Lappeenranta University of Technology
Department of Information Technology

Achieving Traceability

The topic of the master's thesis has been accepted June 14, 2000 by the department council meeting of the Department of Information Technology.
The supervisor is prof. Heikki Kälviäinen.
The practical supervisor is M.Sc. Eng. Matti Ärmänen.

Helsinki, September 24, 2000

Ari Kujala
Hämeentie 27 c 69
00500 Helsinki
Phone: 040-7552651
e-mail: kujala@lut.fi

ABSTRACT

Lappeenranta University of Technology
Department of Information Technology
Ari Kujala
Achieving Traceability
Master's Thesis
2000
67 pages, 21 figures, 3 tables
Supervisor: Professor Heikki Kälviäinen

Keywords:
Requirement, requirement engineering, software engineering, system engineering, traceability, user requirement, system requirement, requirement analysis.

This master's thesis discusses requirement analysis and focuses on the problem of traceability. Requirement analysis is a part of software engineering which is often neglected someway. Engineers do know that to analyse the problem is a key to understand it. This thesis discusses requirement analysis as a part of system engineering process. Ways to present requirements are presented. The nature of traceability is discussed and some conclusions are drawn to document and propose additions to Sonera's current practise in the Mobile Payment Platform project. As result thesis will present a process model for requirement analysis, a structure for requirement collection and some points for traceability manual.

TIIVISTELMÄ

Lappeenrannan teknillinen korkeakoulu
Tietotekniikan osasto
Ari Kujala
Achieving Traceability
Diplomityö
2000
67 sivua ja 21 kuvaa ja 3 taulukkoa.
Tarkastaja: Professori Heikki Kälviäinen

Hakusanat:
vaatimusmäärittely, ohjelmistokehitys, systeemikehitys, ohjelmistotekniikka, jäljitettävyys, käyttäjävaatimus, systeemivaatimus, vaatimusanalyysi,

Keywords:
requirement, requirement engineering, software engineering, system engineering, traceability, user requirement, system requirement, requirement analysis, system engineering

Tämä diplomityö käsittelee vaatimusmäärittelyä. Erityinen keskittymisalue on vaatimusten jäljitettävyys. Vaatimusmäärittely on osa ohjelmistokehitysprosessia. Insinöörit tietävät, että ymmärtääkseen ongelmaa on sen lähtökohdat ymmärrettävä. Tästä huolimatta määrittelyvaihe epähuomioidaan helposti. Diplomityössä kartoitetaan ensin vaatimusmäärittelyä järjestelmäprojektin osana. Vaatimusmäärittelyn rakennetta tarkennetaan ja sen sisältöä tuodaan esille. Olemassaolevana projektina analysoidaan, kuinka Soneran Mobile Pay osaston suorittama vaatimusmäärittely on toteutunut Mobile Payment Platform projektin alkuvaiheessa. Lähinnä keskitytään näyttämään, kuinka vaatimukset on kirjattu ylös. Tämän jälkeen tarkastellaan jäljitettävyyden olemusta. Työssä kartoitetaan lukijalle, mitä jäljitettävyys tarkoittaa. Kartoituksen jälkeen käydään läpi jäljitettävyyttä tukevia toimenpiteitä Sonera Mobile Payn tuotekehitysprosessissa.

Työn tuloksena on esitetty prosessimalli vaatimusten keräämiseksi, malli vaatimusdokumentille sekä ohjeita jäljitettävyyden luomiseksi.

Contents

List of symbols and abbrevations

MPP	Mobile Payment Platform
SMP	Sonera Mobile Pay
R&D	Research and Design
IVR	Interactive Voice Response, connects the computer to the phone network.
KISS	Keep It Stupid Simple. A non scientific "common sense" method to make things work.

1 INTRODUCTION

This thesis was started in January 2000 at Sonera Mobile Pay in Helsinki. SMP (Sonera Mobile Pay) was formed to design Mobile Payment Platform (MPP). MPP is a system which can handle mobile payment transactions and fix them up with implemented services such as vending machines or car wash. MPP contains the main platform with system logic, datalink interfaces for service applications to join and a billing interface for billing the use of service applications.

The aim of the work is to generate a process model for requirement collection and to study how traceability is connected to the requirements. This task was approached by steps. The first step was to participate in SMP's work to observe and take part in requirement collection. During this phase reference material was collected. After studying how the requirements were collected it was possible to document the working practises and compare them to the theories and practises found from the book material. After studying already existing working practises, it was time to study how these practises could be improved.

Chapters two and three contain the theoretical study for process models. General system engineering process is discussed and a requirement analysis phase is taken out from it with greater detail. Chapter four contains observations made during the requirement engineering. It discusses how the work was done and what kind of decisions were made. Chapter five concentrates on studying what traceability means. Chapter six contains the main results from this work such as requirement process improvements.

2 SYSTEMS ENGINEERING

Software engineering aims to produce a software application. When a project is launched to create an application or enhance an existing one, systematization is needed. The reason for this is that a single entity cannot be handled if it has grown too large. It has to be broken down into parts and each part has to be dealt with separately. These parts can be arranged into phases etc. but the principal idea is to create a systematic procedure which takes care of all the necessary functions to create a workable solution to our problem. A software project can be managed as a process if it is systematic enough.

2.1 FUNDAMENTAL STEPS IN SYSTEMS ENGINEERING

The first step in creating a new venture is to consider what is wanted. This step is the requirement phase of a software project. Only after knowing what to do, solutions can be chosen and optimised to meet those needs. They can be implemented and tested against needs (Figure 1). Consider the whole problem before jumping into solutions. This sounds simple and obvious but it is not.

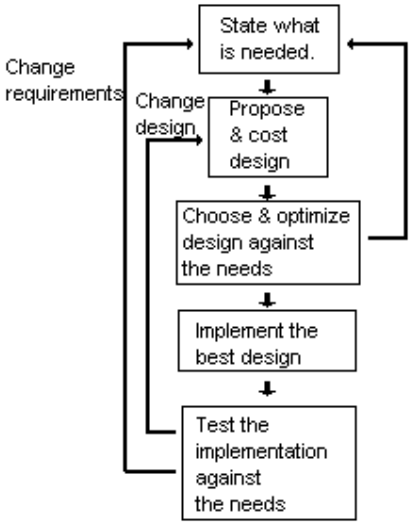

Figure 1. The essence of system engineering / 5, page 345 /

Figure 1 presents the structure of systems engineering. This information has been familiar to the ancient people as well as to us modern people. The Japanese samurai Miyamoto Musashi told in his book of Five Rings to us think of our daily tasks like crossing an ocean. / 3, page 81 / This was a requirement. We need to process our needs and find out simple tasks what to do. Before that we can do nothing but wonder about all the possibilities.

The next step is to find a suitable place for crossing. We do not want to start a voyage which would have too much distance or which would prohibit our voyage by some other obstacle. This compares to the second task of systems engineering where we propose the design and consider the costs.

After we have found a good place to cross, we prepare our ship and wait for the right weather. Now there is a need to choose and optimize the design of our plan against our needs. In this phase it is still possible to change our mind and not to cross. If we choose not to cross, we must go back to start and consider some other possibilities. If we want to cross, we must follow the next step.

When the wind seems right, we will set our sail and start our voyage. This is the implementation of our plan. Hopefully we have made right decisions, when we were still planning the whole project, because now we have to live with our

choices. Every time everything does not go like planned. So there might be a need to review our plans.

If the wind suddenly changes when we are on our voyage and we still have a couple miles to cross. We must take our oars and row the rest of the journey. This compares to changing our plan a little and taking the project to the end with an altered design. It is good to have options if something fails in our planning. There is also a possibility that we notice the wind changing. If we are just about to begin our journey we may need to come back and review our plan. We might also be on journey when we notice that some other great mistake has been made and we are forced to come all the way back to the start. There will also be a followup for future ventures because experience from old projects should be carried over to the new ones wherever possible. So the foundations of all our ventures are based on the simple question what do we want to do?

In this example we can see how important the vision to do something is. It is the spring which starts a stream of creativity. If we would know the exact requirement to “cross the ocean”, which would satisfy our need, we would have a possibility to complete this task. It would not be easy though but we would have a goal in sight and that would give us strength to proceed. If we want to do something and if we can see the way to do it, we can do it. By studying our way hard we might achieve many great things.

2.2 GENERAL SONERA LTD R&D PROCESS MODEL

At Sonera there is a research and design process model, called Sonera R&D Process Model / 6 /, which consists of three main parts. These parts are a pre-study, a feasibility study and project execution. In the Sonera model the pre-study is a phase where it is stated what is needed to be done to execute the project. In a feasibility study phase the requirements are collected. Based on requirements design and cost proposal are made. If the cost and design proposals are all right, the project to implement the design is launched. This project will contain in itself three phases which are encapsuled into the same frame. These parts are to make a design out of proposed design and requirements, to implement the design and to pilot the outcome. After piloting, a new release is ready to be launched.

The Sonera model also contains decision points and milestones. Decision points are external control points and there are six of them. They form the main barometer showing how far the process is going. Milestones are project specific control points. The project groups are responsible for defining them. To guide the process through these points, a review is called and a decision is made if the project is ready to proceed to the next step.

2.2.1 DECISION POINTS

The use of decision points provides structure and decision-making routines into projects and studies. At each decision point there is a meeting where the use of projects resources and their costs and benefits are reviewed. If the project seems worthy the decision to continue is issued. Other possible outcomes can be suspending it, cancelling it or continuing the previous phase with further study.

The decision points are as follows:

DP0 – decision to start pre-study

DP1 – decision to start a feasibility study

DP2 – decision to execute a project

DP3 – decision to implement a design

DP4 – decision to pilot the outcome of the project

DP5 – decision to conclude the project and release the final outcome.

2.2.2 EFFECTS ON MASTER'S THESIS

The sonera R&D model will have an effect on the thesis because the work is done in the context of the Sonera R&D environment. To show that the Sonera model fits in with the system engineering model in Section 2.1, and with the main idea used later in the work, there is a comparision of these two at a general level in Figure 2. Both models are displayed side to side in the same figure. Similarity of these models is well shown. If there would have been great conflicts, the model of system engineering should have been considered again with this work.

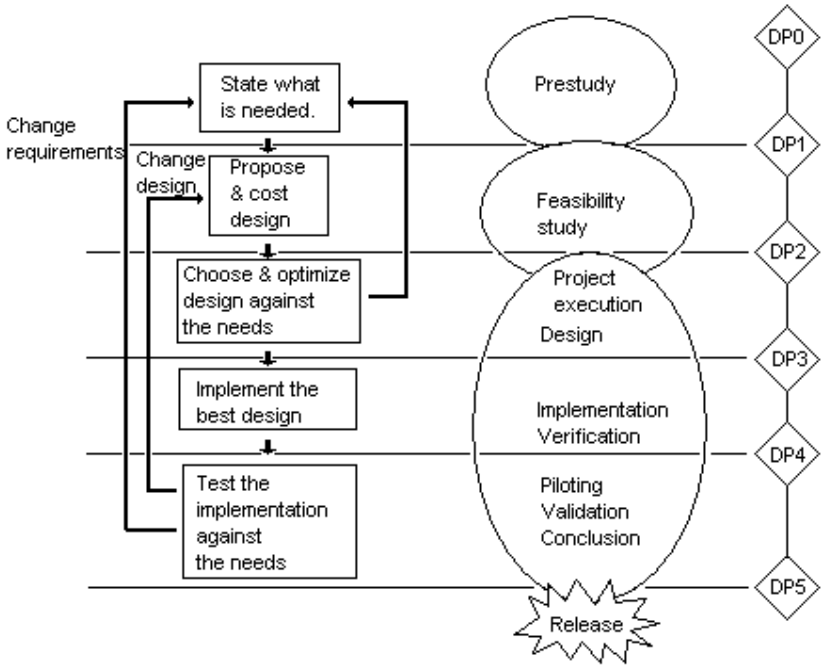

Figure 2. Comparing Sonera Ltd R&D model with essence of system engineering.

3 REQUIREMENT ANALYSIS

In Section 2.1 we had the task of crossing the ocean. The first phase was the need to do something. In the example the need was “to cross the ocean”. It was a clear by stated requirement to make the crossing. Things would be easy to start if we would know all the time what we want to do. We would see the target and the way to get there and then we would just cover the distance and achieve our goal.

Usually our needs are not so easy to realise. Ideas are usually very general and hard to describe in an understandable way. If we want to make a product out of our visions we need a process for preparing up our ideas so that we can say what kind of physical manifestation our visions would have this time. This is the job for requirement analysis.

Requirement analysis should generate at least a functional specification document. Other possible outputs are a preliminary testing plan,a project plan for launching the implementation project and a preliminary user manual. / 2, page 59 /

3.1 COSTS AND BENEFITS

Requirement analysis is one of the inexpensive parts of the system engineering process (Figure 3). Later the costs will rise when the design starts to get more detailed.

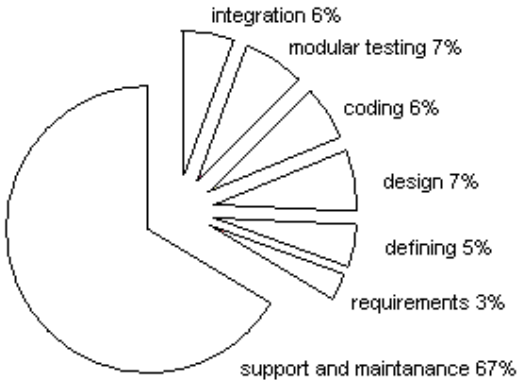

Figure 3. Cost distribution during the lifecycle / 2, page 39 /

The biggest potential for cutting the costs is in support and maintenance. It seems clear that one way to achieve this is to ensure that the system is well designed and documents are kept up to date.

There are reasons why the system might fail and these reasons are listed by frequency. When inspecting failed products through the product perspective, the reasons can be classified (Figure 4).

Table 1. Distribution of error reasons.

Type I (28%)	"product qualities did not meet the needs of end-users, the product was good but nobody really needs it" (planning error)
Type II (24%)	"product qualities did not give a competitive edge" (planning error)
Type III (13%)	"product qualities are mediocre, someone has made this already better" (planning error)
Type IV (7%)	"product qualities did not fill the needs of the environment. Everything which can go wrong goes wrong." (Planning error)
Type V (15%)	"Technical errors, the design does not work with current technology" (design error)
Type VI (13%)	"budget error, the product is too expensive" (design error)

72 % of failures are caused of bad planning and 28% because of bad design.
/ 7, page 10 /

A well planned requirement process can cut out most of the common failure reasons and help diminishing the total costs by decreasing the maintenance costs.

3.2 REQUIREMENTS

When speaking of requirements in software engineering it does not mean that every requirement is possible. System engineers want to talk about the requirements that are pointed to the system they are designing. They want to limit the topic to the most important requirements which will define ideas and later the design in a comprehensive but not too wide way. That is because usually the software system will grow so large that one man cannot handle it all. They also want that the effort of the group needed is focused on a single goal. The group must be united to follow the same goal. Requirements will help them to carry out what they started to do.

Requirements are used for a variety of tasks in the life cycle and are consequently needed to be kept up to date throughout the development. Initially, they define the business and user objectives and are then used for an abstract definition of the solution. An individual design can then usually be optimised by selectively cutting out high-cost and low-benefit areas. Cost-to-completion estimates must be firmly based on the deliverable linked to the requirements. / 5, page 12 /

During design and implementation, potential changes are evaluated against their costs and impact on the design and requirements. Requirements are also a form of retained knowledge, a set of rules extracted from experience and re-applied to the next generation of new systems. / 5, page 13 /

3.2.1 IRRATIONALITY

When discussing about requirements there is a desire for them to be rational. It would also be nice to have a set of rational requirements that would generate rationality to the whole project. The bad news is that the design process not be rational because

- Requirements given for software are almost never wholly known at the beginning.
- Even if requirements would be known, many aspects joining to the design are covered later during the engineering process.
- Even if all facts would be known they would make such a mass that no one could handle them without errors.
- Even if there would be no errors, the facts can change during the process.
- People tend to stick to the solutions they have made earlier.
- Re-usage of old software can lead to strange solutions.

/ 2, page 41 /

3.2.2 RATIONALITY

On the other hand, there are things that convince to follow the rational process model. These are

- Rationale process will guide its user what to do in each phase.
- It is easier for people to join another project if the processes within follow the same principles as in earlier projects.
- When there is a model for a project, it becomes possible to plan and follow it.
- It will be easier for an outside prospector to inspect the process.

/ 2, page 42 /

The practice has shown that it is possible to follow process models and it is possible and reasonable to collect rationale requirements.

Even if the process itself is impossible to make rational, there should be rationale processes and documentation, which are followed as strictly as possible.
/ 2, page 42 /

To make a rationale specification, it means the specifications must be complete, sharp, flawless, understandable, testable and traceable. Complete means that the specifications define all things needed but not more. Sharpness and flawless may sometimes conflict with understandable. Testable means that during a test phase it should be possible to check the test procedures and check if all the requirements can be tested. Traceability means that requirements can be followed to where they have been derived from or what they are affecting (more at Section 5). / 2, page 49 /

3.3 GENERATING REQUIREMENTS

The same way as the whole software engineering project can be thought out as one big process, the smaller parts of a whole project can be constructed into smaller processes. This way the whole process will be constructed of smaller processes. Requirement analysis is the first part of a whole project and it should be constructed as a requirement analysis process which has the structure to serve the purpose of getting to the next phase of a whole system engineering process.

3.3.1 UNCOVERING THE BASIC IDEA

Now imagine that there would be a problem and there would be a feeling that some kind of software product can make profit. Continue to an idea that we want to produce software which will fit our wish. How can the idea be crystallised?
Five hundred years ago Columbus had a feeling that the riches of India will serve him well if he would find a shorter (faster) route to them. So at the end Columbus had the same feeling for profit as almost every business idea has and he wanted to cross the ocean to fulfil his dream.

If the product is created for a market, where the behaviour of the end users will decide whether software project will succeed or fail, a way must be found to satisfy end users' needs. Such set of user requirements must be generated that will guide the project to the crossing point which will take it to its marketing segment. In other words it must be found out what the customers want from the product.

The ship must be prepared and it must wait for the right weather to cross the ocean. Likewise the set of requirements must satisfy the environment the product will be dealing with. This is the part where user requirements are transformed to better-defined system requirements. If some of the user requirements are not in harmony with the system requirement, it would be the same as the ship would leak or it would be in a peril to miss the route and not reach the right destination.

Waiting for the right weather is like optimising the enterprise before leaving the shore and moving to the next phase. In system engineering many possible ways must be studied to achieve the goal during collecting system requirements, and then there is a need to choose a solution which fits the current situation best and proceed with it.

Finally if it is noticed, despite all the effort in optimising and analysing the set of requirements, that the wind changes and starts to blow the ship back. The plans must be changed and the ship must be rowed the final distance. Requirements should be possible to change if it is noticed that some of them are not well suited for a purpose.

When the requirement analysis is analysed, two different groups of requirements can be found out: user requirements and system requirements.

Separating these two is essential, because these two elements are so different in their nature and organisation. The former defines the results that the system will supply to users, while the latter imposes requirements on an abstract model of the final system. / 5, page 22 /

These two groups will also contain two kinds of requirements, functional and non-functional requirements. Very roughly functional requirements will describe what we want the system to do and non-functional requirements place constraints how these functional requirements are implemented.

Every non-trivial system has to interact with other organisations and people with in an existing environment. The environment is considered as all those things outside the system that will affect what we want to build. The major components influencing the developing system are

- operational environment (cooperating, competitive, and support systems);
- development environment. / 5, page 24 /

User requirements must also work within the context of the requirements of the business that has spawned the project. A customer may, for example, specify geographical areas of operations or when operations should be started. These are called business requirements. This business, which has spawned the project, is sometimes called a part of the requirement analysis phase. It can contain parts such as feasibility study or problem definition. Business requirements are good to be linked to the user requirements because there are many traces between these two.

In Figure 4 there are three basic steps in requirement analysis. These steps are a proposal of my own according to the discussion in previous sections. This model is very general but is used as basics in what is needed. The first two steps are the main steps, and they should be included in every requirement analysis. The last two steps are a path towards a design; these are the phases where requirements are optimised and a proposal for a design is made. After an initial analysis phase, the set of requirements must reviewed occasionally in moments when input comes from later parts of the design cycle. This will help when the need to do changes arises or when a decision to start new releases of the system has been made. In other words, there must also be a way to collect and process the changes.

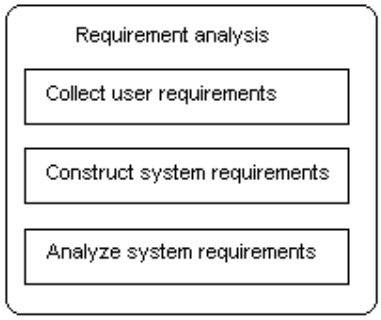

Figure 4. Inside requirement analysis.

3.3.2 ADJUSTED MODEL

Sometimes the requirement analysis is also included with a feasibility study. Feasibility study comes around when approaching a whole new area and when it is not known if it will create new business. It is like a reconnaissance for a military action. It is good to have at the start when it will be of help before the action begins. Feasibility study can be made analysing the problem area at the start of the requirement analysis phase. This is a good way if the marketing segment is not known. This is like looking for shore quickly and deciding if there is a real need to try to cross the waterway. If there is not the effort can be used in some other projects. This part is usually done when new business is generated.

Sometimes prototyping can be used as general method for feasibility study. This is the way which is used when there is a solution for markets but decision makers are not sure if the solution is solid enough. This is like making a small boat and testing it on short trip to ensure that the boats can be made that way. If the boat makers are unfortunate enough to notice that they can not make boats they will realise it soon enough and not in the middle of the ocean, but if they have enough spirit they can construct a big boat and cross the waterway.

Sven Dahlman has presented a model for a user-oriented approach to the requirement analysis (Table 2, / 1, Page 176 /). This differs from the basic model presented in the previous chapter in a way that a project definition part is here in the beginning where the problem area is mapped. This can be called a feasibility study. This model will also contain prototype generating which part was used in MPP as a feasibility study.

Table 2. Schema of an adjusted model of a user-oriented approach / 6, page 176 /

Project phases 1-4	Methods	Documents	Decisions regarding
1. Project definition. Searching, examining, based on normative values.			
Choice of problem Description of problem structure	-literature studies -hearings -interviews with strategic persons -interviews with experts -analysis of functions -system analysis -study visits	Problem structure -delimitation of system -relevant variables -hypothesized dependencies -interested groups	Choice of problem
Formulation of goals and project Analysis of interested groups			Formulation of goals and project
2. Collection and structuring of information. Recording, systematising, objective.			
Plan for user studies	-inventory of resources -allocation of resources	Plan for user studies	
Requirements of the users and the usage situation.	-formal interviews -informal interviews -user and observer scaling -demonstration-observation -measuring env. Conditions -measuring user ability -projective methods -recording usage.	User requirements	
Formulation of user requirements	-identification of object -ident. Of verification method -ident. Of verification level		
Allocation of priorities and systematising to user requirements	-analysis of needs -methods for allocating priorities -aggregation -sorting with regard to object level -analysis of connected functions. -complementary expert requirements.	Use requirements	
Strategy of solution			Strategy of solution -object (system or subsyst) for technical development -consequenses for users

3. Development of requirements & solutions. Interpreting, looking for structure, transforming, creative			
Goals of the development work Interpreparation and transformation of requirements Generation of ideas and design work. Specification of requirements Primary product Building of a prototype	 Delimitation of system Systemfuncs allocation Product type Performance of activities Working principle Working method Configuration Critical properties Embodiment Physical properties Materials Prefabricated parts Solution Specification of requirem.	Goals of the development work Principle product -scetches -mockup models Primary product Prototype generation I -drawings -specification -prototype	 Building of a prototype Evaluation study -object for evaluation
4. Evaluation of requirements and solutions. Recording, comparing, objective.			
Plan of evaluation Evaluation studies Analysis of the evaluation Modification of redesign	-inventory of resources -choice of methods and locality -in real use -in experimental use -in laboratory situation -measurement of single properties -against requirements -against goals -against needs	Plan of evaluation Evaluation results Primary product: Prototype generation II	 Modification of primary product prototype
		Technical product appropriate for use	

3.4 PLACE FOR USER REQUIREMENTS

To be successful the system needs to satisfy its end users, and so it must defined who they are and what they want. If the product is understood well it is to know what is dominant today, but understanding the user requirements tells what will dominate the future. These needs can then drive all subsequent development stages from a user perspective. Even if the requirements are not all practical, user needs must be understood.

When starting to collect the user requirements, first the users should be identified. The users, who will use our system and stakeholders who have some influence on our system, must be separated. The environment of the product must also be covered.

For defining the user requirement, we must find the needs of users. Users often initally state their requirements in terms of solutions, and these have to be pushed back to the real requirements. For example a user might start by stating: “I need to archive the system in a database every week.” A moment’s reflection will make you realise that users do not care about archiving, databases or storage of information. They need to avoid loss of work or perhaps to be able to retrieve information. By asking the question “Why?” we could find out the actual requirement, which might transform the original statement into: “I do not want to lose more than one week’s work by any predictable incident” The second expression is a far better expression of what is wanted.

User requirements are usually captured in a random, disorganised fashion, and they cannot be organised interactively. They must be organised into the right structure and style for designers to use them well.

A backbone structure for the user requirements is the ‘operational scenario’. This is a ‘thought experiment’ analysing the results provided to users as the system is operated, organised by the time. Breaking goals into sub-goals allows the

requirements to be viewed at any required level of detail. ‘Use cases’ in software follow similar principles. Capabilities are user requirements which define main capabilities the system is able to do within an operational scenario. Another major type of user requirements are constraints, not adding any extra capability, but affecting the quality of results provided. These are types of requirements which do not fit comfortably within the scenario – they might apply across the whole scenario or parts of it.

In addition to an individual requirement, we need to add some attributes to it. Attributes are extra information attached to individual requirements, for a variety of purposes such as explanation, selection, filtering, or checking. Individual requirements can be assessed against a checklist and flagged for different characteristics. For example, each requirement must be verifiable, clear and unambiguous. Information to support these characteristics is tagged to each requirement as an ‘attribute value’, i.e., information linked to the main requirement. Typical attributes attached to user requirements are: / 5, Page 36 /

- Source – who asked for the requirement?
- Priority – how important is the requirement?
- Performance – how quickly must this requirement be met?
- Urgency – how soon is the requirement needed?
- Stability – is the requirement really solid enough to start work on?
- Verifiability – can the final product be tested or assessed against this requirement?
- Ownership – who needs this requirement?
- Acceptance criteria – what is the nature of the test that would satisfy the user that the requirement is met?
- Absolute reference – an unchanging control tag that identifies the requirement uniquely. The reference is not re-used if the requirement is moved, changed or deleted.

3.5 COLLECTING USER REQUIREMENTS

There are a variety of different sources for user requirements (Figure 5). You must always remember that users will always know more than you – about their needs from the project. Users 'own' the requirements but they can rarely write them down in a structured organised form. The specialised role of a requirements engineer captures the requirements, writes and structures the information in to more suitable form. This demands persistence in questioning users, forcing requirements to be clarified, and pushing back solutions without ever trying to impose personal views.

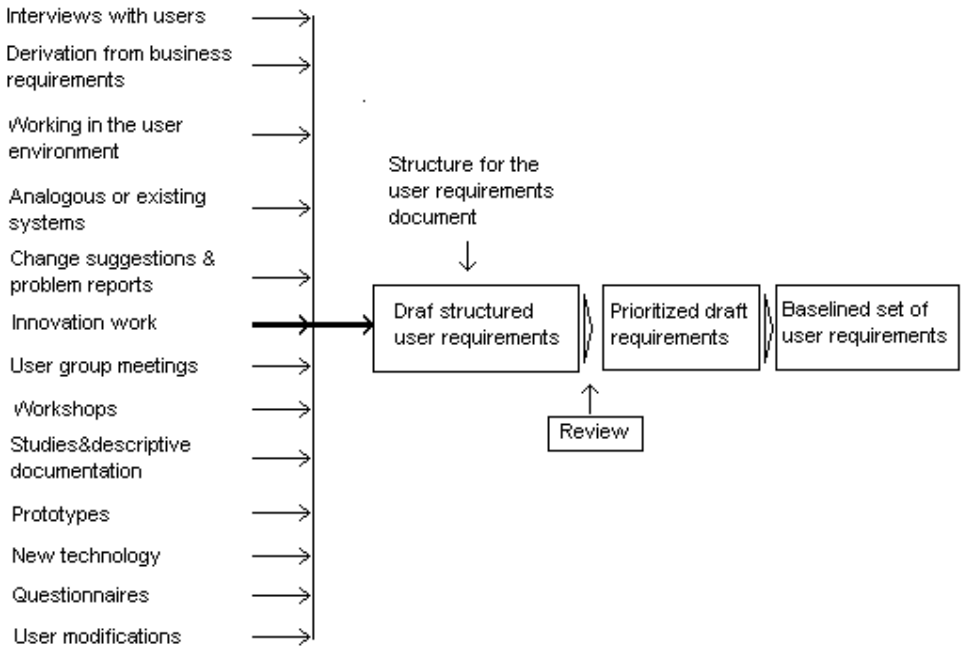

Figure 5. Sources of user requirements / 5, page 28 /

After collecting and organising user requirements we need a way to close the collection. The initial user requirement process is closed by a formal review of the user requirement document. Review is a powerful mechanism for making plenty of small enhancements, by focusing the brainpower onto the document as a whole. / 5, Page 40 /

The review starts by 'baselining' the user document, and issuing it to the reviewers. They must be notified in advance and allowed sufficient time to read and

understand the material. Reviewers then write change requests to the document and locate the problems. These change requests are then sorted by the review secretary against their position in the document. Change requests are typically handled on specific forms which allow each change to be managed and documented as it flows through the process. Non-specific change requests are rejected and returned immediately – trying to provoke the reviewer into making better, more specific, suggestions. All change requests referring to the same problem are joined together to allow a single decision to be made about the whole group.

During the review, all the requirements proposed for change and their current status must be available to every reviewer. At the review meeting a decision is made on each change request moving through the document. The only allowed decisions are 'accepted', 'rejected' or 'accepted with modification'. A review decision can, however, be put on hold while further exploratory work is performed outside the review. When all decisions have been made, the review meeting is closed, but the review process is not finished. All decisions have still to be implemented in the document and all the background work be done to get those decisions realised. The action list from the review has to be chased down to zero, and only then can the user requirement document be signed off.

3.6 PLACE FOR SYSTEM REQUIREMENTS

System requirements explore the solution, but ideally avoid commitment to any specific design. Defining them is a highly creative process, aimed at showing what the system will do, but not how it will be done. The system requirements form a model of the system, acting as an intermediate step between the user requirements and the design, often couched in functional terms. System requirements have to be traceable to both user requirements and design, but they are primarily an artefact needed for the development.

System engineers 'own' the system requirements. Users should understand them enough to be confident that they meet their requirements. System requirements

should meet every user requirement and users should be able to check this. Adding some extra detail into system requirements allows users to detect those that were not thought through correctly. This can lead to controlled change in user requirements.

System requirements contain both formal requirements and descriptive information. They have several distinct uses:

- Giving an abstract view of the system;
- Allowing trade-offs, exploration and optimisation before committing to design;
- Demonstrating to users how their needs are reflected in the development;
- Providing a solid foundation for design;
- Providing a basis for testing the final system;
- Communicating the previous decisions to developers. / 5, page 50 /

System requirements need to be understandable by almost everyone in the project, so they must be short and clear. This also means that the notation for systems requirements should be as non-technical as possible, without sacrificing accuracy. In practise, both textual and graphical notations are essential for a typical mix of users (Figure 6). Specific types of system requirements, such as safety, integrity, security or legal requirements, may need to be expressed formally, and consequently may not be easily be understood by users. This is fine as long as the system requirements are traceable back to user requirements.

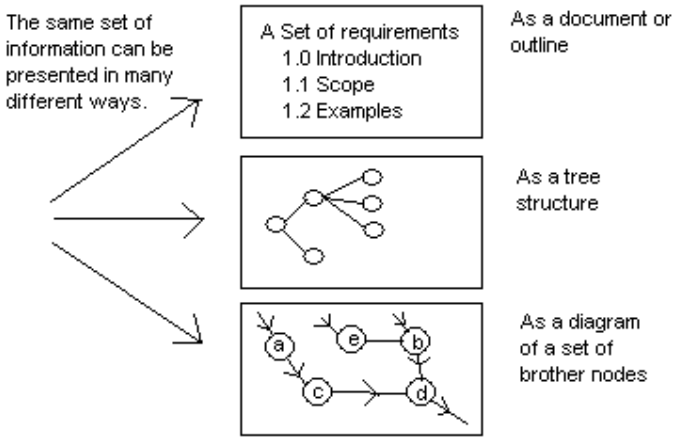

Figure 6. Different representations of information. / 5, page 51 /

Many different characteristics may need to be defined – functionality is not enough. System requirements must show how non-functional requirements, such as safety or reliability, are linked to specific functions. Functionality in itself is useless if the function is, for example, unreliable or not fast enough. Any of the following types of requirements may also be necessary:

- performance requirements;
- information relationship and history requirements;
- temporal and dynamic behaviour requirements;
- requirements for parallelism or concurrency;
- logical behaviour (e.g. conformance to a mathematical model);
- flow of control;
- flows of data or material;
- non-functional requirements (constraints);
- interactions with external systems;
- End-to-end scenarios. / 5, page 54 /

In a system requirements document, the constraints are typically organised (Figure 7) in the following sections:

- transformation of user requirements;
- discipline-specific constraints;
- applied and induced environmental constraints.

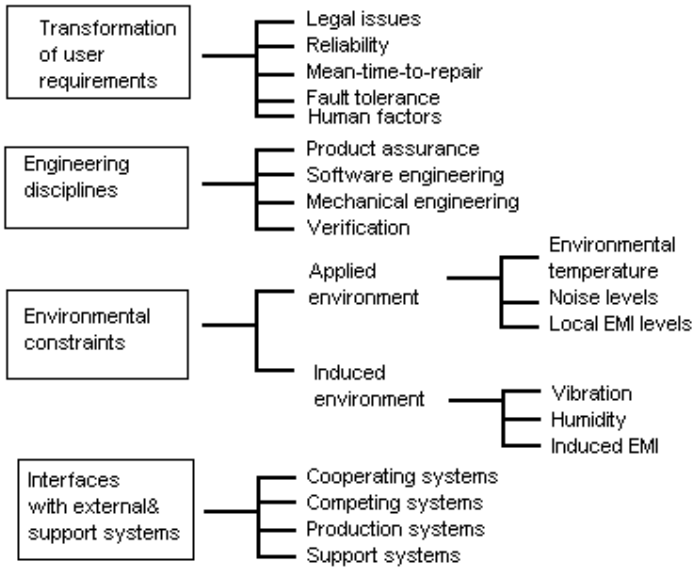

Figure 7. Example sources and targets of non-functional requirements. / 5, page 70 /

3.7 GENERATING SYSTEM REQUIREMENTS

Figure 8 presents the overall process for defining system requirements. The first activity defines the major functional elements, typically as functional block diagrams or state diagrams. Non-functional requirements are defined simultaneously and linked to the relevant functions. Transforming functional requirements into textual form can make them more precise and approachable to non-specialists. This is one of the few times in the system life cycle that information duplication is advisable. The review process for system requirements is similar to that for user requirements, but developers control the requirements instead of users.

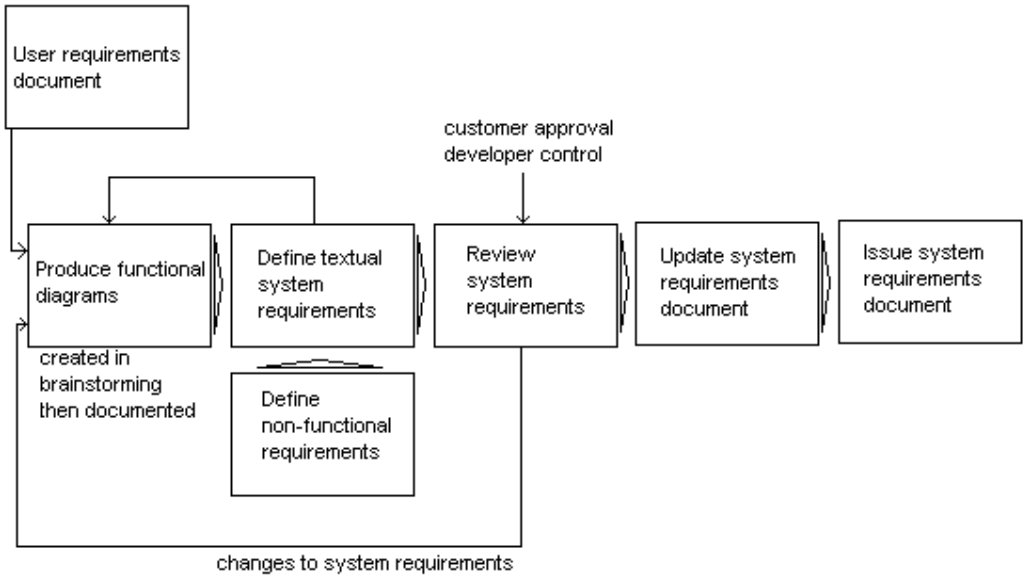

Figure 8. Producing system requirements. / 5, page 52 /

Any realistic set of system requirements will need to be organised hierarchially, helping us to view and manage information at different levels of abstraction. Decomposition should be done two or three levels at a time, exploring the levels below before confirming any choice above. At the top level of a system, this may take only a few minutes, and involve intense interaction and arguments between engineers.

Non-functional requirements should not be applied in bulk at the system level. A requirement such as a safety constraint may be limited to a single function. Unless this is done many functions will end up being over specified.

4 IMPLEMENTED REQUIREMENT ANALYSIS PROCESS

There are existing pilot projects for a mobile payment, but all the applications are independent. If continued this way the approach was thought to cause problems in the future. The main problems would be complexity for support and the cost in production. Complexity would come from many different applications and their versions. Cost in production would rise because of the need to develop many individual systems. The business wanted more efficiency and engineering needed to cope with it. The idea of implementing all mobile payment services to the one platform was born.

4.1 FIRST ROUND

The first round of requirement analysis for MPP started in Week 5 2000. This round is called the first because the SMP department was just formed for a new venture. At the beginning it was not decided if the MPP would form a project. The first part was to make requirements and test if the concept is possible. This is often the case with new technologies. The time schedule was also tight because of the business, which required pilot project to start in June. Weeks 5 and 6 were allocated for collecting the user requirements. Weeks 7, 8 and 9 were saved for a feasibility study to test if the concept can be implemented.

Figure 9 presents the theory constructed earlier. The whole requirement analysis phase is divided into 2 pieces which fall down to phases called collecting requirements and feasibility study. In short, the requirement analysis went through collecting user requirements which formed the requirement catalogue. After this the analysed main requirements document was constructed. This name is a bit misleading compared to the steps in theoretical requirement analysis, actually this

represents better the system requirements document and could be named as it. The document grew in size as the system was analysed and the basic functionality was tested with prototype. This part was analysing the system requirements. It did not create a new document but it purified the system requirements. During the process the document became an analysed main requirements. After the analysis phase a technical implementation proposal was constructed. It contained the system requirements and the knowledge of tested technologies that can be used to implement the concept.

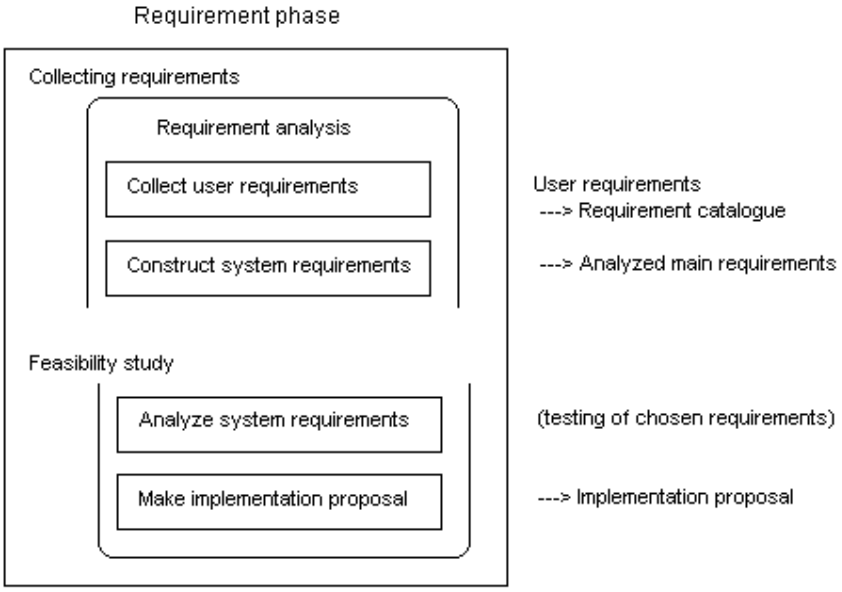

Figure 9. Comparing theoretical steps to the steps in actual process.

In the next chapters we review more detailed how the requirement analysis was carried through.

4.2 STRUCTURE OF THE USER REQUIREMENTS DOCUMENT

Requirements can be gathered and ideas invented, but they remain formless unless they are given a form. As long as they are formless they remain invisible and are hard to pass on forward. A good way to give ideas a form is to write them down in such a way so that they can be called, for example, user requirements. (Section

3.2.1). When these requirements are collected into same document, the document will also take form. The form is as important to a document than to a requirement. Documents can have pre-made forms which are called templates. These templates show a way how to structure information. When the right form for a requirements document is chosen it can help collecting the requirements by showing what kind of information is expected to fill the structure.

With the MPP-project the starting point was such that the users/stakeholders were quite clear, but the system was a bit dizzy. With this kind of problem it is good to approach unknown from a direction that is visible. This is the case where one has a business idea but not the way to achieve it. This leads to a viewpoint approach.

4.2.1 VIEWPOINT APPROACH

The main idea with viewpoint approach / 4, Page 72 / is that if we look our system concept from different perspectives, it will look different and it must deliver different functionalities. For example end-users will be using mobile phones to get the service and our billing partners will be monitoring if transactions are delivered into their systems. The billing partner does not care much of the time it takes for a user to use the service but he/she will care if the bills are coming properly from the system. So by taking different views from different stakeholder groups the whole spectrum of required attributes for the system can be generated. Viewpoints can also be arranged by the functionalities, not only by the stakeholders.

The principal advantages offered by viewpoints are as follows.

- The requirements are likely to be more complete than if viewpoints are not explicitly identified. In the latter case, important requirements may be easily overlooked because their viewpoints were never recognised.
- A separation of concerns is provided which permits the development of a set of ‘partial specifications’ in isolation from other viewpoints. This avoids having to conform conflicts with other viewpoints’ requirements during elicitation. A result of this is that, when they prove necessary, the trade-offs between requirements can be better informed.

- Traceability is enhanced by the explicit association of requirements with the viewpoints from which they are derived.

4.2.2 VIEWPOINTS WITH MPP

The first part in the MPP-project was to find out who are the users interacting with the new system concept. Three main groups were identified right from the beginning. They were: end user, service operator and vending operator. Each group was reviewed and requirements that could come up with brainstorming were written down. In the light of these requirements, the systems main functionality was thought up and list of different stakeholders was updated. To help collecting new and arranging old requirements, the requirement catalogue was divided into main functionalities, which were identified to be:

- Registration
- Ordering
- Delivery control
- Payment
- Management
- General

Updated actor or stakeholder list contains:

- User
- Product provider
- Payment service operator
- Service owner
- Partner operator
- Billing partner
- Mobile portal
- IVR
- Support

Under each of these functionalities, there is space for requirements for each of stakeholder group. Every stakeholder does not have requirements for every

functionality but every functionality will have requirements from some of the stakeholders (Figure 10). Main advantages from this type of structure seems to be:

- Whole big entity is divided into smaller entities, which will contain their own requirements. Requirements are easier to collect, because they can be collected one functionality at time.
- Reviewing through requirements is easier because there is some logic in structure of how they are collected.
- Structure makes it possible to continue breaking requirements into components while starting to construct system requirements. Traceability is more easily supported because the base of the functionality breakdown can be used with system requirement documents also.

1	Purpose
2	Revision history
3	Terms, acronyms and abbreviations
4	Introduction
4.1	Sender
4.2	Receiver
4.3	Background
4.4	Description and motive
4.5	Planning information
4.6	Acceptance criteria
5	References
6	General requirements
6.1	Technology independence
6.2	High availability
6.3	Architecture
7	Registration
8	Ordering
9	Delivery control
10	Payment
11	Management
12	Example of use

Figure 10. Structure in a first version of MPP requirement catalogue.

4.2.3 PROBLEMS WITH THE APPROACH

The main problem with the current structure is that it is hard to understand. Almost all of the requirements are placed under the functionality that they are thought to belong to. This is an easy way when the requirements are identified for the first time but it seems to cause a setback when someone asks something that is not directly connected to any specific funtionalities. Examples are the performance and interfaces. These kinds of questions are prone to rise with possible business partners. Partners cannot know the functionality and so they cannot ask any specific questions about some functions, even Sonera's own sales people find this hard to handle. The first solution was to make a document which was aimed to sales people and to customers. A better solution might be to reconstruct the bad parts from the requirement catalogue structure and maintain the good part (see Chapter 6 for a

suggested solution). It seems wise to use a little more effort while generating a better structured document so that later there will be less need to generate whole new documents ad hoc (see Chapter 3).

4.3 HOW TO DESCRIBE USER REQUIREMENTS

Single user requirement in requirement catalogue contains an identification code and a description. The description contains information of importance of the requirement , we categorised them by priority as must-, shall- and may-requirements. This is a rough priorisation because must-requirements are something which are implemented into system, shall-requirements are something that shall be implemented some day and may-requirements are something of which we are uncertain if they will ever be implemented. Good for this is that the single requirement does not take too much space to be described and this will limit the size of the requirement catalogue so that it is easier to be read through. A drawback is that requirements will miss information that is attached for requirements in theory discussion (Chapter 3.4).

These missing attributes are

- Performance – how quickly must this requirement be met?
- Urgency – how soon is the requirement needed?
- Stability – is the requirement really solid enough to start to work on?
- Verifiability – can the final product be tested or assessed against this requirement?
- Ownership – who needs this requirement?
- Acceptance criteria – what is the nature of the test that would satisfy the user that the requirement is met?

4.4 STRUCTURE OF THE SYSTEM REQUIREMENTS DOCUMENT

The system requirements document at MPP project is called as “analysed main requirements document”. It contains an abstract view of the system. In the user requirements document the requirement was textual and in one format. System requirements are different. Only part of them are in textual format and the rest of the requirements are presented in graphs or diagrams.

The system requirement document contains (analysed main requirement document)

- An abstract view of the system.
- Information about pieces represented elsewhere, requirements for testing, documentation, deployment and runtime maintenance.
- Main use case showing all the functional parts of the system.
- Detailed diagrams for each functional part.
- Traceability matrixes for checking which user requirements are filled.
- A list of user requirements which are suspended from the current design.

First parts of the system requirements are general requirements. They are requirements, which will affect the concept as whole, not just some functions. They are constructed mainly from the requirements coming from the business and then processed to the basic environment and basic functionality requirements. These requirements are quite general. These might be for example “R20. Service application fault control must be supported.”

The document contains also chapters for testing, documentation, deployment and runtime management processes. These are detailed more specifically with test plan and other documents.

In the last chapter of the document the functionality of the platform is presented with use cases. Use cases contain diagrams and textual information to present particular case with functionalities and operations.

4.5 HOW TO DESCRIBE SYSTEM REQUIREMENTS

In the chapter 4.4 it came out that system requirements are presented with many different ways. In the Mobile Payment Platform two different kinds of diagram presentations and textual presentations have been used.

Textual presentation is used for requirements that are not directly tied to any single function as a functional part of it. Diagrams are used to show how different functions within the concept are wanted to operate. So the system requirements are not about how we want the functions to work, they are more about what operations we want to put into our functions. System requirements define small operations and together they form the concept which shows the designer what our functions need as input and what is wanted to come out as output.

Functionalities are presented as diagrams and explained with text. First diagram is a main use case (Figure 11). The main use case contains all functions on level which shows who are the participants dealing with each function. These functions contain tasks like initiating the service (ordering from vending machine for example), registering, managing profile and calling to helpdesk.

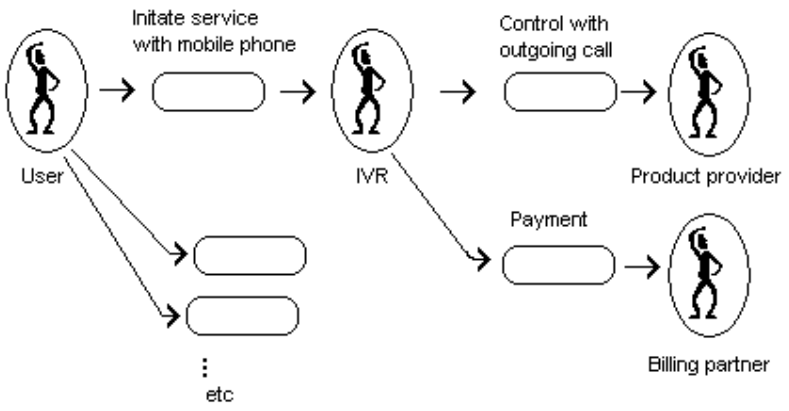

Figure 11. Example of main use case diagram.

Following the main use case there are chapters to describe each functionality. For example one functionality case is the ordering for vending machines. This use case is presented with a diagram (Figure 12) which shows more accurately how each actor is communicating with each other. This is followed by textual description explaining what happens when something is ordered. The function is divided into operations which will follow each other. These operations are listed also in textual form. Functions contain also a list of error cases and how they are planned to recover. For example one of the errors with ordering is "E-1: the number of the service does not answer" and a way to recover follows it "The user calls again. The administrator of the system is informed of problems."

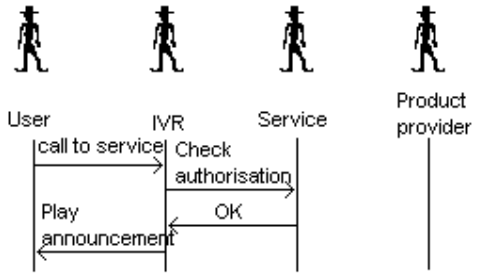

Figure 12. Example of the diagram showing communication between each actor.

Each use case has also a matrix attachment which has entries for requirements that are traced from the user requirements to the system requirements in that particular use case. If some of the user requirements need more details, there is a column in a

matrix which contains a more detailed hint. For example the registration chapter contains a matrix entry, as shown in Table 3.

Table 3. Example of matrix entry.

User actor	Fulfil	Comment
R23. Use of some services must not require registration.	Yes	Only phone bill

This means that the concept has a user requirement which tells the service about use without registration. The comment adds a detail that this is filled if the user uses phone bill as a payment method. The process is presented with more detail in use cases. Matrix is for traceability and checking that all the requirements are taken care of in the document. Unused user requirements are listed as an appendix so that it is possible to check what has been left out from the concept.

The systems requirements should satisfy the following list of features: / 5, Page 50 /

- Giving an abstract view of the system;
- Allowing trade-offs, exploration and optimisation before committing to design;
- Demonstrating to users how their needs are reflected in the development,
- Providing a solid foundation for design;
- Providing a basis for testing the final system;
- Communicating the previous decisions to developers.

5 TRACEABILITY

One demand for a good requirement was to be traceable (Section 3.2.2). If requirements are traceable then it can be said that traceability is included in them. Without working traceability requirements and design could not communicate and it would not be known what would happen if something changes requirements or in worst case, no one would even know if they were on a right track, because there would not be a workable way to check it out. Traceability is needed for system requirements to be traced to user requirements to show that all the user requirements are met, and that all the system requirements are necessary. This task needs engineering judgement, is arduous and error-prone. Also users always need to find out which requirements are accepted, rejected or postponed which is why traceability to the users requirements must be retained. Traceability is not implemented for free. It needs some tracking and maintenance to work. Some requirements management tools can help document the links and their rationale.

5.1 THE NATURE OF TRACEABILITY

The nature of traceability is to show us a way to answers considering our requirements. By studying books describing the questions, lists of different questions can be found out to define different types of traceability.

A set of questions for traceability to answer:

- Are these user requirements met by the current design?
- Are these user requirements met by the current implementation?
- What is the level of criticality of this piece of equipment?
- What is the functionality of this equipment?
- Which requirements are to be met in the next release?

- Which user requirements have not been tested?
- What is the likely cost of this proposed change? / 5, page 270 /

Information on the source of the requirement could also be added to the list.

Another definition for a traceable requirement is:
"A requirement is traceable if you can discover who suggested the requirement, why the requirement exists, what requirements are related to it and how that requirement relates to other information such as system designs, implementations and user documentation." / 4, page 217 /

5.2 DIFFERENT TYPES OF TRACEABILITY

A lot of things can be asked of requirements. All these questions would get different answers. As many questions can be made as there are relationships between requirements. As many answers can be given as there are questions. These question/answer dilemmas can be called as relationships between the requirements. I have found a list of relationships by combining lists from / 5, page 270 / and / 4, page 226 /. My list includes:

1. A relationship describing who specified the requirement. This is recording the requirement source.
2. A dependency relationship between the requirements. Also called coverage relationship which is showing that one process covers the specification of a previous process. This should always be included.
3. A relationship between the rationale and requirement. Also called as an applicability relationship to show how non-functional requirements apply to functional requirements;
4. A relationship to show how tests are related to components and the requirements for those components;
5. A development relationship that show who is responsible for performing specific tasks. This can be divided into three different domains, which may or may not be applicable for a single requirement. These domains

are requirements-architecture, requirements-design and requirements-interface traceability.

6. A relationship to show how structured information is related to descriptive information.
7. Relationships of generic engineers to specialist engineers, usually working with specialist tools (data flows, control flows).

To achieve traceability there are a few principles that must be followed. Traceability tends to weight our requirement process and we do not want to get it too heavy. Too heavy requirement process would also easily lead to piles of requirement data, which nobody would have energy and will to comprehend.

When dealing with the problem of traceability and thinking of what relations are more important than others are, there will be a need to sort these relations out somehow. Figure 13 presents a requirement catalogue which contains 4 requirements. These requirements and this catalogue are tied to its surrounding. Lines and arrows in the figure show how the business idea comes before requirements and how the test plan tests the single requirement and how the implementation proposal is made after collecting requirements. There is also a market study showing a new market invention that is converted to requirements and that way into the design. The picture shows the main types of relations with requirements themselves and their surroundings. The picture should show also how these relationships can be sorted out with couple different characteristics:

- Whether the relation is forward or backward in a design cycle;
- If the relation is going inside or outside of scope of the requirements catalogue.

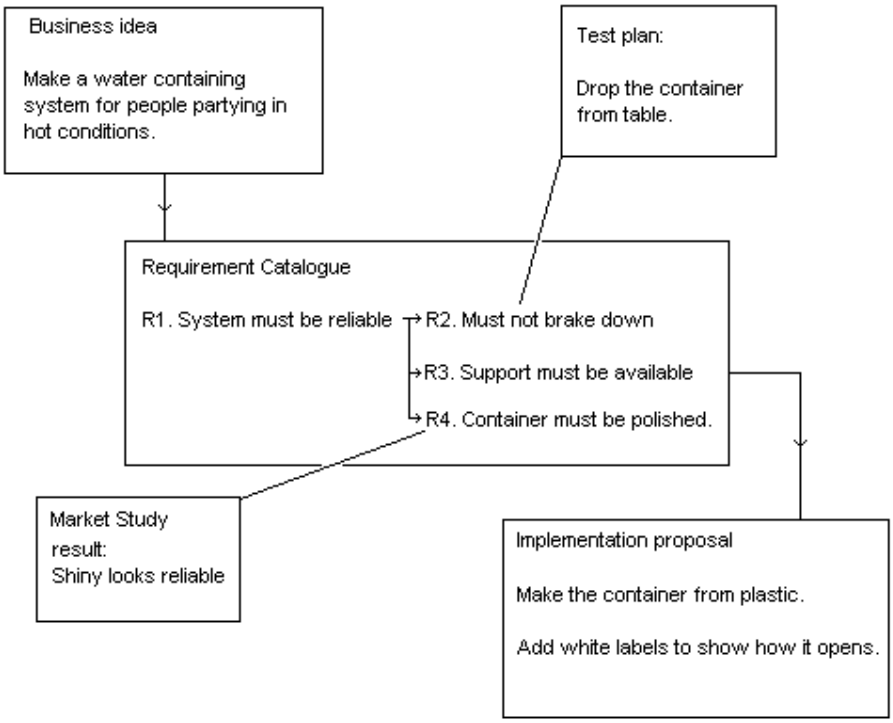

Figure 13. Traces leading forward/backward and going inside/outside requirement catalogue context.

5.2.1 FORWARD/BACKWARD

In this study, it was noticed that there are forward and backward relationships. Forward type of relationship means to look what will follow the requirement in a design cycle. Backward means to look what was stated before the requirement.

All types of relations contain both forward and backward type of relation. That is because if there is a relationship between two objects in a traceability kind of sense, we can look relations both ways. If we are looking at, for example, what will follow an object as more detailed requirements or as some designed solution, we are looking at a forward trace. If we look at the same relations the other way, from the design solution or some detailed requirement, we can find out what was the source we have used to justify our detailed requirement or solution. This is a backward trace.

Some might ask about requirements, which are simultaneous? If they are in the same phase and do not follow each other, but are like two different requirements each describing the same source requirement, like requirements R2 and R3 in Figure 13, the answer is, they do not have a trace between them. Surely they have some kind of relation, everything is relative, but in the context of traceability they do not have a straight trace between them.

5.2.2 INSIDE/OUTSIDE

In this study it was noticed that relationships can also be arranged by inside/outside type of relations. Inside type relationship means we are dealing with relationship which does not lead directly to the outside of our requirement document. An example of this could be the relationship between non-functional requirement and functional requirement. An outside type relationship means we are going out from our requirement documents;for example, if we have a relation between a requirement and the design component we are speaking of an outside trace.

Examples of inside relations are using the numbering of the list in section 4.2:

- 2. A dependency relationship between the requirements. Also called coverage relationship which is showing that one process covers the specification of a previous process.
- 3. A relationship between the rationale and requirement. Also called as an applicability relationship to show how non-functional requirements apply to functional requirements;

Examples of outside relations are:

- 1. A relationship describing who specified the requirements. This is recording the requirement source.
- 4. A relationship to show how tests are related to components and the requirements for those components;
- 5. A development relationship that shows who is responsible for performing specific tasks. This can be divided for three different domains which may or

may not be applicable for a single requirement. These domains are requirements-architecture, requirements-design and requirements-interface traceability.

- 6. A relationship to show how structured information is related to descriptive information.
- 7. Relationships of generic engineers to specialist engineers usually working with specialist tools (data flows, control flows).

5.3 TRACEABILITY TECHNIQUES

The whole system can be thought of as a set of organised information, linked to minimise duplication. A requirement should ideally be stated once and applied to many different areas, by linkages, rather than repeating the requirement. If a change is required, only one item needs to be updated, and the result is cloned to many different places. / 1, page 269 /

There are three basic techniques, which may be used to maintain traceability information. These are as follows.

1. Traceability tables
2. Traceability lists
3. References
4. Automated traceability links

5.3.1 TRACEABILITY TABLES

Traceability tables show the relationships between requirements or between requirements and design components. The requirements are listed along the horizontal and vertical axes and relationships between requirements are marked in the table cells. They can be implemented using a word processor or spreadsheet tables; a requirements database is not necessary. / 4, page 227 /

Traceability tables showing requirements dependencies should be defined with requirement numbers used to label the rows and columns of the table. Then if requirements have some kind of dependency you simply put a mark (* for example) in the table cell. Tables should be constructed so that the leftmost column shows a requirement and rows are showing which requirements are dependent on it. For example in Figure 14 we have three requirements (R1, R2, R3) and we can see that R2 is somehow dependent on R1 and R3 is dependent on R1 and R2. R1 is not dependent on any other requirements. This kind of representation shows us both a forward and a backward type of relations in a same table. The main disadvantage is that the tables will become unmanageable if the number of requirements grows too big.

	R1	R2	R3
R1			
R2	*		
R3	*	*	

Figure 14. Example of traceability table.

5.3.2 TRACEABILITY LISTS

Traceability lists are a simplified form of traceability tables where, along with each requirement description, you keep one or more lists of the identifiers of related requirements. Traceability lists (Figure 15) are more compact than traceability tables and do not become as unmanageable with large number of requirements. / 4, page 229 /

Requirement	Depends-on
R1	
R2	R1
R3	R1,R2

Figure 15. Example of traceability list.

Traceability lists are more compact than traceability tables. For this reason they do not become unmanageable with large number of requirements. Extra information like comments can also be added to them (Figure 16). If both forward and backward type of relations are shown, independent lists must be maintained for these relations.

Requirement	Depends-on	Comments
R3	R1,R2	Here is something good to know.

Figure 16. Example of a comment field in a traceability list.

Traceability lists can also be joined together to form forward and backward type of information and still achieve more compact form than traceability table. (Figure 17).

Requirement	Depends-on	Affects-to
R1		R2,R3
R2	R1	R3
R3	R1,R2	

Figure 17. Forward and backward trace in a same list.

5.3.3 REFERENCES

References are also traceability information. They are often used in documents where some information is derived from some other document. Even in my thesis I must use references. I use them like / 2, page 236 / which means that I have found information from the source number two and on page 236. Requirements can contain references. This can be used when linking outside-type of relations, for example non-structured descriptive information, to the requirements. An example for descriptive information might be a marketing study.

5.3.4 AUTOMATED TRACEABILITY LINKS

It is possible to use automated traceability links if the requirements are maintained in an established database where individual requirements are stored as entries in this database. The main benefits are: / 4 , page 236 /

- It makes easier to maintain links between individual requirements and to search for and abstract related groups of information.
- If the database is a general-purpose repository for system information, links from the requirements to design and implementation information may be maintained.
- If the database supports concurrent working, it allows for different groups to work on the requirements specification at the same time without generating requirements inconsistencies. Database facilities for backup, integrity and security mean that requirements engineers need not be concerned with these issues.
- The requirements may be automatically processed to extract particular types of information. For example, it may be possible to generate traceability tables and lists automatically from the information in the requirements database.

5.4 TRACEABILITY MANUAL

To give traceability a form of some kind of traceability, techniques must be used. To give techniques a form, so that design and requirement engineers can benefit, they must also be structured. This is done by the same kind of principle why we must have a structured way to generate for example requirements document. The traceability manual should tell how the traces are generated and where they are stored. / 4, page 232 /

The traceability manual is a supplement to the requirements document which includes the specific traceability policies used in a project and all requirements traceability information. This document is used by requirements engineers and system developers.

5.4.1 BENEFITS

It is no use to make documents just because they look nice . So why not just implement the traceability right into the user requirements document and technical implementation plan, etc? If traceability policies and all traceability tables would be incorporated right into the requirements document, the size of that single document would grow much. Too much information (too thick document) will make it useless just because of lack of energy and time people have. The documents should be thin so that the information can be found fast enough. Another point is that the traceability comes from many different directions. It comes from design and it comes from stakeholders, market studies, etc. The information would spread all over the project and an exact piece would be hard to find if there would not be a central record for traceability information. It is possible to list all the policies in the manual. From these policies it is possible to check where to find a single piece of traceability information needed at the time or where to make updates when traces need changes. The manual should contain much traceability information.

The main benefits for a traceability manual is: / 4, Page 232 /

- Team members can easily find the specific traceability policies for their project.
- A traceability manual keeps all traceability information in one place and makes information (relatively) easy to find and update.
- The specific traceability policies used in a project are made available to all project members through the traceability manual.
- For systems where a safety or security case must be made a traceability manual may be used to show that components are independent or to argue that component failure cannot propagate in an uncontrolled way.

5.4.2 IMPLEMENTATION

The traceability manual is a central record of the traceability policies for a specific project and all of the relevant traceability information. Your general traceability policies should be specialised to take into account the characteristics of the project. This may involve leaving out some traceability information, deciding exactly how traceability information should be represented, deciding on the responsibilities for traceability information collection, etc.

The specific traceability policies, which should be used for a project, depend on a number of factors. These factors include the following: / 4, Page 233 /

1. Number of requirements.
2. Estimated system lifetime.
3. Level of organisational maturity.
4. Project team size and composition.
5. Type of system.

Traceability information must be regularly updated. If it is not it will not stay useful. The traceability manual is good to be implemented as a networked electronic document rather than as a paper document. When traceability information is needed it can be consulted from the screen or the important parts can

be printed on paper. Maintaining the whole document on paper will make frequent updating cumbersome. The traceability manual should be managed using normal configuration management processes. To ensure that the traceability manual is kept up-to-date, someone should be assigned to manage it. He/she should work with system developers and ensure that changes to the requirements/design, etc. have been incorporated in the manual and should review and update traceability policies. Assigning tasks to people and not just hoping that someone should do them is a workable solution because things that are not assigned to any specific person will soon turn out to be things not done by anyone.

6 ENHANCHING REQUIREMENT PROCESS

In Section 3.1.2 it was pointed out that the requirements should be complete, sharp, flawless, understandable, testable and traceable. The aim of this work is to discuss what traceability is and how it is implemented with requirements. It also seems that when enhancing traceability, the whole process can be enchanced the same time. This section contains practises that have been formed according to earlier discussion. I have found it important to discuss three topics which will have an effect on traceability:

1. Requirement process and its surroundings
2. Structure of the requirement catalogue
3. Traceability manual

6.1 REQUIREMENT PROCESS

Knowing yourself is basic for starting to improve. For this reason it is important to draw a figure how the requirement process will fit into the SMP business (Figure 18). This figure will also help the reader to understand what are inputs (traces into) and outputs (traces out from) to the requirement analysis.

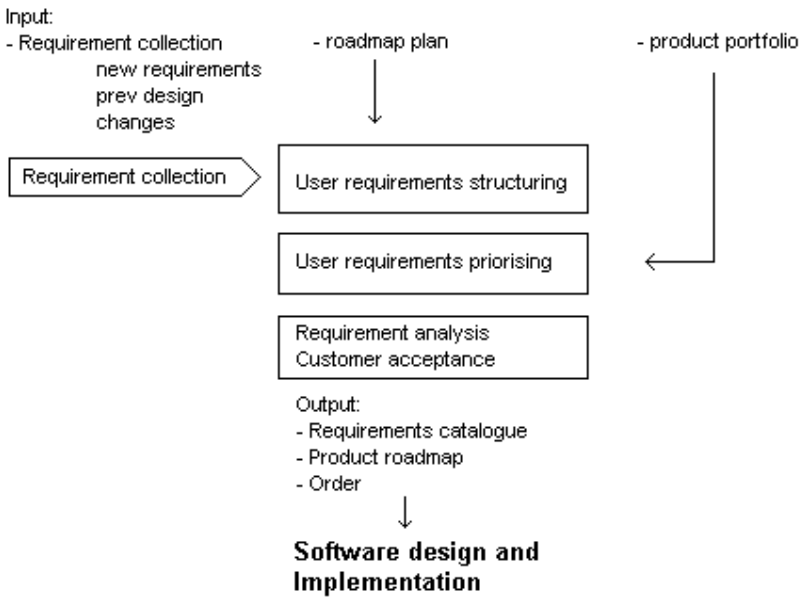

Figure 18. Process model for SMP requirement phase.

The requirement analysis gets as an input collected requirements, a roadmap plan and a product portfolio. Outputs for the process are a requirement catalogue, a product roadmap and the order. The order is documented independently because SMP does not make implementation but orders it from different subcontractors.

The first phase of this model is to structure the information. This phase will review requirement collection process and get a set of already collected requirements from there. Requirement collection is a continual process. It contains all types of requirement collection, starting from ideas and lasting to the collecting changes made through change management. The requirement collection is its own entity so that it is done during other phases of the whole system engineering process and during this time it does not consume much time. Only results are needed.

After structuring the requirements there is a phase where requirements are prioritised. This is where some are left out and others are reviewed and decided if they should go to the next release.

The last phase is analysing the requirements. This phase is here to deliver the requirement catalogue to the partners who are responsible for implementing the

design. The requirements may need to be presented as system requirements to ensure they are understood.

6.2 REQUIREMENT CATALOGUE

The structure of a catalogue is important. If the structure is bad even good requirements will become hard to use. On the other hand even bad requirements will benefit if the structure of the document is good. Traceability does also concern the whole structure of the requirement catalogue. It is not just in the requirements, it is also in the group they form. This shows up for example in version control.

The structure of a requirement catalogue must be made clear so it can be used to skim through the text to find the areas of interest. For example if the reader is interested mainly in the performance abilities and the catalogue has a chapter about performance, the reader can go right there from a contents list. Largely this is a part of traceability because it helps to find a connection between a quality criterium (performance ability) and a requirement. A more common sense way would be to say, “It is easier to read this”.

Figure 19 presents an adjusted structure for a requirement catalogue. It is based mostly on the model presented in reference 2, page 60, and adjusted with lessons from the previous MPP requirement catalogue structure (see Section 4.2).

1. Preface
   - 1.1. Revision history
   - 1.2. Contents
   - 1.3. Purpose
   - 1.4. Terms, acronyms and abbreviations
   - 1.5. Guidelines with the document
   - 1.6. References
2. General description
   - 2.1. Product background
   - 2.2. Environment
   - 2.3. Main functionality
   - 2.4. Modules and their relations
   - 2.5. User groups
   - 2.6. Performance
   - 2.7. Costs/benefits
   - 2.8. General constraints
3. Data and database
4. Functional requirements
   - 4.1. Functionalities
5. Interfaces
   - 5.1. User interfaces
   - 5.2. Hardware interfaces
   - 5.3. Software interfaces
   - 5.4. Datalink interfaces
6. Other characteristics
   - 6.1. Performance
     - 6.1.1. Static requirements
     - 6.1.2. Dynamic requirements
   - 6.2. Security
   - 6.3. Support & maintenance
     - 6.3.1. Maintenance
     - 6.3.2. Installing
   - 6.4. Compatibility
   - 6.5. Operationing
7. Design constraints
   - 7.1. Standards
     - 7.1.1. Software standards and programming languages
     - 7.1.2. Datacommunication standards
   - 7.2. Hardware constraints
     - 7.2.1. Used hardware
     - 7.2.2. Database constraints
   - 7.3. Software constraints
     - 7.3.1. Operating system
   - 7.4. Output formats
8. Testing requirements
   - 8.1. Acceptance Criteria

Figure 19. Structure for a requirement catalogue

Chapter 1 is a preface. It contains information about the document itself. It has the revision history from where the document is derived, a contents list, the purpose why the document exists, a term list to help understanding the text, guidelines for using the document and a list of references used with the document.

Chapter 2 is a general description. It has information about the product, the reason why the system is made, in what environment it is interacting, what are its main functionalities, what modules does it consist of, who are the user groups dealing with it in greater detail than in the environment chapter, what general performance ability it contains, what are its possible costs and benefits with business and what general constraints it deals with. General constraints might deal with law or work tools used to implement the system.

Chapter 3 contains requirements for data and databases needed in the system. It clarifies the information contained, data storing, capacity, search-time, etc.

Chapter 4 has an entry for each main functionality and these entries should contain a list of requirements for each functionality. There should be a purpose for function, the input it needs, how the handling takes place and what output it generates. The format for each technique sub-chapter should contain requirements organised from the viewpoint they came in with (see Chapter 4.2.1). This is to list requirements by a user group that the requirement most affects. The structure could be

4.1 Ordering
    End user:
    …
    Product provider:
    …
4.2 Payment
    …

Chapter 5 contains more definite requirements for systems interface. A general description for these should be found from Chapter 2.2 Environment. Interfaces can be described in detail or they can be left to design phase. It is optional.

Chapter 6 contains systems non-functional requirements which make up most of the quality requirements associated with the product.

Chapter 7 holds the constraining requirements. These contain limits which are set by standards, laws, software, hardware, etc associated with the product.

Chapter 8 holds the requirements for testing. It can contain tests needed and also criteria when tests are accepted.

6.3 TRACEABILITY MANUAL

Because the traceability techniques are not currently defined at the SMP department and the whole implementation process would take a long time, it is here rather discussed what should be done in the first place to create a condition for successful adaptation of traceability techniques needed.

The traceability manual should normally be developed incrementally as the system is specified, designed and implemented. The first chapter should always include the project traceability policies. Requirements dependencies can then be documented as soon as the requirements document is agreed but design traceability, documentation traceability, etc. must be added at later stages of the development process.
/ 4, page 233 /

6.3.1 COMPLEXITY

The Sonera Mobile Pay department is mainly concerned with designing the concept for mobile payment and selling it internationally. The department generates requirements for the concept and tests the functionality. The design phase where the code is generated is carried out in different departments. So the main concern for the Sonera Mobile Pay department is to gather and handle the business area for a MPP and to verify that the MPP design is filling the requirements generated by the business.

According to the factors presented in Section 5.4.2 the following factors can be obtained from the MPP concept:

1. Requirement catalogue contains 153 user requirements.
2. Estimated system lifetime is long. This system should be one that can be updated as needed so that the platform should be alive after at least a decade. This will generate a need to know what was implemented in each version and what has changed. There is also a need to find out what will change when some old requirement changes in later versions.
3. Level of organisational maturity is low. The SMP-department was generated from a scratch for a whole new business area. This means that there are no previously implemented practises.
4. Project team size and composition. The department contains roughly three main areas. The First one is the business staff who are selling the concept and gathering new customers. The second one consists of the technical workers who are working as engineers and making requirements out from the business requirements and are also responsible for verifying that the design made by other partners will fit together and form a solid platform. The third staff is formed of the people who are responsible for implementing the concept when it is ready for production.
5. Type of the system is a user oriented commercial system which will handle small billing transactions. Failures will cause possible loss of finance and bad

design can give discomfort while used resulting into smaller usage.
Transactions with money will be quite small but there will be a lot of them.
Small streams can build up a big river if errors start to cumulate.

Sections two and four indicate that the traceability should be maintained quite strictly. From the sections one and three we can see that while the business is just beginning, it is not catastrophic even if there are no currently working traceability techniques. That is because a small number of requirements and a new organisation which does not have strict practisis hard to change. Section five gives also a little time because it states that we do not deal with human lives. For the future it seems wise to at least make preparations for practises ready for implementation.

6.3.2 STRUCTURE

The traceability manual should now take a form, otherwise the information cannot be stored. Figure 20 presents a possible structure and a discussion of how it should be used.

1 Preface
1.1 Revision history
1.2 Contents
1.3 Purpose
1.4 Terms, acronyms and abbrevations
1.5 Guidelines with the document
1.6 References
2 Project policies
2.1 Requirements
2.2 Dependencies
2.3 Source
2.4 Rationales
2.5 Tests
2.6 Development
2.7 Descriptive information
2.8 Resources
3 Traceability information
3.1 …

Figure 20. Structure for a traceability manual.

Chapter 1 is a preface and it contains information of the document in much the same way as in the first chapter of the requirements catalogue structure in Figure 19. This is also trace information for the document. The revision history will show the previous versions and their numbering before the current document. The contents will help the reader to find interesting parts in the document. The purpose will tell why this document has been created. Terms, etc. will help understanding e.g. the shortcuts. Guidelines will have information assisting with the use of the document and a references list for further reading.

Chapter 2 contains traceability policies used with the project the document belongs to. It has been structured so that first there is a chapter on requirements. This is used to define how new requirements are approved to existent requirement catalogue and in what format they should be written down. This is followed with seven chapters which each define a different type of relations associated with requirements (see Chapter 5.2). These relations can be shown with traceability techniques (see Chapter 5.3 for available traceability techniques). In Figure 21 there is a suggested list of techniques that can be used to show the relation with the MPP project.

	Guide how to implement
Requirements	Attach the requirement with unique id, author, source, description
Dependencies	Use tables or automated links
Source	This should be written down while collecting the requirement.
Rationales	This connection with functional<->non-functional requirement is implemented with automated links or with tables.
Tests	When tests are made up tables should be used to check that all requirements are tested. Later these tables can be used to check what each test was actually testing from a requirements catalogue.
Development	This contains three areas, architecture, design and interface. A good structure with requirements catalogue should clear this quite much at first place.
Descriptive information	Use references to show where to find more information.
Resources	Here should be defined who are working with special tools, for example with the doors if they are implemented.

Figure 21. Guide how to implement different kinds of traceability.

Chapter 3 contains requirement information. Traceability information is presented in such way as defined under policies in Chapter two. This information can be references to documents which contain traceability information, or it can be the traceability information or combination of both.

7 CONCLUSIONS

The main results achieved during this study were the process model for requirement analysis, the template for collecting requirements and clarifying what traceability means. These were also mostly due to my own research and observation. Other result was a principle which has also been told to be the KISS principle (Keep It Simple Stupid). This is an informal and unscientific principle but it reminds to forget about too much complexity. In conclusion I will also clarify some thoughts about it.

It seems that in theory its quite easy to produce models for requirement analysis. The hard part is to find a level where the models are right. The work models should not get on a level where they are too detailed. Too many details will make them cumbersome, because there is a level where people just act and do not think. This is the same as acting in a spirit of void which is told to be the principle how things should be done in Musashi's book. In my opinion it can be said that people can be beginners, novices and professionals. This is very rough.

Beginners are people who are just studying the general skills needed or have never heard of them. In terms of warfare this would mean that they are learning the tactics which is the same as small movements and how they are carried out in the right way. In computer science this means that they are practising how the program is written and what languages there are to choose, etc. Larger parts are also revealed but they cannot be tried out in detail. One can read about managing a big company or producing a huge software project but in practise it is not possible to try it out in this phase. A lot of studying is needed.

Novices are people who know what should be done and the general way how to do it but they are not certain of all things. They need support and assistance. This is the case when one starts to work in a workplace and does not yet know how things are

done there. Here the worker starts to learn how to adapt small things into a bigger scale. Studying continues, not so much in theory but also in practise by trying out and looking how things just seem to be carried out.

Professionals are people who have the knowledge and who have found out the variations or many of them which can take place. Professionals have the ability to adapt to new situations fast because they do not need to think so much. In other words they can see the whole strategy and just act according to it.

The models introduced should be on a level where they guide beginners and do not restrict the professionals. It could be said that documented models and rules should be on a level where they define the strategy that is followed and leave the workers freedom to follow it. So the greatest benefit of models and templates is when new employees are introduced to work. Documentation should help to sew skills into strategy to achieve a uniform way for a company.

8 REFERENCES

1. Sven Dahlman, User requirements a resource for the development of technical products, Chamlers Tekniska Högskola, Department of Consumer Technology Göteborg 1986.
2. Ilkka Haikala&Jukka Märijärvi, Ohjelmistotuotanto, 3.painos,ISBN 951-762-497-2, Gummerus Kirjapaino Oy Jyväskylä 1997.
3. Miyamoto Musashi, Maa,Vesi,Tuli,Tuuli ja Tyhjyys, 3.painos,kustannusosakeyhtiö Otavan painolaitokset, Keuruu 1995, ISBN 951-1-07526-8.
4. Ian Sommerville&Pete Sawyer, Requirements engineering, a good practice guide, reprinted July 1998, ISBN 0 471 97444 7.
5. Richard Stevens, Peter Brook, Ken Jackson & Stuart Arnold, Systems Engineering , coping with complexity, 1998 edition published by Prentice Hall Europe.
6. Sonera R&D Process Model, revision 1.1.
7. INSKO. Vaatimusmäärittely järjestelmän suunnittelussa, 2. Painos Helsinki, Insinöörijärjestöjen koulutuskeskus INSKO 1991. (INSKO-julkaisu 124 - 90).