is desired that the original document formats should be preserved the procedure is somewhat complicated. First a so-called outline, that contains the desired format, must be created, see *The RequisitePro Users' Guide* [9], for a detailed description of this. The next step is to add the outline to the document type that should be used for the new document. When this is done the requirements document can be imported with the original format applied.

5.1.3 Creating Requirements

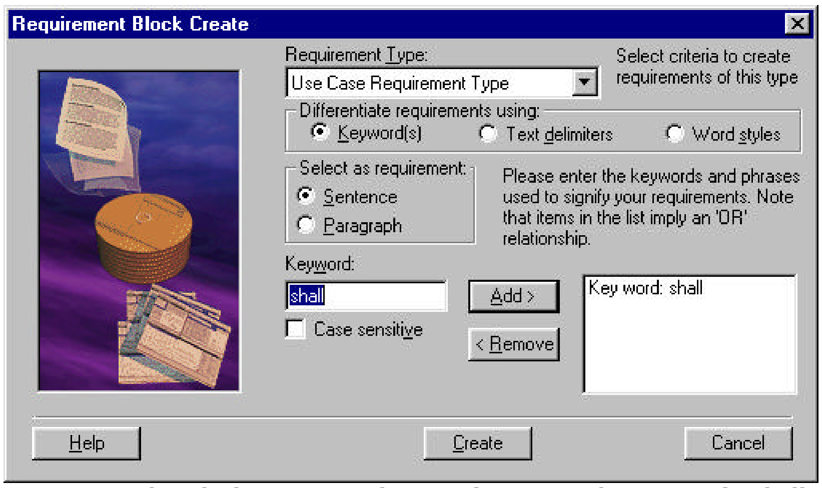
In RequisitePro the requirements can be created either from within a document, or directly in the database. During the case study all requirements, with one exception, were created in different documents. The exception was requirements of the type class, which were added to the database through a synchronization with Rose. When requirements are created in a document the RequisitePro Tool Palette is used. The procedure is very simple: Mark the text to be a requirement, and then click the **Create Req** button, or choose the menu alternative **Req => Create…** . However simple to perform, this procedure is very time consuming and tedious. The process of creating requirements is rather monotonous, and for each requirement that is created the process is slowed down. This in turn results in that the tool is much slower than the user. RequisitePro has a solution to this, which is to use the **Block Create** function.

The Block Create Function

With the Block Create function (see *Figure 10*) many requirements can be created simultaneously from a large selection of text. There are three different methods that can be used to differentiate between the requirements:

- Specific Keywords
- Text Delimiters
- Word Styles

Here it must be pointed out, that the Block Create function is most useful if the requirements document has been structured and/or written with consideration to the use of this function.


Requirement Block Create

Requirement Type: Use Case Requirement Type

Select criteria to create requirements of this type

Differentiate requirements using: [✓ Keyword(s)] Text delimiters Word styles

Select as requirement: [✓ Sentence] Paragraph

Please enter the keywords and phrases used to signify your requirements. Note that items in the list imply an 'OR' relationship.

Keyword: shall

Case sensitive

Key word: shall

Figure 10: The Block Create Dialog Window. Here the Keyword "shall" has been Used to Differentiate Between Requirements

The Block Create function was used in the Product Requirement Document of this study. The document had not been prepared in any manner to use this function, and therefor the keyword *shall* was chosen to differentiate between requirements. This was not an efficient way to use the Block Create function. The document needed thorough reviewing, because all requirements did not contain the word shall, and sometimes a single requirement contained more than one shall. Moreover, the appearance of the requirements in the document had to be adjusted. Another major drawback of the block create function was that only parent requirements of a hierarchy could be created, which will be discussed next.

Hierarchical Requirements

Hierarchical requirement relationships make it possible to decompose one requirement into several more specific requirements. In RequisitePro this is done by establishing a parent requirement, and creating one or many children of that parent. A child must always reside in the same location as its parent, either in a document or in the database.

The part perceived to be most difficult when hierarchies were established was the analyzing work. Before the hierarchies could be created in RequisitePro it was necessary to analyze what requirements should be parents, what children should those parents have, and how many levels of parent-child relationships should be used. The act of creating the parent-child relationship in

RequisitePro was easy to perform, but rather slow and monotonous, as when creating “regular” requirements. There was an obvious need for the block create function, but as already mentioned, this function is not applicable on hierarchical requirements.

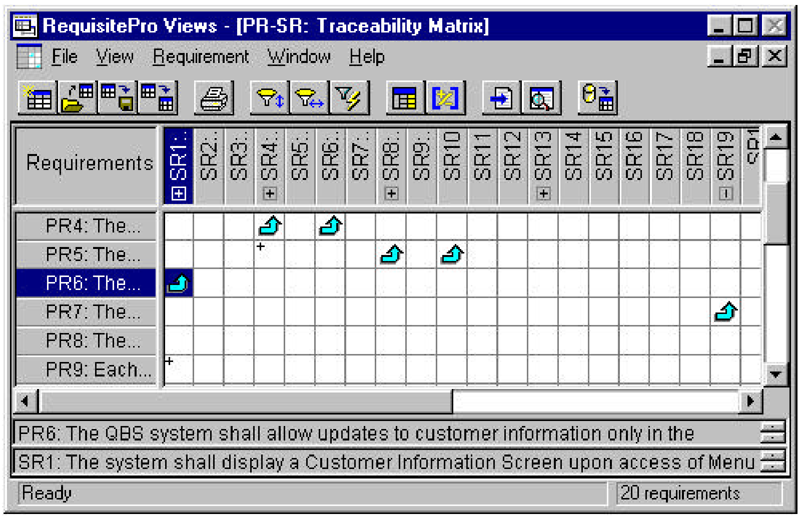
5.1.4 Guidelines for Managing Changing Requirements

Managing changing requirements include activities like determining which dependencies are important to trace, establishing traceability between related items, and keeping a change history record. Further, in the Unified Process, the artifact Requirements Attributes is essential to the management of changing requirements. This artifact defines and shows the status of a set of requirements attributes of each item that is being managed.

Traceability

There are several ways to create traceability relationships in RequisitePro. It can be done either in the Words workplace or in the Views workplace, see RequisitePro Users’ Guide [9] for a more thorough description of the possible ways to create traceability relationships. RequisitePro provides two types of links between requirements, a **Trace To** link and a **Trace From** link. The two types are provided for convenience, because some users think of decomposing high level requirements into lower level, whereas other users think of low level requirements fulfilling high level requirements. Within a project though, only one type of link should be used as it affects the way traceability trees are displayed.

During the present study, all different approaches to establishing traceability relationships in RequisitePro were considered. The approach that was found to be by far the most convenient was to work with traceability matrices in the Views workplace, see *Figure 11* below.



Requirements	SR1:..	SR2:..	SR3:..	SR4:..	SR5:..	SR6:..	SR7:..	SR8:..	SR9:..	SR10	SR11	SR12	SR13	SR14	SR15	SR16	SR17	SR18	SR19
PR4: The...				↗		↗													
PR5: The...				+				↗		↗									
PR6: The...	↗																		
PR7: The...																			↗
PR8: The...																			
PR9: Each...	+																		

PR6: The QBS system shall allow updates to customer information only in the

SR1: The system shall display a Customer Information Screen upon access of Menu

Ready | 20 requirements

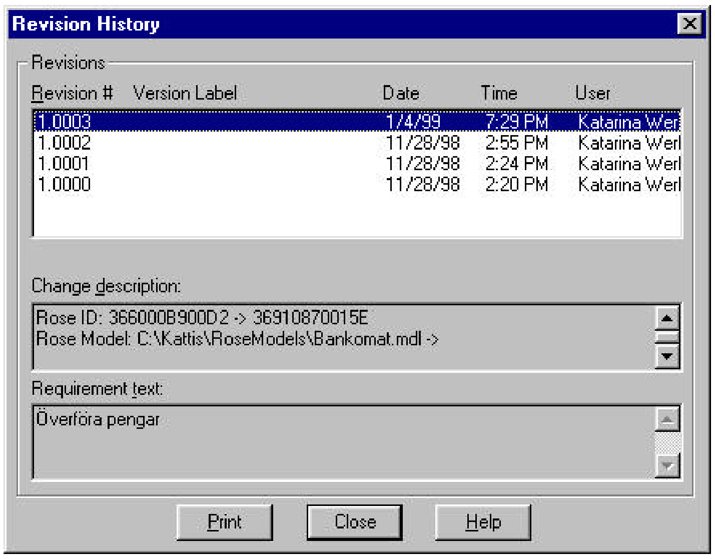
Figure 11: Traceability Matrix in RequisitePro

Each link is created by: Placing the cursor in the intersection between the requirements to be linked. Then clicking the right-hand mouse button to display a menu, from which either of the options Trace To or Trace From is chosen.

Creating traceability links is much faster than creating requirements, and it is a very elementary procedure. The difficulty about establishing traceability links is that it demands that thorough analyzing is conducted before the links are created in RequisitePro.

Change History

The tracking of requirement history is a significant concept in requirements management. By capturing the nature and logical basis of requirements changes, reviewers receive the information needed to respond to the change properly. In RequisitePro change history is kept on three different levels: On the entire project, on the documents and finally, on each individual requirement (see *Figure 12*). RequisitePro saves all of the prior revisions of these items as they are modified.



Revision History

Revisions

Revision #	Version Label	Date	Time	User
1.0003		1/4/99	7:29 PM	Katarina Wer
1.0002		11/28/98	2:55 PM	Katarina Werl
1.0001		11/28/98	2:24 PM	Katarina Werl
1.0000		11/28/98	2:20 PM	Katarina Werl

Change description:

Rose ID: 366000B900D2 -> 36910870015E
Rose Model: C:\Kattis\RoseModels\Bankomat.mdl ->

Requirement text:

Överföra pengar

Figure 12: The Change History Record of the Use Case Requirement "Överföra Pengar"

In the History record the following is stored:

- **Revision #:** Contains the Revision number of the project, document, or requirement.
- **Version Label:** User-defined text that describes the revision.
- **Date:** Contains the date created or last modified.
- **Time:** Contains the time created or last modified.
- **User:** Lists the person who last changed any of the project components.
- **Change description:** Contains the Change Description of the selected revision.

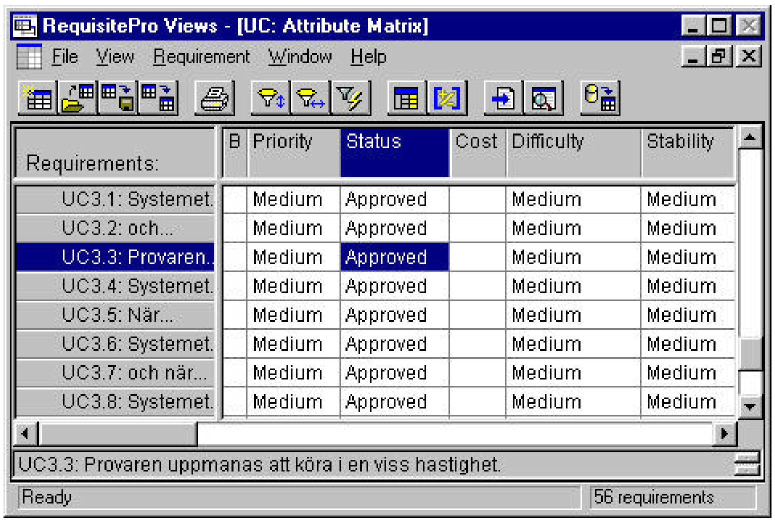
The History record was found to be useful when reviewing the evolution of requirements. A desired extension of this function would be the possibility to automatically revert to an older version of a requirement. Although, it should be mentioned that RequisitePro has a function for archiving the complete project, and is also integrated with tools that manage version control. However, neither the RequisitePro function, nor any function of the other tools, can manage the task of reverting an individual requirement.

Attributes

Requirements are given attributes to help keep track of their general status. Understanding attributes of the requirements help in managing the scope of the project and the application. Each project may come up with their specific set of attributes, and attributes may differ depending on the type of element to track. In the current study it was found that the RequisitePro ability to handle requirements attributes could be very purposeful. Subset of the requirements can be extracted by using queries on the attributes. The queries can be customized from simple queries to provide requirements that fulfill a certain attribute value, to more complex queries that provides requirements that fulfill several attributes values. Examples of different queries that were used:

- Find all high risk requirements
- Find all high risk requirements, that are assigned to a certain person
- Find all high-risk requirements that are assigned to a certain person, and were modified in a certain time interval.
- Etc. …

The results of the queries are displayed in attribute matrices in the Views Workplace, and can also be saved in RequisitePro. See figure below.


RequisitePro Views - [UC: Attribute Matrix]
File View Requirement Window Help

Requirements:	B	Priority	Status	Cost	Difficulty	Stability
UC3.1: Systemet.		Medium	Approved		Medium	Medium
UC3.2: och...		Medium	Approved		Medium	Medium
UC3.3: Provaren.		Medium	Approved		Medium	Medium
UC3.4: Systemet.		Medium	Approved		Medium	Medium
UC3.5: När...		Medium	Approved		Medium	Medium
UC3.6: Systemet.		Medium	Approved		Medium	Medium
UC3.7: och när...		Medium	Approved		Medium	Medium
UC3.8: Systemet.		Medium	Approved		Medium	Medium

UC3.3: Provaren uppmanas att köra i en viss hastighet.

Ready 56 requirements

Figure 13: The Attribute Matrix Showing Some Attribute Values of Use Cases

The Unified Process contains directions about how to construct attribute matrices without tool support. However, if a requirements management tool such as Requisite Pro is used, as in this case, it is suggested that the matrices are maintained directly in the requirements management tool.

5.2 Rose

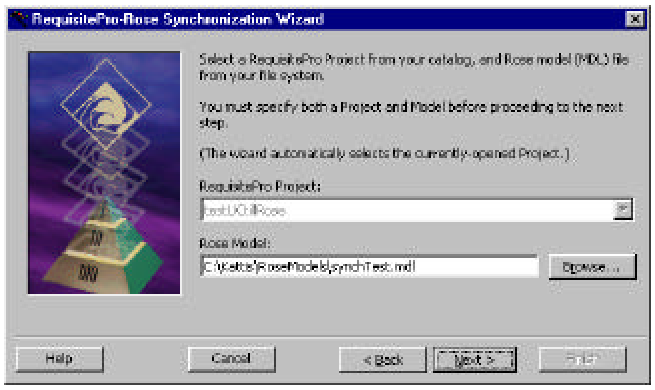
The Unified Process recommends that requirements should not only be described in textual descriptions, but they should also be depicted as visual models. For this purpose a visual modeling tool, like Rose, is desirable. The core artifacts of visual modeling according to the Unified Process, are use cases and a use-case model.

In the case study, as was described in *Methods*, on page 18, use cases were first documented in RequisitePro, and then synchronized with Rose. It is also possible to work the other way around, i.e., first create the use cases in Rose and then synchronize with RequisitePro, this is described further in section 5.3, on page 27. When the use cases were elaborated into classes and objects in different diagrams, it was made possible to indirectly trace product requirements to classes via use cases. This was accomplished due to traceability links between product requirements and use cases, and a function in Rose that allows the user to generate a list of all classes connected to a selected use case.

Changes to requirements naturally impact the models produced in the analysis and design models in Rose, as well as the resulting code. Traceability relationships between requirements and other development artifacts are the key to understanding these impacts.

5.3 Synchronization Between RequisitePro and Rose

Information can be exchanged between RequisitePro and Rose. A synchronization wizard (shown in *Figure 14*) executes this. The wizard can be started either from RequisitePro or Rose. The direction of the information exchange, i.e. from RequisitePro to Rose or the opposite, is independent of where the wizard was started. A RequisitePro project can only be synchronized with one Rose model at a time. The synchronization is based on the coupling between requirement types, and the Rose items Use Cases, Actors, Classes, and Packages.


RequisitePro-Rose Synchronization Wizard

Select a RequisitePro Project from your catalog, and Rose model (MDL) file from your file system.

You must specify both a Project and Model before proceeding to the next step.

(The wizard automatically selects the currently-opened Project.)

RequisitePro Project: testUCfillRose

Rose Model: C:\Kattis\RoseModels\synchTest.mdl

Figure 14: The RequisitePro-Rose Synchronization Wizard

The testing of the synchronization capabilities showed that the wizard's instructions could be followed without problems. The remaining results were more disappointing. They revealed that the issue about the synchronization is not user friendliness, but the haphazard results it produces. The following descriptions concludes the observed results:

Initial Synchronization

In the initial synchronization use Cases, actors, and classes were successfully exported, either from the RequisitePro project to the Rose model, or the opposite direction. No relationships, such as hierarchies, traceability links, and associations were possible to interchange.

The initial synchronization resulted in that new attributes were added automatically to the requirements type associated to the Rose items added. When a Rose item is associated with a requirement type, the wizard adds six attributes to each type: Rose Model, Rose ID, RoseItem Type, Rose

Documentation, Rose Ext Document, and Rose Sync. The new attributes of special interest were:

- **Rose Documentation** which represents the brief description.
- **Rose External Document** which contains external documents linked to the item.

Brief descriptions could not be exported in the initial synchronization from RequisitePro to Rose, but had to be transferred by the copy/paste function. However, if the initial synchronization had the direction Rose to RequisitePro, the brief descriptions were transferred.

Updating RequisitePro projects and Rose Models by Synchronization

When use cases, actors, and classes were first added or modified in either the RequisitePro project, or the Rose model, and then synchronized, the following was observed:

- Both the RequisitePro project, and the Rose model were correctly updated with any new use cases, actors, or classes.
- The values given to the attribute **Rose Documentation** in RequisitePro were conveyed as specifications in Rose.
- If any elements had been removed from either of the tools, they still remained in the other after the synchronization. Further there was no possibility to reintroduce the removed element.

6 DISCUSSION

The software crisis is a topic that has been debated for some years now. There is no possible way to deny the obvious facts, which have been reported by many different organizations, e.g., US Government Data from 1995 shows that less than 5 % of all software systems developed could be used without changes. The remaining 95 % either required changes or were unusable. Two major reasons given for this state are bad requirements management practices, and lack of a well defined process or method [6, 8, 12, 13, 16]. This is why requirements management is so important. But requirements management is not an easy task. It requires guidelines for extensive, structured analyses, and is almost impossible without the support of appropriate tools. What then is an appropriate tool? It is a tool that can enhance a process by unloading the burden of tedious and repetitious work that follow in the wake of requirements management.

6.1 Can Benefits of the Tool Justify the Investment?

Tools that are claimed to render more effective requirements management are emerging on the market. These tools supposedly assist in organizing, documenting, tracing and in other ways managing changing requirements. A tool that fulfills these capabilities can be invaluable. Never the less, managers need to know if the costs associated with incorporating the tool(s) can be justified by the resulting benefits.

An obvious property that all requirements management tools must have is that they should be easy to use. This is certainly a property of RequisitePro with its adaptation to Microsoft standards, which seem to dominate the software world of today. RequisitePro is also excellent to manipulate requirements: With only a couple of simple keystrokes it is possible to track requirements with specified properties and/or requirements affected by a change. However, in order to make use of those excellent capabilities thorough preparations must be done. First requirements analyses must be conducted to decide what requirement types to use, what requirements attributes are needed, what requirement hierarchies should be set up, and what traceability paths/links should be established? This activity can be supported by a tool, but it can not be solved by a tool. It calls for the knowledge of an experienced requirements engineer.

The second part of the preparations is that the analyzing work has to be realized by the tool by introducing the requirements, assigning them attributes, and creating requirements relationships, in the form of hierarchies and traceability links. As already reported in the Results, this work was both time consuming and monotonous. Therefore it is highly recommended that the Block Create function for creating requirements (see section *5.1.3 The Block Create Function*, on page 23), be considered during analysis. This will not only save a lot of time, but it also relieves people from a very dull task. Still there are no similar possibilities to facilitate creation of hierarchies and traceability links.

Now, the question that follows is: Can the expenses and efforts be justified by the benefits of the tool? The answer is, it depends. It mainly depends on the size of the system. Or as Sommerville and Sawyer [15] expresses it:

System size makes a tremendous difference. The problems of requirements engineering increase exponentially with the size of the system.

This means that the larger the system the greater the need for tool support to assist in requirements management. If the intended system is of as slight proportions that an iterative approach will not be needed, it is probably not very purposeful to use RequisitePro, or any other requirements management tool. Never the less, a small system may one day be further developed into something more extensive. If the system has already been prepared in RequisitePro this will make the evolution much smoother.

6.2 Combining a Process with Tool Support

Most faults found during testing and operations results from poor understanding or misinterpretation of requirements. The later the faults are discovered the more expensive they are to correct. It costs 100 times more to correct faults found after delivery than in analysis [13, 14, 15, 16]. This is one of the reasons to use an iterative approach, which provides for early elimination of high risks associated with requirements. As already been reflected upon, the fundamental problems of requirements management can not be solved by a tool, but by experience and analysis. These problems call for the assistance of a process that provides “best practices” guidelines.

Before starting developing the system it is necessary to decide which process to use: should it be a process like the Unified Process that deals with requirements as use cases, or a process that deals with requirements in a more traditional manner? It must further be taken into consideration which tools can support the process: Are there any tools that are already available in the organization that can be used, or must new tools be purchased, and if so can the new tools be combined with existing tools?

The combination of the Unified Process with RequisitePro has many advantages. The process deals with requirements as dynamic entities that change over time, which is handled in RequisitePro by such capabilities as traceability and change history. Never the less, there are some difficulties that are important to be aware of, which are discrepancies between the process and RequisitePro concerning use cases. These problems are delivered in detail in *section 5.1.2*, on page 21. The origin of the problems is that the Unified Process does not give any useful guidelines for documenting use cases with RequisitePro. Some of the questions brought forward by this are: What should be requirements in use cases? The name, the flow of events, details in the flow of events…? How can use cases be stored as comprehensive requirements in RequisitePro? A reason for this lack of instructions on use case documentation may be that RequisitePro was originally developed to work with traditionally managed requirements. Further, RequisitePro has only relatively recently (April 1997) been incorporated with the Rational products, and hopefully the use case documentation problem will be dealt with in future revisions of both the process and the tool.

6.3 Benefits and Drawbacks with Synchronization

In the process of developing a software system several tools are used to support the different activities performed. This places a demand on the possibility to transfer data from one tool to another. RequisitePro can exchange data with several tools, but the topic discussed here only concerns the synchronization with Rose.

Use cases, classes and actors can be transferred between the tools RequisitePro and Rose, but how useful is it really? The time saved by synchronization was almost insignificant. The possibilities to update inconsistencies between the Rose model and RequisitePro project, were diminished by the fact that once an item in Rose or a requirement in RequisitePro had been deleted it could not be reintroduced through a synchronization (see section *5.3 Synchronization Between RequisitePro and Rose,* on page 27). The problem here is that elements of the Rose model and elements of the RequisitePro project are stored in different places. If corresponding elements could be stored in one place only, many problems could be solved as, e.g., inconsistencies between project and model.

However, there was one thing that was found to be of particular interest. This was the property that supplied indirect traceability down to classes (*section 5.2,* on page 26). This property can be an important substitute to creating traceability links to classes in RequisitePro. It will not provide a link from a higher requirement to one individual class, but a link to all classes implementing a use case connected to a specified higher requirement.

Finally, a recommendation that should be brought forward is that use cases first be modeled in Rose, and then synchronized with RequisitePro. This is a more natural workflow when developing according to the Unified Process. Besides this, it enables the brief descriptions of the use cases to be transferred from the beginning (see *section 5.3 Synchronization Between RequisitePro and Rose,* on page 27).

6.4 Conclusions

The results of this study confirmed that if a specific process is combined with the appropriate tool it is possible to facilitate requirements management, i.e.: the combination of the Unified Process with RequisitePro provides a means to facilitate requirements management. This study showed that the requirements management tool RequisitePro can be very useful to gain active control of requirements, but also that there are activities that could be performed smoother, and that knowledge of specific behavior and properties of the tool is needed to obtain full benefits of the tool. However, in order to draw any general conclusions about requirements management tools, further research with other tools and processes need to be done. A comparative evaluation of different tools would be highly interesting.

Today there is a demand for a much higher degree of automation and information interchange between tools, than currently exist. Are there any possibilities to improve the existing tools and the integration between them?

Developers of software systems must be able to integrate requirements management tools with both the chosen process, and tools used for other development activities.

Finally, to conclude this discussion: The tool is not the solution to requirements management, it is only an assistant to the developers. The solution to requirements management is found in the guidelines of a process, and the professional experience and knowledge of a skilled requirements engineer.

7 ACKNOWLEDGMENTS

I would like to thank Claes Lundquist and Marcus Olofsson, at Frontec in Gothenburg, for helping me review requirements documentation, and supplying useful advice. Claes has also been very helpful explaining the details of the system to which the original requirements documentation referred.

I also appreciated the help and advice that I got from my supervisor, Bengt Hagebratt, at the Institute of Informatics, University of Gothenburg.

THANK YOU!

8 GLOSSARY

Actor
An actor is someone or something, outside the system that interacts with the system. An actor is anything that exchanges data with the system. An actor can be a user, external hardware, or another system. The difference between an actor and an individual system user is that an actor represents a particular class of user rather than an actual user. Several users can play the same role, which means they can be one and the same actor. In that case, each user constitutes an instance of the actor.

Artifact
Artifacts are the work products of the process, the things that are produced or used, etc. during a project. An artifact can be any of the following: a document, a model, a model element. 'Artifact' is the term used in the Unified Process. Other processes use terms such as 'work product', 'work unit', etc., meaning the same thing.

Brief Description
The brief description of the use case should reflect its role and purpose, and the actors involved in the use case are referred to.

Collaboration Diagram
A collaboration diagram describes a pattern of interaction among objects; it shows the objects participating in the interaction by their links to each other and the messages they send to each other. Collaboration diagrams are used to show how objects interact to perform the behavior of a particular use case, or a part of a use case.

Component
A non-trivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture. A component conforms to and provides the physical realization of a set of interfaces.

Flow of Events
The Flow of Events of a use case contains the most important information derived from use-case modeling work. It should describe the use case's flow of events clearly enough for an outsider to easily understand it. The flow of events should present what the system does, not how the system is designed to perform the required behavior.

Iteration
A distinct sequence of activities with a base-lined plan and valuation criteria resulting in a release (internal or external).

Notation
A notation is defined by symbols that represents all the entities used to model a system, such as classes, objects and their relations. It is used to document the models that are the deliverables of analysis and design. A notation should be able to convey the thoughts of the model builder to a simple and consistent picture, that can be used as a communications means between the developers, customers and other involved parties.

Package
A grouping of modeling elements.

Sequence Diagram
A sequence diagram describes a pattern of interaction among objects, arranged in a chronological order; it shows the objects participating in the interaction by their "lifelines" and the messages that they send to each other.

Software Architecture
Software architecture encompasses:

- the significant decisions about the organization of a software system,
- the selection of the structural elements and their interfaces by which the system is composed together with their behavior as specified in the collaboration among those elements,
- the composition of the structural and behavioral elements into progressively larger subsystems,
- the architectural style that guides this organization, these elements and their interfaces, their collaborations, and their composition.

Stakeholder
An individual who is materially affected by the outcome of the system.

Stakeholder need
The business or operational problem (opportunity) that must be fulfilled in order to justify purchase or use.

UML (The Unified Modeling Language)
The UML is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software intensive system. The UML provides a standard way to write a system's blueprints, covering conceptual things, such as business processes and system functions, as well as concrete things, such as classes written in a specific programming language, database schemas, and reusable software components.

Use Case
A use case defines a set of use-case instances, where each instance is a sequence of actions a system performs that yields an observable result of value to a particular actor. A use-case class contains all main, alternate flows of events related to producing the 'observable result of value'. Technically, a use-case is a class whose instances are scenarios.

Use-case Diagram
A use-case diagram shows actors, use cases, use-case packages, and their relationships.

Use-case Model
A model of what the system is supposed to do and the system environment.

Use-case Package
A use-case package is a collection of use cases, actors, relationships, diagrams, and other packages; it is used to structure the use-case model by dividing it into smaller parts.

Worker
A worker defines the behavior (i.e. activities) and responsibilities (for artifacts) of an individual, or a set of individuals working together as a team. Each worker has a set of cohesive activities associated with it. "Cohesive" in this sense means those activities best performed by one individual. The responsibilities of each worker are usually defined relative to certain artifacts.

9 REFERENCES

Internet

[1] Kar, Pradip and Bailey, Michelle. *Characteristics of Good Requirements*. Requirements Working Group Information Paper. 1996. http://www.incose.org/workgrps/rwg/goodreqs.html

[2] *Requirements Management Technology Overview*. Excerpt from a report dated 24-June-1994, titled: Analysis of Automated Requirements Management Capabilities. INCOSE (International Council On Systems Engineering). 1994. http://www.incose.org/workgrps/tools/reqsmgmt.html

[3] Integrated Requirements Traceability (IRT) http://www.iconixsw.com/Spec_Sheets/IntReqtsTrace.html

[4] Ramesh, Bala, Stubbs, Curtis Lt., Powers, Timothy, and Edwards, Michael. *Lessons Learned from Implementing Requirements Traceability*. 1995. http://www.stsc.hill.af.mil/crosstalk/1995/apr/lessons.html

[5] Morris, Philip., Masera, Marcelo., and Wilikens, Marc. *Industrial Workshop on requirements Engineering*. November 1995. http://www.docaware.it/sta/dsa/report.htm

[6] *Chaos*. The Standish Group International, Incorporated. http://www.standishgroup.com

[7] Ericsson, Maria., Oberg, Roger., and Probasco, Leslee. *Applying Requimrents Management with UseCases*. Technical Paper TP505. Rational Software Corporation. 1998. http://www.rational.com

[8] Al-Sadoon, Omar. *Introduction to AURA – Automated User Requirements Acquisition*. http://www-users.cs.umn.edu/~fzhu/research/aura-intro.html

Manuals and User Guides

[9] *RequisitePro User's Guide*. Rational Software Corporation. December 1997.

[10] *Rational Unified Process*. Rational Software Corporation. November 1998.

[11] *GREP Handbook – Generic Requirements Engineering Process*. TSE. Ericsson. 1998

Books

[12] Quatrani, Terry. *Visual Modeling with Rational Rose and UML*. Addison-Wesley. 1997.

[13] Royce, Walker. *Software Project Management – A Unified Framework*. Addison-Wesley. 1998.

[14] Philippe Kruchten. *The Rational Unified Process: An Introduction*. 1998.

[15] Sommerville, Ian and Sawyer, Pete. *Requirements Engineering – A Good Practice Guide*. 1997.

[16] Pohl, Klaus. *Process Centered Requirements Engineering*. 1996.

[17] Eriksson, Hans-Erik., and Penker, Magnus. *Objektorientering – handbok och lexikon*. Studentlitteratur. 1996.