Applying QFD to improve the requirements and project management in small-scale project

Terhi Kivinen

University of Tampere
Department of Computing Sciences
Computer sciences
M.Sc Thesis
Supervisor: Zheying Zhang
May 2008

Abstract

University of Tampere
Department of Computing Sciences
Terhi Kivinen: Applying QFD to improve the requirements and project management in small-scale project
Master's Thesis, 67 pages, 6 appendix pages
May 2008

Quality is one of the key factors in software engineering business. Since errors, defects and deficiencies in requirements cost considerably more to correct than errors of later phases of software development there is a lot of interest to improve the quality of requirements.
This master's thesis introduces Quality Function Deployment (QFD) matrix tool as a method to improve the quality of requirements and help project management in a small-scale project. QFD is a method of customer oriented product development and it has been used a lot in various industries.
In the theoretical discussion part of thesis different concepts connected to the quality of software are presented and their characteristics are discussed especially from point of view of a small project.
The empirical part of this study describes first a study case of a software project called Code register. The scope of the project was to rebuild a Code register -software used in a large public administration organization in Finland. Secondly the empirical part concentrates on using QFD in requirements analysis and describing the result of using it.

Keywords and terms: Quality Function Deployment, Small-scale project, Software engineering

---

Index

1. Introduction 1
   1.1. Background 1
   1.2. Motivation for the research 2
   1.3. Objectives and methods 2
   1.4. Main concepts and the structure of the master's thesis 3
2. Small scale project 4
   2.1. Software development processes 6
   2.1.1. Waterfall model 7
   2.1.2. Incremental models 8
   2.1.3. Evolutionary process models 9
   2.1.4. Specialized process models 11
   2.1.5. The Rational Unified Process 12
   2.1.7. Process model in a small scale project 14
   2.2. Requirement analysis in software project 15
   2.3. Project management process and small scale project 18
   2.4. What makes a project successful? 22
3. Quality 25
   3.1. Quality as a concept 25
   3.2. Software quality 26
   3.3. Quality in requirement analysis 31
   3.3.1. Prioritization of requirements 32
4. QFD 34
   4.1. General 34
   4.2. The Four-Phased approach 35
   4.3. Types of requirements 36
   4.4. House of Quality 38
   4.4.1. Customer requirements 38
   4.4.2. Customer prioritization 38
   4.4.3. Technical requirements 39
   4.4.4. Requirements correlation 39
   4.4.5. Technical feature comparison 39
   4.4.6. Design Targets 39
   4.5. QFD in software engineering 40
   4.6. Software tools of QFD 41

5. QFD in a small software development project – a case study 42
5.1. Introduction to code register 42
5.2. Motivation to use QFD in code register project 44
6. House of Quality in use 46
6.1. Description of the work flow 46
6.2. Stakeholders of Code register 48
6.3. Determining the customer needs 49
6.4. Determining technical requirements 51
6.5. Composing the D part of the House of Quality matrix 52
6.6. Technical feature comparison 55
6.7. Analysing the results 56
7. Summary 57
7.1. General experiences of QFD 57
7.2. QFD as a method to improve requirements 58
7.3. QFD as a method to help project management 59
7.4. Workload of QFD 59
7.5. QFD and software process model 60
7.6. Generalization of the results 61
References: 62
Appendix A: Customer requirements from previous project 68
Appendix B: The Customer requirements in Tree diagram 72

1. Introduction

1.1. Background

Creating a software system using a software engineering process contains three main tasks or phases: the functions and features of the expected the software has to be defined, the software has to be implemented and it has to be deployed in an operating environment.

Most software systems are developed as project since a software system is usually unique and the software should be produced in a certain time limit and resources. The sizes of projects vary a lot. There are also a lot of different methods and different process models to manage the projects but the common feature is that phases mentioned above are always included in to a software system development project even though they usually are divided and specified into more detailed tasks.

The functions and features of the expected software are called requirements. The word requirements has been defined more precisely by Kotonoya and Sommerville [1998] as a description of how the system should behave, application domain information, constraints on system's operation or specifications of a system property or attribute. Requirement analysis is the phase of software development where feasibilities studies are made, competitors and existing systems are examined and the new system is specified.

In spite of new and effective software engineering techniques, according to various researches a lot of software development projects tend to fail. A result of a survey in which 1500 IT project managers were questioned across the UK in all industry sectors Huber [2003] stated that only 16 % of project managers examined at the survey had succeeded in all their targets: schedule, budget and scope/functionality. The three main reasons for the failure in projects were:

- Lack of User Input
- Incomplete Requirements & Specifications
- Changing Requirements & Specifications

The most critical area of the software development project is the acquisition and management of the requirements.

There is also a lot of evidence that the errors in requirements are the most expensive errors to fix during the project. And the errors in requirements tend to become more expensive to repair the later they are found. Davis [1993] has estimated that an error in requirements costs 200 times more to fix during the maintenance stage of the system compared to fixing it during the requirements analysis phase.

On the whole the acquisition of requirements and managing them during the software project is one of the key factors to make the project successful. The lack of quality in requirements means problems and rework in all following phases of the project. Additionally most projects are balancing between the requirements and the lack of time if the requirements are not defined and prioritized carefully it's difficult to make right decisions if some requirements have to be left undone if there is shortage of resources or time during the project.

Consequently there is a lot of interests to have as good quality in requirements as possible. There are a lot of different techniques to improve the quality of requirements. Nowadays the process how the requirements are acquired almost without exception includes customer participation.

Quality Function Deployment (QFD) is a set of matrix tools, which are used in product development in deploying customer requirements into a product, service and business operations [Liu, Sun, Cane, 2005]. It's been said that it really doesn't matter if the project team think that the final product was unsuccessful. What really matters is the voice of the customer: if he finds the product suitable then it was successful. The principle of QFD is similar - the idea is to take the customer into the product development process to ensure that the core quality will be in the product or service - the customer satisfaction.

The central tool of the Software Quality Function Deployment (SQFD) is the matrix chart called House of Quality. Also known as a quality chart, this tool is a powerful way of generating specific, prioritized, and measurable technical requirements from often ambiguous customer needs.

1.2. Motivation for the research

Requirements acquisition and managing them during the software project is one of the key factors to make the project successful. Good quality in those phases is most profitable. Originally QFD has been developed to manufacturing industry to improve quality but lately is has been used also in software projects [Zultner, 1992].

A typical project has to produce a certain result within limited resources and time. Very often it turns out that the planned results can not be reached within planned time or resources. The project manager has to make decisions in project: is it possible to use more time, is it possible to get more resources or which requirements can be left out? If the time period can not be extended and more resources are not available the only way is to leave some requirements undone. If the requirements are not carefully acquired and prioritized the decision which requirements can be left out rely more on instincts than facts.

1.3. Objectives and methods

The scope of the research is to clarify what kind of difference it makes to the project to use QFD especially the House of Quality model during the requirements analysis phase in a small-scale project. How much extra work does it entail? Is it practical to deploy QFD in a small-scale project? How does it fit to a project which implements water fall process model? And finally how does it help the project manager to make decisions during the project and does it improve project quality.

The strategy to be used in this study to research QFD in software project requirement analysis phase is case study. A Case study is an empirical inquiry that investigates a phenomenon within its real-life context [Perry, Sim and Easterbrook, 2004]. The chosen case is a paradigmatic type of case, which may be defined as an exemplar or prototype. The purpose of the study is to evaluate QFD as a software requirement analysis tool in a small scale project as proposed by Haag, Raja and Schkade [1996] who limited the focus of QFD to requirements engineering. The case study is performed with a single case.

1.4. Main concepts and the structure of the master's thesis

The most important concepts used in this study and their relationships are illustrated in Figure 1.

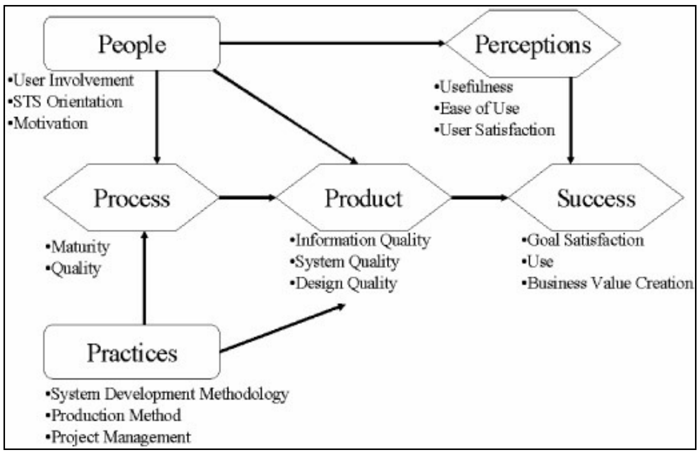

Figure 1. Information systems quality model [Duggan and Reichgelt, 2006].

The most important concept is the quality of software. There are a lot of factors which define the quality of software: reliability, usability, etc and also multiple perceptions of quality. There are also other of factors and forces which collectively affect the quality of software. These factors and their multidimensional effect on each other are illustrated in Figure 1.

The software development process and product it produces are both impacted by the process management practices (Practices) employed and by people. Practices include the existence and usefulness of a software development methodology, the appropriate choice of a software production method, and the effectiveness of project [Duggan and Reichgelt, 2006]. People issues encompass the degree of user involvement, the application of socio-technical systems principles, and the motivation of participants in the delivery process. The factors the affect success are the expectations of users and the quality of product.

QFD is a requirements engineering approach that focuses on quality [Haag, Raja and Schkade, 1996]. Combining QFD to the concepts or quality factors in Figure 1 it affects all of them but implementing it most directly to People and Practices. Thus this study concentrates mainly on left part of the picture: factors that affect process management: especially software development process, project management and user involvement. It is stated that a product’s quality is improved when the software development process used to make the product is improved. Each development process contains several phases and this study concentrates on requirements engineering and management.

Software project management also affects the quality of software. The chosen software development process model and the practices of requirement engineering and management have influence on choosing the management practices. The concepts are described in next chapters in detail.

The first chapter of the study is introduction and it offers a general idea and main concepts to the topic.

Second chapter explains concepts and process models that are used in project work. The point of view is on small scale projects and the process models are explained in general and evaluated according to their suitability to small scale projects. The phase of requirements acquisition and evaluation is explained as well as factors which affect the success of the project.

Third chapter is about quality: in general and in software project work. Special emphasis is on the quality of requirements of software. Fourth chapter is about QFD. Its history is described shortly and also the main method of QFD, the House of Quality is described in detail. Also former uses of QFD in software processes are explained and some software tools are described briefly.

The objectives and methods of this study are discussed in fifth chapter. The case code register is introduced and the motivation for choosing it.

The sixth chapter includes descriptions of the use of QFD. The seventh chapter summarizes the results of this study.

2. Small scale project

The definition of the size of a project is not precise: there is no accurate, unambiguous way to define criteria to indicate the size of a project nor there is coherent understanding how the projects should be categorized by size: what is meant by a small scale project.

The most commonly used criterion to express the size of a project is the amount of work what is needed to produce the software. The other commonly used criterions to compare the sizes of projects are the size of the budget and the number of people required to develop the software. Yet these criterions are affected by the amount of work.

The most evident element that determines the size of a software project is the size of the software to be developed. But there are also other elements that affect the size of the project by affecting the workload namely what kind of software is being developed, personnel factors, programming language, other project influences and diseconomies of scale [McConnell, 2006]. The type of project - maintenance, implementing a package system or creating new software - also affects the workload. The amount of reusable elements (source code, documents, test material) is another factor that affects the size of the workload [Forselius, 1999].

There are several methods to express the size of the software. One of the most used methods is the amount of lines of source code (LOC, SLOC, KLOC). The amount of logical sentences is closely related to lines of source code but it is more difficult to obtain. The amount of document pages can also be used to describe the size of the software if the methods and standards to produce the material are precise. Functional Point Analysis (FPA) is quite commonly used method to determine the size of the software. There is a lot of different versions and variations of FPA. Backfiring is a method where the size of the software is estimated combining FPA and SLOC [Forselius, 1999]. COCOMO and its different versions are the most commonly known backfiring methods.

Requirements count is also a way to measure the size of a project [Potter, 2005]. It is not a precise method but the magnitude of requirements to be implemented is an important indicator of the size of the project. If the requirements are not uniform in scope a complexity or weighting factor should be assigned to make requirements count larger when requirements are larger. UML use case metrics or the number of screens on a user interface is also fairly simple methods to estimate the size of a software project.

When using the size of software as a criterion, the most important factor in deciding which methods to use is the amount of information available; in other words the phase at which the project is [Forselius, 1999]. After preliminary analysis usually the only available method is to compare the project to other same kind of projects. The result is very vague because there are never exactly similar projects and the experience of the person doing the estimation is crucial.

After requirements analysis the requirements can be used in determining the size of software based on the requirements. The number of requirements gives vague idea of the size but if using UML the use case metrics or the number of screens on a user interface is more precise and quite simple methods to estimate the size of a software project. If not using UML different versions of FPA are useful and yet more time consuming. COCOMO method requires even more work but the result can quite accurate. The amount of documentation is useful only if the development process is stable and there is enough projects with which the amount of documentation can be compared to.

When the software is implemented the amount of source code can be used to measure the size of software.

Another way to determine what is a small scale project is to gather some characteristics of it. A small scale project has short schedules, limited resources, small project team and only few external interfaces.

As stated before there is no explicit, unambiguous method to determine the size of a project. Different methods in reality give only different aspects of the size of the project. According to ESA Board for Software Standardisation and Control [1996] a software project can be considered to be small if one or more of the following criteria apply:

· less than two man years of development effort is needed

· a single development team of five people or less is required

· the amount of source code is less than 10000 lines, excluding comments.

SMS [2004] has stated that a software project which requires less than 1500 work hours should be considered small.

2.1. Software development processes

A process framework establishes the foundation for a complete software process by identifying a small number of framework activities that are applicable to all software projects, regardless of their size or complexity [Thayer and Christensen, 2005]. The main activities are applicable to the vast majority of software projects and they are:

- communication with stakeholders which includes requirement acquisition
- planning
- modelling
- construction
- deployment.

Software engineering incorporates a development strategy that encompasses the process, methods, and tools. This strategy is often referred to as a software process model or a software engineering paradigm. A software development process can also be defined as a set of activities needed to transform the user requirements into a software system [Jacobson, Booch and Rumbaugh, 1998].

Thayer and Christiansen [2005] divide the process models into two types: prescriptive and agile software development. Prescriptive software process models prescribe a set of process elements: framework activities, software engineering actions, tasks, work products and quality assurance and change control mechanisms for each project. Each process model also prescribes a workflow, that is, the manner in which the process elements are interrelated to one another.

All prescriptive process models accommodate the earlier mentioned main activities, but each applies a different emphasis to these activities and defines a workflow that invokes each framework activity as well as software engineering actions and tasks in a different manner. There are a lot of different kinds of development processes and some of them are just combination of features from the others. Thayer and Christensen [2005] classify the prescriptive process models into five types: waterfall models, incremental process models, evolutionary process models, specialized process models and the rational unified process (RUP).

Traditional process models tend to be inflexible to changes which helps them maintain a predictable schedule, but it does nothing to ensure that the final results meets the customers real, changing needs. Agile software development aims to flexibility and adjustability [Koch, 2005]. It emphasizes individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation and responding to change over following a plan. Commonly used agile process model types are Extreme Programming (XP), Scrum, Adaptive Software Development, Feature Driven Development and DSDM but there are several others too.

The classification of process types according to Thayer and Christiansen [2005] is not quite clear: many process types are using features of the others. For example most agile process types are using features of incremental and evolutionary processes.

2.1.1. Waterfall model

Waterfall model is sometimes called the classic life cycle, because it is the oldest model that has been used in software development, introduced already in 1970 [Connors, 1992]. It suggests a systematic, sequential approach to software development that begins with customer specification of requirements and progresses through planning, modelling, construction, and deployment, culminating with on-going support of the completed software.

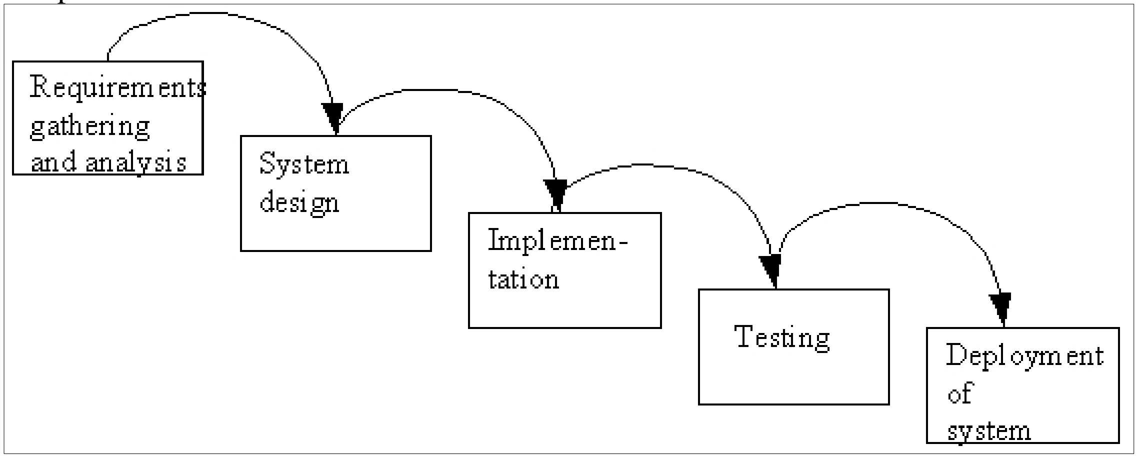

Figure 2. The Waterfall model.

Waterfall model has been identifies as an inefficient model to use and it is inflexible. Especially if the project is long the requirements tend to change because of the changing environment. It is suitable for small scale projects if requirements are well defined and reasonably stable. Even then it requires patience from the customers, because a working version of the program will not be available until late in the project time span. A major blunder, if undetected until the working program is reviewed, can be disastrous. Typically the work in a small software project is fast-paced and subject to a never-ending stream of changes to features, functions and information content. The waterfall model is usually inappropriate for such work.

2.1.2. Incremental models

The incremental model combines elements of the waterfall model and applies repetition in an iterative fashion. The incremental model contains linear sequences as calendar time progresses. Each linear sequence produces deliverable "increments" or versions of the software which have increasing amount of features and functions as the number of increments grow. The process flow for any increment may incorporate the prototyping paradigm so the result of each increment can be a new version of a prototype.

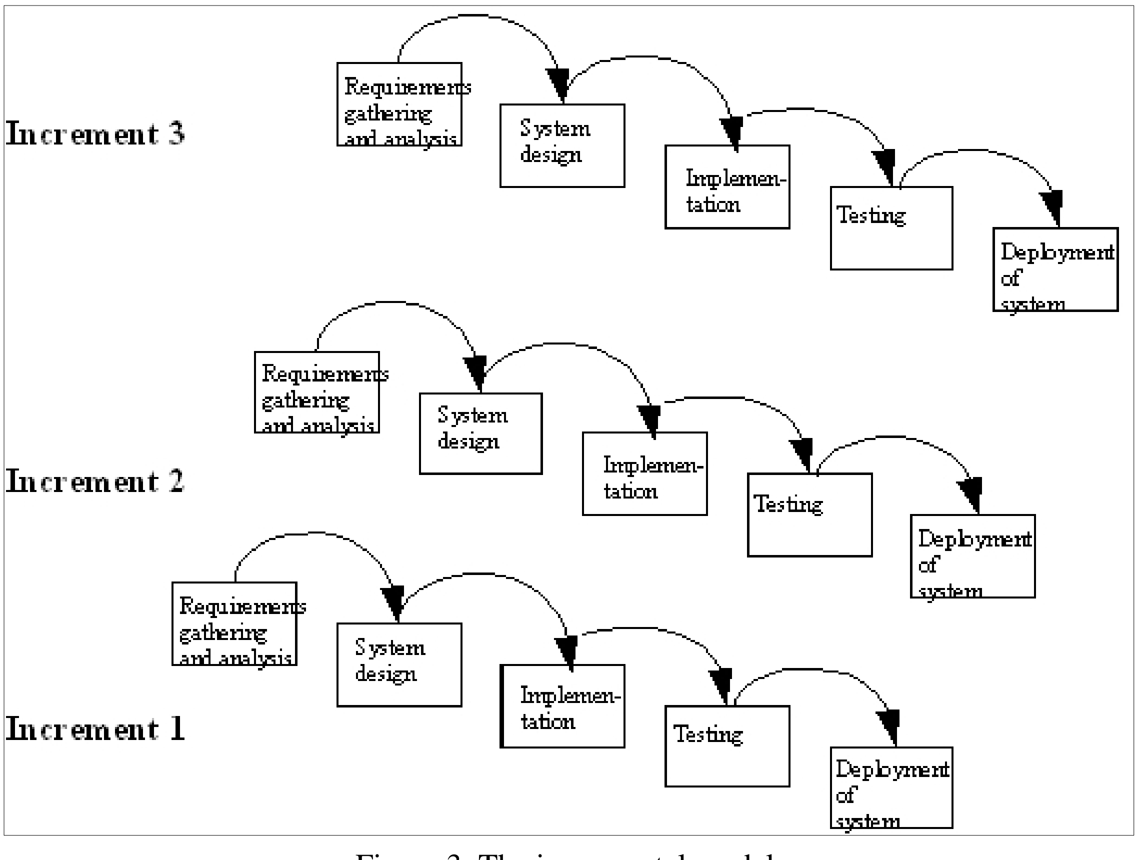

Figure 3. The incremental model.

Incremental development is particularly useful when staffing is unavailable for a complete implementation by the business deadline that has been established for the project. Early increments can be implemented with fewer people [Wysocki, 2006]. If the core product is well received, then additional staff (if required) can be added to implement the next increment. In addition, increments can be planned to manage technical risks or new architecture. The first increments are built with concentrating in technically or architecturally risky parts with minimal functions [Thayer and Christiansen, 2005].

Thus incremental model is suitable also to a small scale project if staffing is unavailable, limited software functionality to users has to be provided quickly or there are technical or architectural features that need to be tested.

2.1.3. Evolutionary process models

Evolutionary models are iterative. During the process the software engineers develop increasingly more complete versions of the software. Often, a customer defines a set of general requirements for software, but can not identify them in detailed. In other cases, the developer may be unsure of the technical or architectural solutions. In these and many other situations, a prototyping paradigm may offer the best approach [Thayer and Christiansen, 2005].

Although prototyping can be used as a stand-alone process model, it is more commonly used as a technique that can be implemented within the context of any one of the process models. Prototyping provides a communication basis for discussions among all groups involved in the development process, especially between users and developers. In addition, prototyping enables to adopt an approach to software construction based on experiment and experience [Lichter, Schneider-Hufschmidt and Züllighoven, 1993].

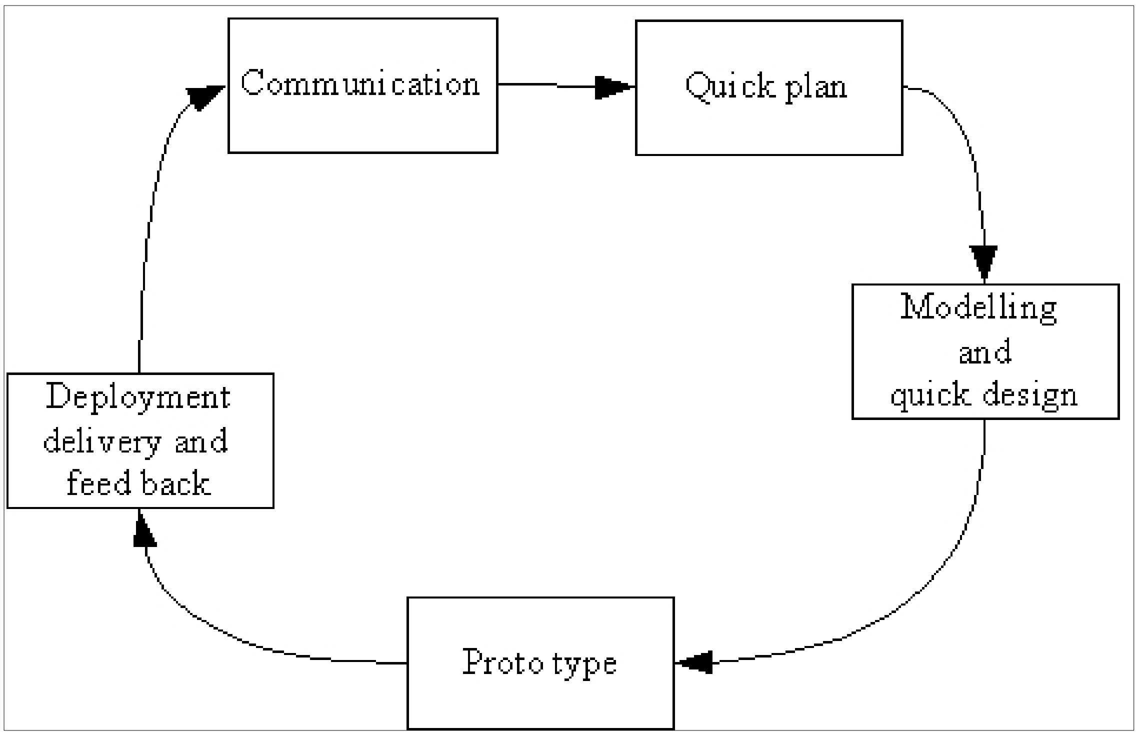

Figure 4. The prototyping model.

The prototyping paradigm begins with communication. The software engineer and customer meet and define the overall objectives for the software, identify whatever requirements are known, and outline areas in which further definition is mandatory. Prototyping iteration is planned quickly and modelling occurs. The quick design focuses on a representation of those aspects of the software that will be visible to the customer or end user or are important to verify the architectural or technical solution. The quick design leads to the construction of a prototype. The prototype is evaluated by various stakeholders and feedback is used to refine requirements for the software to be developed.

In Throwaway prototyping model the development of the prototype is constructed during the design phase and after the evaluation of the prototype the implementation phase will start [Leon, 2000]. During the iteration the prototype produced incrementally and after each evaluation new features are added to it.

The prototyping and evolutionary process model in small scale project is justified especially if there is uncertainty of technical or architectural features, if details of the requirements are unclear or stakeholders have difficulties to comprehend the documentation of requirements. The disadvantage of prototyping is that sometimes it is difficult to stop it and produce the final product.

The spiral model was proposed by Boehm [1986]. It is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model. It provides the potential for rapid development of increasingly more complete versions of the software.

Using the spiral model, software is developed in a series of evolutionary releases. During early iterations, the release might be a paper model or prototype. During later iterations, increasingly more complete versions of the engineered system are produced.

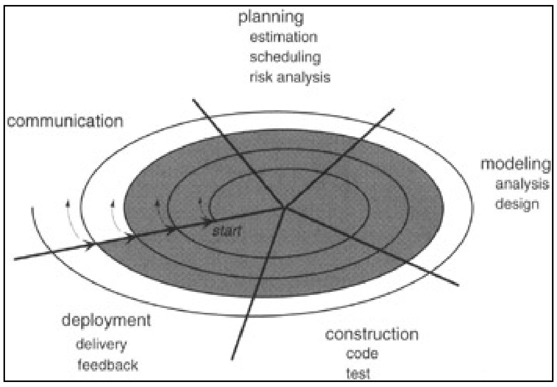

Figure 5. A typical spiral model [Thayer and Christiansen, 2005].

The first circuit around the spiral might result in the development of a product specification and subsequent passes around the spiral might be used to develop a prototype and then progressively more sophisticated versions of the software. Each pass through the planning region results in adjustments to the project plan. Cost and schedule are adjusted based on feedback derived from the customer after delivery. In addition, the project manager adjusts the planned number of iterations required to complete the software.

The spiral model is a realistic approach to the development of large-scale systems and software. Because software evolves as the process progresses, the developer and customer better understand and react to risks at each evolutionary level [Tomayko and Hazzan, 2004]. The spiral model uses prototyping as a risk reduction mechanism but, more important, enables the developer to apply the prototyping approach at any stage in the evolution of the product. It maintains the systematic stepwise approach suggested by the classic life cycle but incorporates it into an iterative framework that more realistically reflects the real world. The spiral model demands a direct consideration of technical risks at all stages of the project and, if properly applied, should reduce risks before they become problematic.

2.1.4. Specialized process models

Special process models take on many of the characteristics of one or more of the models described in the previous sections. These models tend to be applied when a specialized or narrowly defined software engineering approach is chosen and they can also be characterized as a collection of techniques or a methodology [Thayer and Christiansen, 2005].

The component-based development model commercial off-the-shelf (COTS) software components, developed by vendors who offer them as products, can be used when software is to be built. The model incorporates many of the characteristics of the spiral model. It is evolutionary in nature, demanding an iterative approach to the creation of software. However, the component-based development model composes applications from pre-packaged software components. Small scale project can typically be this kind of projects.

The formal methods model encompasses a set of activities that lead to formal mathematical specification of computer software. Formal methods enable a software engineer to specify, develop, and verify a computer-based system by applying a rigorous, mathematical notation. Cleanroom is one variation of formal methods and it is used by some software development organizations. The Cleanroom concept depends on application of statistical quality control techniques, a structured architecture, a specification language based on a mathematical approach for specifying the functional requirements, elimination of unit or module testing, and emphasis on usage testing or functional testing. This method uses mathematical and statistical techniques to specify correctly and precisely the functional and performance requirements. Then a structured architecture and design methodology is used to build the various functional components that are integrated without unit testing. The testing that is conducted is usage testing where the performance of the components under actual usage or operational conditions is tested [Leon, 2000].

When formal methods are used during development, they provide a mechanism for eliminating many of the problems that are difficult to overcome using other software engineering paradigms. Ambiguity, incompleteness, and inconsistency can be discovered and corrected more easily, not through ad hoc review, but through the application of mathematical analysis. When formal methods are used during design, they serve as a basis for program verification and, therefore, enable the software engineer to discover and correct errors that might otherwise go undetected.

Although not a mainstream approach, the formal methods model offers the promise of defect-free software. Yet considering especially small scale projects the development of formal models is currently quite time-consuming and expensive and because few software developers have the necessary background to apply formal methods, extensive training is required. But the formal methods approach is useful in software projects which build safety-critical software e.g. aircraft avionics projects and medical devices projects and among projects which can lead into severe economic hardship if software errors occur.

2.1.5. The Rational Unified Process

The Rational Unified Process (RUP) is a heavyweight object-oriented software development process. It emphasizes the adoption of certain best practices of modern software development. RUP is an iterative process and it recognizes five phases in the software development process.

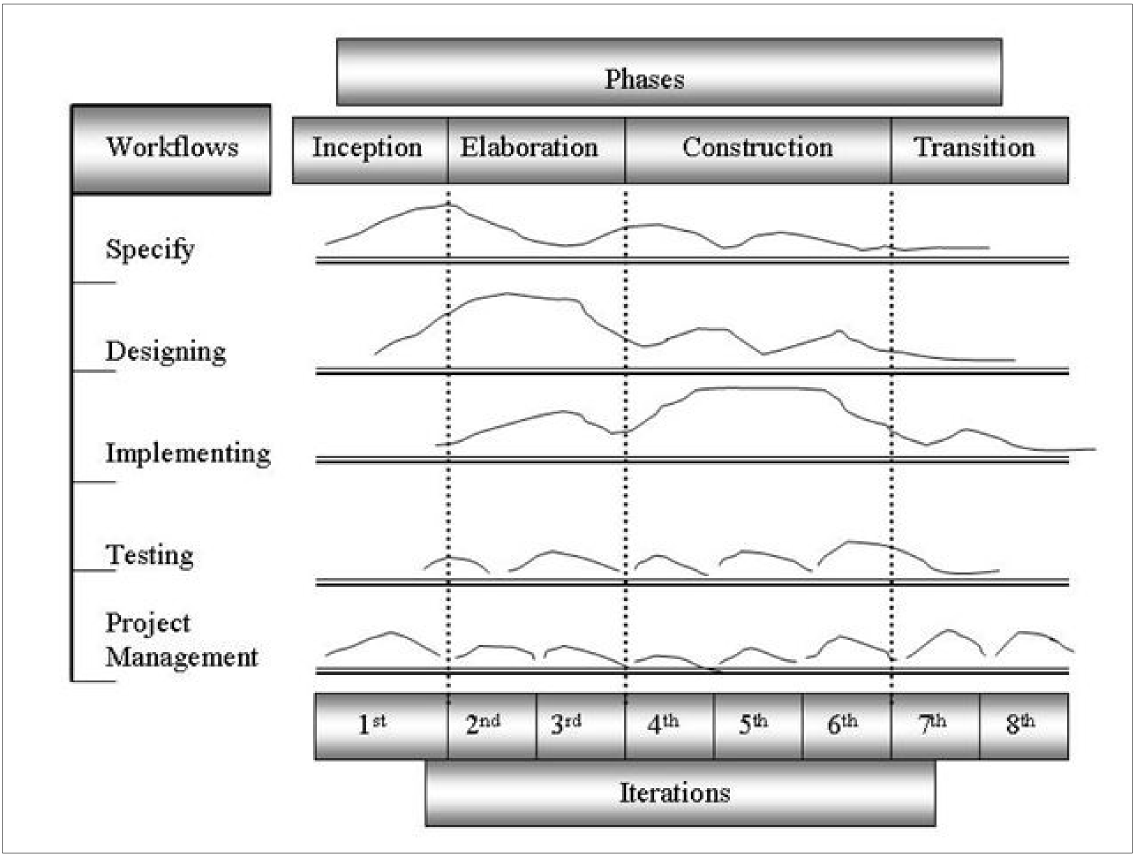

Figure 6. The Unified Process [Tomayko and Hazzan, 2004].

During the inception project scope and the business case are defined. The architectural foundation for the system is defined during the elaboration phase. During the construction phase the system to be deployed will be finalized and during the transition phase the system will be delivered.

The RUP has several strengths [Ambler and Constantine 2000]. First, it is based on sound software engineering principles such as taking an iterative, requirement-driven, architecture-based approach to development in which software is released incrementally. Second, it provides several mechanisms, such as a working prototype at the end of each iteration and the "go/no-go" decision point at the end of each phase, which provides management visibility into the development process.

RUP is suitable for very large software development projects undertaken by development teams of many people. It is not entirely suitable for small projects because it is quite complex and time consuming. Many small scale projects use only some features of the RUP process. E. Trengove and B. Dwolatzky [2004] have defined a software development method which combines quality management system and software design process. They combined some features of ISO 9001 software management standard and RUP.

A hybrid method of RUP, OPEN (Object-Oriented Process, Environment, and Notation) and XP (Extreme Programming) is also developed for small scale projects. It combines good practises from XP fitting with RUP like phases abiding by meta-model similar to OPEN and is more applicable and easily bearable to small scale projects. [Al Masum, Morshed and Mitsuru, 2004]

2.1.6. Agile process methods

Although some of the Agile methods have existed in one form or another for some time, the term Agile Method became known as light methodologies in 2001. There several different agile methods of which the Extreme Programming (XP) is a collection of 12 practices that focus specifically on the mechanics of developing software. Some of the practices of XP, many are humane like the 40-Hour Week -practice [Tomayko and Hazzan, 2004]. These practices include also such topics as the planning game, pair programming, refactoring, and testing [Koch, 2005].

Scrum is primarily a product development method. It specifies different predefined roles and seven practices which focus on planning and managing a development project but do not address any specifics about software. Therefore, it can be used in conjunction with any software development method [Koch, 2005]. Scrum is based on small teams and it enhances communication between team members.

Adaptive Software Development (ASD) is based on complex adaptive systems theory and treats software development as a collaborative learning exercise. Dynamic System Development Method (DSDM) is mainly a philosophy about system development that consists of nine principles. DSDM focuses on system development and does not get into the details of writing software, so it can be used in conjunction with any of the more software-intensive Agile methods, like XP. Feature-Driven Development (FDD) treats software development as a collection of features that are implemented one at a time. Unlike the other Agile methods, FDD includes upfront architectural analysis, such as the development of a Domain Object Model, which becomes the basis for planning the project iterations. It also includes a mechanism for objectively reporting progress against plan.

2.1.7. Process model in a small scale project

When no software development model is used at all, software development can only be thought of as a black box relative to the system development process. This is sometimes also called the Code-and-Fix model. With this model written specification usually doesn't exist. Coding begins immediately, and bugs are fixed as they are discovered [Schmidt, 2000].

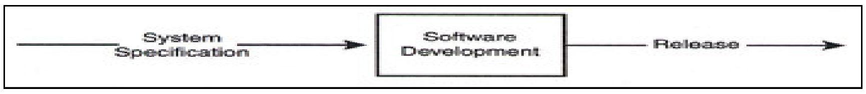

Figure 7. Black box software development.

Software process models are designed to help to manage the project, avoid the risks and improve quality to project. In a way the process creates a structure to a project. Different process models stress different features of software development and many of them include features or even are based on each other. One perspective to process models is that they offer tools to manage the cost of change during a project [Mangione, 2003]. Processes that follow waterfall and iterative models control costs by reducing need for change as costs increase. In contrast, processes based on the spiral model ensure that the cost of change is fixed.

A process model for software engineering is chosen based on the nature of the project and application, the methods and tools to be used, and the controls and work products that are required [Thayer and Christensen, 2005]. As project complexity increases, the likelihood of nailing requirements decreases. This follows logically from the fact that the human brain can retain in memory only about seven pieces of information [Wysocki, 2006]. Also as project complexity increases, so does the need for process flexibility. Increased complexity brings with it the need to be creative and adaptive. Neither is comfortable in the company of rigid processes.

Nowadays it is generally accepted that the attendance of customer to project work is important factor to improve quality. In agile methods because of iterativeness the customer attendance continues till the end of the project and is more in favour of face-to-face communications. Because in small projects there is less people involved and the duration is shorter it is easier to have face-to-face contacts and customer involvement.

The bigger a software project is the more it relies on process because the duration of the project is longer which generate more changes to environmental and customer requirements. Yet the development process is necessary also in small scale projects. It helps to plan the project, gives structure to it and helps to avoid duplicate work. The challenge in choosing the right development process model for small scale project is to find a model that is not too bureaucratic but has enough tools to produce good quality. The use of agile methods has increased especially in complex and in projects where requirements or technology in uncertain. But waterfall model is still very useful in small projects since they usually are more short term thus the requirements are more stable.

In waterfall and prototyping model all the requirements are acquired and analyzed in one phase before starting the implementation phase. In spiral, incremental and RUP model and agile methods the amount of requirements grows as the project advances.

2.2. Requirement analysis in software project

In reference to last chapters it is evident that regardless of the software development process model requirement acquisition and analysis is a part of the process. Sommerville and Sawyer [1997] have defined requirements as a specification of what should be implemented. Requirements are descriptions of how the system should behave or of a system property or attribute and they may be a constraint on the development process of the system.

Requirements acquisition and management can be seen as one of the most important processes during the project. Chris Sauer and Christine Cuthbertson from Oxford University's Template Collage led a survey in which they questioned 1500 IT project managers across the UK in all industry sectors [Huber, 2003]. Only 16 % of project managers examined at the survey had succeeded in all their targets, schedule, budget scope/functionality. Only 55 % of projects had completed on time and 54 % of projects failed to deliver a system with all the planned functionality and 9 % of the projects were totally abandoned.

In 1997 a survey questionnaire focusing on IT project management issues was sent to Canada's leading 1450 public and private sector organizations [Whittager, 1999]. Project failure was defined in three ways: overrunning its budget or schedule by 30% or more or failing to demonstrate the planned benefits. 61 % of the projects that had given response reported failure. Of the failed projects 87% overran schedule, 56% overran budget and 45% didn't meet requirements.

There are also a lot of surveys that show that the errors in requirements are the most expensive errors to fix during the project. Typically the errors in requirements also tend to become more expensive to repair the later they are found. Davis [1993] has estimated that an error in requirements costs 200 times more to fix during the maintenance stage of the system compared to fixing it during the requirements analysis phase.

Requirements engineering is the systematic process of developing requirements through an iterative co-operative process of analysing the problem, documenting the resulting observations in a variety of representation formats and checking the accuracy of the understanding gained [Pohl, 1993].

The elicitation of requirements aims at discovering, revealing, articulating, and understanding the problem to be solved, the system services, the required performance of the system, hardware constraints and security needs of the system. The most important and most used way to elicit the requirements of a system is to consult with stakeholders or observe them at work. In order to do that systematically the identification and analysis of stakeholders – their relationships, importance and influence - has to be done first. Stakeholders include the parties that can influence or be affected by the new system, which is why they have an interest in the system. Stakeholders of a system can be grouped into six categories: end users, customers, domain experts, regulators, developers and neighbouring systems. In addition to stakeholders the requirements can also be discovered from system documents that describe current or competing products or systems and also from problem reports and enhancement requests for a current, domain knowledge and market studies. Also external sources: other companies, vendors, publications, seminars, workshops and on-line data services should be used.

After elicitation of the requirements they have to be analysed in order to discover problems, incompleteness and inconsistencies in the elicited requirements. As the problems, inconsistencies and conflicts within requirements arise, they have to be solved by negotiating with the different stakeholders. The listing or rating in order of priority - prioritization of the requirements - is also necessary but also challenging phase. If the prioritization is done thoroughly the project will avoid a lot of problems later on especially is there is going to be shortage of resources: time, money or effort during the project [Wiegers, 1999]. There are a many different techniques to do prioritization: prioritization scales, prioritization models, Kano method, Quality function deployment (QFD) and Total quality management (TQM) [Wiegers, 2003].

Requirements documentation is the result of previous phases and when it is finalized it provides a representation of the system for the customer's review and approval. During the requirements validation the requirements document is checked for consistency and completeness. Validation assures that the right problem is being solved in the process the requirements specification is internally consistent and consistent with the stakeholders intentions.

Typical outputs of the requirements engineering process are system specification, associated analysis models and agreed requirements. Once reviewed and approved, these outputs define the requirements baseline for the development effort, an agreement between the development group and its customers.

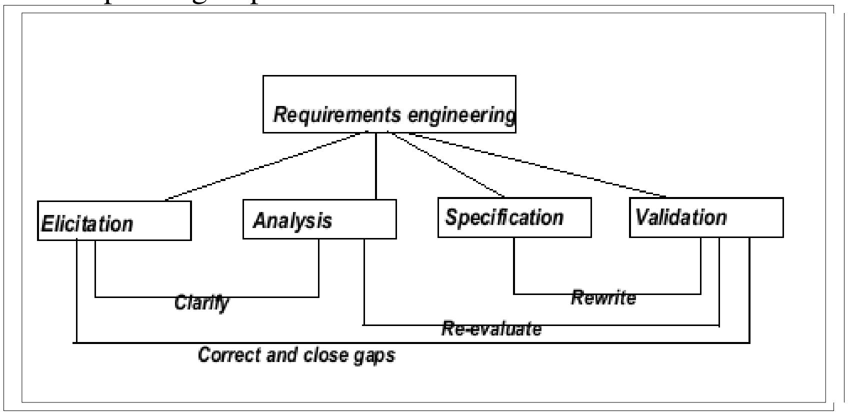

Figure 8. Requirements engineering.

Requirements management process follows the requirements engineering process. It involves processes, tools and practice to *maintain* the integrity and accuracy of the requirements agreement. According to Wiegers [2003] the four main processes of requirements management are:

- Change control
- Version control
- Requirements tracing: managing relationships between requirements and dependencies among requirement document

- Requirements status tracking: defining and tracking statuses.

Change control is a method to manage changes to agreed requirements. The proposed changes are recorded and handled and their impact is analysed. The decision if the changes should be implemented also has to be done. The requirement specification and other documents have to be updated. It is also useful to measure the requirement's stability.

The most important activity of requirement management version control is to define version identification schema. The versions of requirement specification also have to be identified. Each circulated version of the requirements documents should include a revision history that identifies the changes made, the date of each change, the individual who made the change, and the reason for each change. It's sometimes also useful to versions also the individual requirements.

Tracking the status of each functional requirement throughout development provides a more accurate gauge of project progress. First of all the statuses of requirements which are interesting to track have to be defined. Typical requirement statuses are: proposed, approved, implemented, verified, deleted and rejected.

Tracing of the requirements is managing the logical links between individual requirements and other project work products. By doing and updating this information it is a lot easier to figure the impact of a change to a requirement.

Requirements management is an essential part of controlling complexity, risk and project scope.

2.3. Project management process and small scale project

Most projects have three constraints or limitations: time limit, cost limit and scope [Luckey and Phillips, 2006]. The time limit – schedule - defines timing and sequence of tasks within a project. A schedule consists mainly of tasks, task dependencies and durations. Estimated cost of the project is the budget of the project and it defines not only the maximum amount of money which can be spent on the project but also all the resources required to carry out the project. Costs include the people and equipment that do the work, the materials they use, and all the other events and issues that require money or someone's attention in a project. The scope of the project defines what should be the results of the project. It can be a tangible item or a service.

The combination of these three elements can be referred as the project triangle [Chatfield and Johnson, 2004]. The project triangle illustrates the process of balancing constraints because the three sides of the triangle are connected, and changing one side of a triangle affects at least one other side.

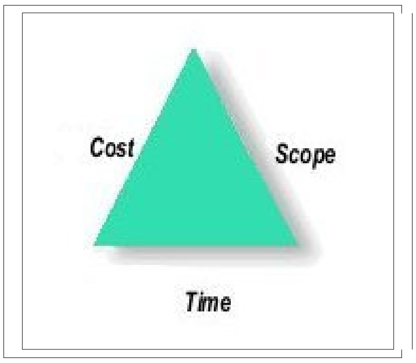

Figure 9. Project triangle.

Understanding the project triangle will allow one to make better choices when tradeoffs are necessary to make. If any one side of the triangle is adjusted, the other two sides are affected. For example, if the project has to be finished earlier than planned, either the costs usually increase because more work has to be done in shorter time which requires more resources or scope has to de decreased. Increasing the scope might take more time and cost more money in the form of increase of resources. If the cost is decreased result might be a longer schedule and a decreased scope. Yet changes to your plan can affect the triangle in various ways, depending on your specific circumstances and the nature of your project. For example, in some instances, shortening your schedule might increase costs. In other instances, it might actually decrease costs.

Project management can be defined as the work that is done within the project triangle. It is the application of knowledge, skills, tools, and techniques to project activities to meet project requirements [Project Management Institute, 2004]. Project management is accomplished through processes, using project management knowledge, skills, tools, and techniques that receive inputs and generate outputs. Effective project management concentrates on getting work done on time and within budget while meeting customer expectations.

Wysocki [2006] has defined software development project management as a discipline of assessing the characteristics of the software to be developed, choosing the best fit software development life cycle, and then choosing the appropriate project management approach to ensure meeting the customer needs for delivering business value as effectively and efficiently as possible.

In a successful project the project team has to:

- Select appropriate processes within the project management process that are required to meet the project objectives
- Use a defined approach to adapt the product specifications and plans to meet project and product requirements
- Comply with requirements to meet stakeholder needs, wants and expectations
- Balance the competing demands of scope, time, cost, quality, resources, and risk to produce a quality product.

Software project management, much like other project management, has four major phases, planning, organizing, monitoring and adjusting, POMA [Tsui, 2004]:

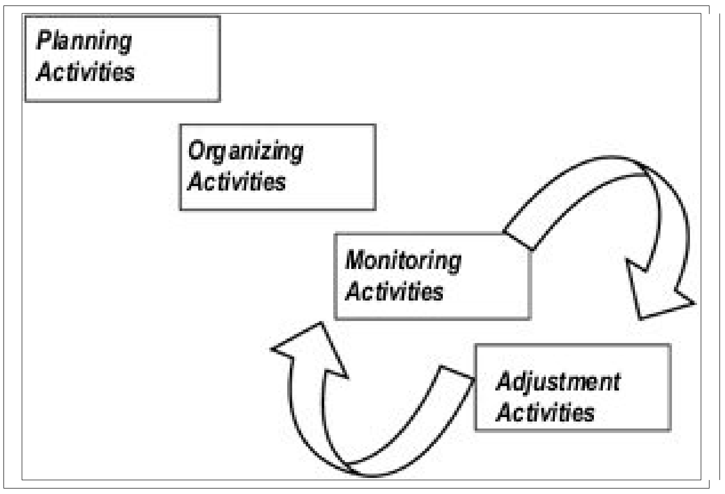

Figure 10. The POMA model [Tsui, 2004].

The POMA management process starts with the planning of tasks and moves through the remaining categories of project management activities. All the phases of POMA model are needed to some extent in all software projects. Even small scale projects should contain all of them.

Luckey and Phillips [2006] list nine project management knowledge areas:

- scope management: to control the planning, execution, and content of the project
- time management: managing everything that affects the project's schedule
- cost management: cost estimating, budgeting, and control.
- quality management: to ensure that the produced product meets customer expectations
- human resources management: to hire or assign and manage people who are competent
- communications management: focuses on who needs what information - and when.
- risk management: focus is on how to anticipate risks, handle them when they arise, and take advantage of the opportunities that can help a project
- procurement management: it may be required to work with or as a vendor to purchase goods and/or services. This knowledge area is concerned with the processes to create vendor contracts and to purchase goods and services.
- integration management: purpose is to ensure the coordination of all the other knowledge areas.

All the activities in all projects – also small projects - should be managed by POMA phase model. Yet lighter methods and less effort is usually needed in small projects. The knowledge, skills and processes should not always be applied uniformly on all projects. The project manager, in collaboration with the project team, is always responsible for determining what processes are appropriate, and the appropriate degree of rigour for each process, for any given project. The question what project management approach is appropriate for managing the software development process is not an easy one to answer. The most important factors that influence the choosing management tools are business environment, technology, industrial standards, quality program, vision, budget, size, structure and culture of organization.

Generally the bigger a project is the more difficult it is to manage. Each management area requires more work in a big project: especially time, risk and communication management becomes more challenging in a big project. It is usually recommended to be divided a big project into smaller more manageable projects which are conducted either parallel or successively. The ideal duration of a project is not longer than 6 months. But even then there are more integration and time management demanded.

One of the arguments against using project management methodologies is that they are very process-centric and produce vast quantities of project documentation which are simply not practical or desirable on small projects [Buehring, 2006]. Yet any method which focuses on producing documentation at the expense of delivering the real business benefits of the project will be a hindrance rather than a benefit. The focus in project management is delivering business objectives, not producing unnecessary documents so it is critical to find the right level and intensity of management without burden of piles of documents to be created and especially updated.

Also in a small project scope management is essential. It is important to define the objectives and scope and find an agreement of them with the stakeholders. It is also necessary to document them as one would document any kind of agreement. Without having a written scope and a project plan it is difficult to define what is included in the project and what has to be considered as a change.

Time management is also necessary in a small project. Even small projects include different tasks which have to be done and without planning, documenting and tracking them it is difficult to track the progress of the whole project. The tools and methods can be as simple as a bar chart or table.

Cost management estimating, budgeting, and control is required regardless of the size of the project. Yet usually the workload is heavier the bigger the project is.

In small project the quality management also has to be done some way. The methods can be light but it is necessary to ensure that the produced product meets customer expectations.

In small projects there is usually less than 5 people working so the human resources management is easier. On the other hand the skill of the team members usually have to be more multifaceted therefore more difficult to assign. Also communication management is easier since usually there is fewer stakeholders involved in small projects.

Risk management is also a important part of smaller projects. What makes it easier in a small project is that the amount of risks is usually less than in a big project. The method to manage risks can be as simple as listing and categorizing the risks and reviewing and updating them during the project. Procurement management can also be needed in a small project if services or goods have to be purchased for the project or if the project is working as a supplier. Integration management can be a challenging task also in a small project if the result of the project is connected to many other systems.

Table 1 contains a summary of different project management knowledge areas and some features of how the size of the project affects them.

Knowledge Area	**Small project**
Project Scope Management	Usually less complex and there is less dependencies
Project Time Management	Usually shorter which results into less changes. There is also less people involved. Methods can be simple: table or bar chart.
Project Cost Management	Cost estimation is easier because there is less tasks and the duration of project is shorter. Controlling is also less demanding because of the nuber of tasks. Methods can be simplified.
Project Quality Management	Can be time consuming also in small project if there is a lot of customers involved. Yet methods can be lighter since there is less tasks and features in small projects
Project Human Resources Management	Less demanding since usually there is less people involved in the work.
Project Communications Management	Less demanding since usually there is less stakeholders involved.
Project Risk Management	Since there is less tasks and the duration is shorter the total amount of risks is usually also less. Yet some other factors f. ex. Risks connected development environment are the same as in bigger projects. The consequences are usually lighter in small projects. Simplified methods can be used in risk management.
Project Procurement Management	Depends more on the type of project than the size. Also small projects can include many vendors contracts and a lot of purchasing of services.
Project Integration Management	Depends more on the type of product to be produced. If the product is supposed to have interfaces to many other systems integration management can be demanding.

Table 1. Project management areas in small project.

2.4. What makes a project successful?

The definition of successful project can mean different things. Successful project can mean that the external customer's expectations are fulfilled, the objectives of a development project were realized and the results of the project are part of a processes of the organization [Rajala, 2004]. There can also be different levels of success depending on which critical success factors and accomplishment rates are defined to the project. Important aspect is also the internal point of view and the vendor's point of view, internal payback versus performance. Did the project process work well? Were the methods used correctly, were the resources really reserved for the project and were they used efficiently?

White and Fortune [2002] made a survey in a form of questionnaire which was sent to 995 project managers. The response rate was 23,7 %. The respondents were asked to indicate and rank the criteria they used for judging success of the project. As a result the five most important criteria used for judging project success were:

- Meets clients requirements
- Completed within schedule
- Completed within budget
- Meets organizational objectives
- Yields business and other benefits.

The first three criteria are actually working within project triangle the other two emphasize more the organizational view to the project.

There are various lists in literature describing the factors which make a project successful. According to Snedaker [2005] the success factors are:

- Executive Support
- User Involvement
- Experienced project manager
- Clearly defined project objectives
- Clearly defined (and smaller) scope
- Shorter schedules, multiple milestones
- Clearly defined project management process
- Standard infrastructure.

Executive support is the number one factor impacting project success. If executives are not supportive, they're unlikely to assign needed resources: labour hours, money, equipment etc. to the project. Additionally, they are less likely to defend the project if it runs into trouble. Executive support can also help ensure that cross-functional teams have resources and responsibility to work on their assignments, which is especially challenging when projects cross departmental and divisional corporate lines. Finally, executive support can give a project company-wide visibility.

The user involvement is number two on the list. The user involvement is especially important during the earlier stages of a project. Typically the lack of user involvement results into products which are not useful to end users.

In recent years, it’s become clear that an experienced project manager has a significant impact on the success rates of IT projects. Experience becomes a more important factor as the project size, cost, and timeline increase.

There are a number of reasons why projects lack clearly defined objectives. In most projects the objectives are defined but the problem is that they are understood in various ways or they are changing continually.

Studies have shown that the longer a project runs, the less likely it is to be successful therefore there is an inverse relationship between project success and time. Scope*,* defined as the total amount of work to be accomplished, is inextricably linked to time—the more work that needs to be done the longer it usually will take. The scope of the project is reflected in the project’s objectives.

Scope defines the total amount of work to be accomplished and the schedule defines the shortest possible path to accomplishing that work. If the schedule starts getting longer, it’s possible that the scope is creeping which will directly reduce the chances for success.

Studies have also shown that projects with more milestones placed closer together are more successful. They can be project phase transition points; points at which external data, resources, or decisions must be gathered, or simple checkpoints. The more often there is a checkpoint, the more successful the project is likely to be. The sooner it’s noticed the project is heading off-course, the easier it is to make small adjustments.

Clearly defined project management process is the seventh success factor for several reasons. First, when processes are clearly defined, re-use of processes is possible. Over time, this leads to improved processes that can easily be implemented by members of the project team. Also, clearly defined processes reduce some of the project re-work. Having well-defined project management processes also helps to do the right things in the right order without having to give too much thought to the process. There’s also an efficiency that comes with having well-defined, easy-to-use processes.

The last major success factor is using standard software infrastructure. Clearly, this applies directly to software development initiatives, but it can also be applied to other IT projects. According to the Standish Group research, 70% of all application code is considered infrastructure, which means that the other 30% is the custom code. Standardizing the infrastructure of the code leads to both efficiency and stability in the code.

Bechthold [1999] lists the success factors of a project as:

- Appropriate processes to conduct the project
- Capitalization is covering all the required expenses
- Customer focus: the better the project members are familiar with customer and his needs the better the project succeeds
- Historically based, relevant, predictive data can be required for calculating productivity rates, estimating product size and complexity, estimating total hours

to build the various components of the system, and planning a schedule of activities with estimated durations

- Modern but approved support environment and tools
- Project brevity, the shorter the project is the easier it is to conduct
- Requirements stability
- Strategic teaming is possible meaning that it is possible to find partners to add one or more capabilities that are not available from within your company
- Team cohesiveness is usually better if the team members have been working together long and they work well under pressure, yet maintain momentum in the absence of pressure and the team have a track record of welcoming and rapidly acclimating new project personnel
- Team expertise in both the problem domain and the solution domain
- Training support is available if needed
- The project manager is an expert as a manager.

According to White and Fortune [2002] the success factors of a project were very similar to earlier found. In their research the three critical success factors turned out to be:

- clear goals and objectives
- support from senior management
- adequate funds and resources.

As a summary the most important factors to produce a successful project is that the project scope and objectives are clearly defined and documented. User involvement in the project is the second most important factor since in the end it is the customer who decides if the project was successful. The third most important success factor is that the project has an experienced project manager.

3. Quality

3.1. Quality as a concept

Quality is a concept that lacks a clear and concise definition and is thus difficult to accurately measure, improve or even compare across different industries, products and services. The quality of a service refers to the extent to which the service fulfils the requirements and expectations of the customer.

An understanding of the basic concepts on quality and its management is essential for the professional management of Quality of Service (QoS). According to ISO 8402 standard the concept of quality can be defined as "totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs". An entity is an item that can be individually described and considered. An entity may be an activity or a process, a physical product or an organisation, actually anything that has a quality attribute.

Quality is a phenomenon; it is an emergent property of people's different attitudes and beliefs, which often change over the development life-cycle of a project.

The concept of quality can be observed different standpoints. Lillrank [1990] names six different viewpoints. A customer oriented viewpoint emphasises customer satisfaction, which arises from fitness for use. A manufacturing oriented viewpoint means similarity and consistency in the manufacturing process. A product oriented viewpoint emphasises product performance and value oriented viewpoint concentrates on getting best possible quality for the lowest possible price. Additionally an environment oriented viewpoint emphasises the environmental impact of the product.

A basic element of quality is that it is not free: it always requires efforts typically reviews, testing, inspections etc. which cost but on the other hand it always adds some value to the customer. Experiences in manufacturing relating to the cost and return of quality improvements suggest that there are diminishing returns to quality expenditures. Therefore the key management problem is how to make profitable decisions on quality expenditures. Typically quality characteristics may be required or not, or may be required to a greater or lesser degree, and trade-offs may be made among them.

3.2. Software quality

Software of good quality can be defined as software without any errors and deficiencies. Yet it is very difficult to prove that software doesn't contain any errors. Thus software of good quality is software without any known errors and deficiencies.

The business value of a software product results from its quality. Quality is increasingly seen as a critical attribute of software, since its absence results in financial loss as well as dissatisfied users, and may even endanger lives.

Since the importance of setting software quality requirements and assessing quality is better recognised a shift from creating technology-centred solutions is made to satisfying stakeholders. Software acquisition, development, maintenance and operations organizations confronted with such a shift are, in general, not adequately equipped to deal with it. Until recently, they did not have the quality models or measurement instruments to allow or facilitate the engineering of quality throughout the entire software product life cycle. The objective of software product quality engineering is to achieve the required quality of the product through the whole production process: definition of quality requirements and their implementation, measurement of appropriate quality attributes, and evaluation of the resulting quality. The objective is, in fact, software product quality.

Software quality can be divided into three main categories or topics: software quality fundamentals, software quality management processes and practical considerations as in Figure 10 below [Abran, Moore (eds) et al., 2004].

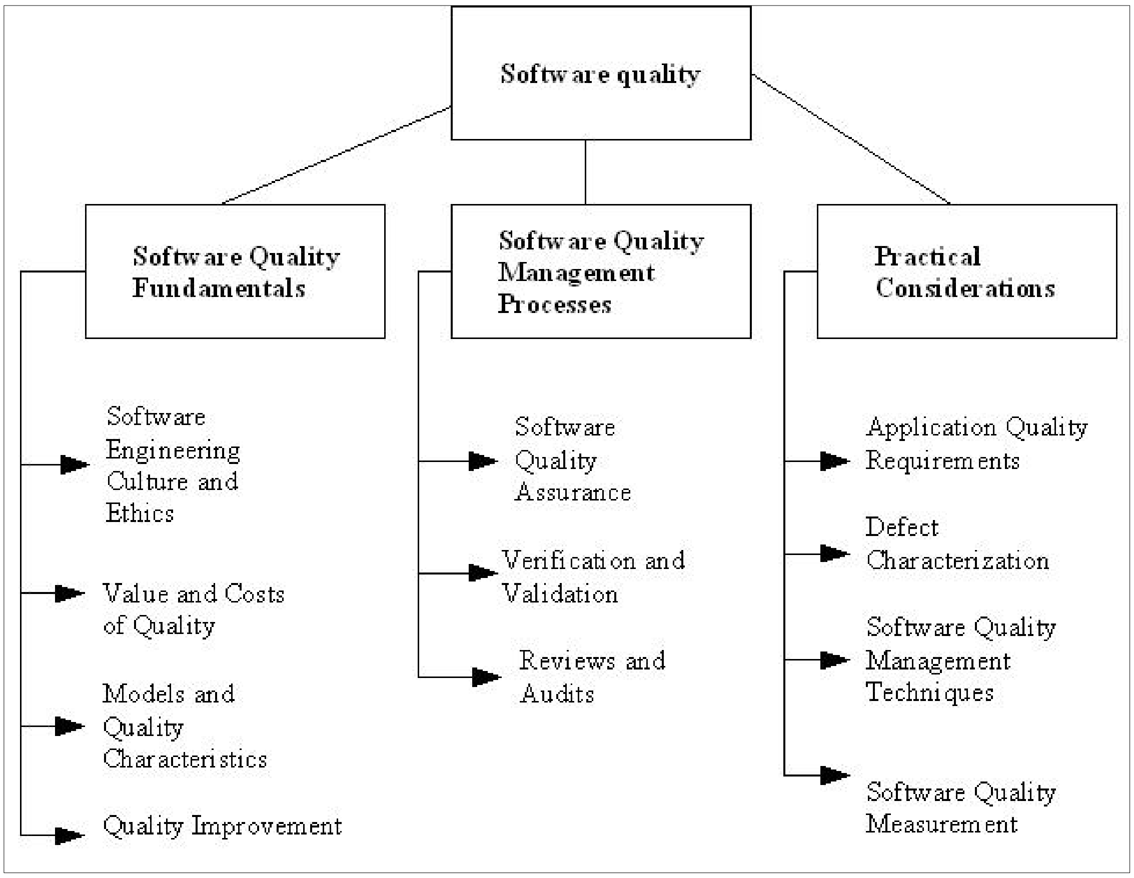
Software quality
Software Quality Fundamentals
Software Quality Management Processes
Practical Considerations
Software Engineering Culture and Ethics
Value and Costs of Quality
Models and Quality Characteristics
Quality Improvement
Software Quality Assurance
Verification and Validation
Reviews and Audits
Application Quality Requirements
Defect Characterization
Software Quality Management Techniques
Software Quality Measurement

Figure 11. Categories of software quality.

Software quality is based on software engineers who share a commitment to software quality as part of their culture. Ethics can play a significant role in software quality, the culture, and the attitudes of software engineers. Quality culture cannot be "bolted on" to an organization; it has to be designed-in and nurtured. The ultimate aim of upper management is to instil a culture that will allow the development of high-quality products and offer them at competitive prices, in order to generate revenues and dividends in an organization where employees are committed and satisfied [Duggan and Reichgelt, 2006].

Software process engineering has made many advances in the last decade and has introduced numerous quality models and standards concerned with the definition, implementation, measurement, management, change, and improvement of the software engineering process itself. Process quality is part of the technical and managerial activities within the software engineering process that are performed during software acquisition, development, maintenance, and operation. The most common quality standards are the ISO9001-2000 standard, CMMI (Capability Maturity Model Integration) TQM (Total quality management) and ITIL (the IT Infrastructure Libraby), which is actually a collection of best practices.

The quality of software products can be improved through an iterative process of continuous improvement which requires management control, coordination, and feedback from many concurrent processes: the software life cycle processes, the process of error/defect detection, removal, and prevention, and the quality improvement process. Software quality management (SQM) applies to all perspectives of software processes, products, and resources. It contains three basic processes: quality planning, quality control, and quality improvement. It defines processes, process owners, and requirements for those processes, measurements of the process and its outputs, and feedback channels. The software quality management processes addresses how well software product will satisfy customer and stakeholder requirements, provide value to the customers and other stakeholders, and provide the software quality needed to meet software requirements. Capability maturity model (CMM) is method to estimate and improve the processes of organization [Capability Maturity Model Integration, 2002]. It scales the maturity of organization into five levels which describe how the processes in organization function. The maturity level of an organization provides a way to predict the future performance of an organization within a given discipline or set of disciplines. At the first maturity level called Initial the processes are usually ad hoc and chaotic. Success in these organizations depends on the competence and heroics of the people in the organization and not on the use of proven processes. At the fifth level called Optimizing the processes are continually improved based on a quantitative understanding of the common causes of variation inherent in processes. Maturity level 5 focuses on continually improving process performance through both incremental and innovative technological improvements. At this level the quality of production is easy to maintain because of the support of processes.

Software quality assurance (SQA) processes provide assurance that the software products and processes in the project life cycle conform to their specified requirements by planning, enacting, and performing a set of activities to provide adequate confidence that quality is being built into the software. This means ensuring that the problem is clearly and adequately stated and that the solution’s requirements are properly defined and expressed.

Software verification and validation is a disciplined approach to assess software products throughout the product life cycle. It uses testing techniques which can locate defects so that they can be addressed. The verification and validation process determines whether products of a given development or maintenance activity conform to the requirement of that activity, and whether the final software product fulfils its intended purpose and meets user requirements. The quality of software is not only influenced by the quality of the process used to develop it, but by the quality of the product as well. The software product quality model provided in ISO/IEC 9126-1 defines six quality characteristics:

• Functionality

– suitability, accuracy, interoperability, security, compliance

• Reliability

– maturity, fault tolerance, recoverability, compliance

• Usability

– understandability, learnability, operability, attractiveness, compliance

• Efficiency

– time behaviour, resource utilisation, compliance

• Maintainability

– analysability, changeability, stability, testability, compliance

• Portability

– adaptability, installability, co-existence, replacability, compliance

Software testing should cover all these characteristics.

There are five different types of reviews and audits: management reviews, technical reviews, inspections, walk-throughs and audits. The purpose of a management review is to help in decision making. They monitor progress, determine the status of plans and schedules, confirm requirements and their system allocation, or evaluate the effectiveness of management approaches used to achieve fitness for purpose. The purpose of a technical review is to evaluate a software product to determine its suitability for its intended use. The objective is to identify discrepancies from approved specifications and standards. The purpose of an inspection is to detect and identify software product anomalies. Inspection differs from review in to aspects: an individual holding a management position over any member of the inspection team shall not participate in the inspection and an inspection is to be led by an impartial facilitator who is trained in inspection techniques. Walk-through is meant to evaluate a software product. The walk-through is similar to an inspection but is typically conducted less formally. The purpose of a software audit is to provide an independent evaluation of the conformance of software products and processes to applicable regulations, standards, guidelines, plans, and procedures.

To gain good quality software there has to be also processes in SQM to find and manage defects. Characterizing and classifying those defects into different categories according to their type and criticality is important to ease communication between the stakeholders. It also leads to an understanding of the product, facilitates corrections to the process or the product, and informs project management or the customer of the status of the process or product. Many defect taxonomies exist; the point is to establish a defect taxonomy that is meaningful to the organization and to the software engineers.

SQM techniques can be categorized to static, people-intensive, analytical, and dynamic. Static techniques involve examination of the project documentation and software, and other information about the software products, without executing them. People-intensive techniques include reviews and audits and they can be formal meetings, informal gatherings or desk-check situations. A software engineer usually uses analytical techniques. These techniques are either tool-driven or manual. Examples of such techniques include complexity analysis, control flow analysis, and algorithmic analysis. Various dynamic techniques are performed throughout the development and maintenance of software. Generally, these are testing techniques, but techniques such as simulation, model checking, and symbolic execution may be considered dynamic. Testing is also important part of quality assurance. SQA ensures that appropriate types of tests are planned, developed, and implemented, and test plans, strategies, cases, and procedures are being developed.

The models of software product quality often include measures to determine the degree of each quality characteristic attained by the product. If they are selected well measures can help the management in decision-making. They can also find problematic areas and bottlenecks in the software process and give assistance in deciding when to stop testing. The cost of SQM processes is an important issue but it is difficult to gain exact answers. Often, generic models of cost are used, which are based on when a defect is found and how much effort it takes to fix the defect relative to finding the defect earlier in the development process.

What does quality have to do with the project triangle? Quality is at the centre of the project triangle. Quality affects every side of the triangle, and any changes you make to any side of the triangle are likely to affect quality. Quality is not a factor of the triangle; it is a result of what is done during the project with time, money, and scope.

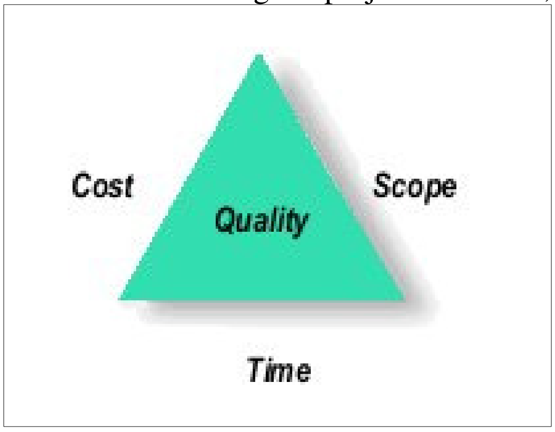

Figure 12. Quality in project triangle.

If there is additional time in project schedule, one might be able to increase scope by adding tasks and duration. With this extra time and scope, one can build a higher level of quality into the project and its deliverables.

Or if there is need to cut costs to meet the budget, the scope has to be decreased by cutting tasks or reducing task durations. With decreased scope, there might be fewer opportunities to achieve a certain level of quality, so a lower quality results from the need to cut costs.

This study concentrates on the requirements of software. Yet all the categories of software quality by Abran, Moore (eds.) et al. [2004] described previously affect also the quality of software requirements looked from a wide point of view. The elements affect the quality of requirements differently: some elements like software quality assurance, verification and validation and reviews and audits, have more effect, some influence more covertly like software engineering culture ethics. Some elements of quality are slower to change f. ex. software engineering culture ethics, some are easier to bring into use. In next chapter the most typical quality factors of requirement analysis are discussed.

3.3. Quality in requirement analysis

Requirements are the foundation of the software development process. They provide the basis for estimating costs and schedules as well as developing design and testing specifications. So the success of a software project, both functional and financial, is directly related to the quality of its requirements [Javed, Maqsood and Durrani, 2004]. Additionally defects in requirements are more difficult and expensive to fix the later they are found. Thus the quality of requirements has great importance considering whole quality and success of the project.

The primary product of the requirements engineering process is the requirements specification which should state what a system should do - not how it should do it [Kotonoya and Sommerville, 1998]. The requirements specification should state both the functional and non-functional such as performance, reliability, safety, security and usability requirements. The quality of requirements specifications lays the basis to the quality of the whole project - however it is difficult to evaluate the quality of requirements specification in time. The requirements specifications vary in structure, content, accuracy and representation formats. One reason for this is that there of many standards and guidelines which define the content and the structure for a good requirement specification document in different ways. Another reason is the influence of the requirements engineering process methods: different methods can be used or they can be emphasised differently.

Validation and verification are methods to assure the quality of requirements [Wiegers, 2003]. The activity of assuring that the requirements capture the customer intention is referred to as validation. Validation can be defined as the process of ensuring that the engineer has understood the requirements correctly, in other words, "Have we got the right requirements?" The result of requirements validation is requirements document which is an agreed, complete set of requirements. The way to add quality to requirements is reviewing, inspecting, walkthroughs and testing f. ex. by prototyping. Prototypes for requirements validation demonstrate the requirements and help stakeholders to discover problems. Checklists of what to look for may be used to drive a requirements review process [Kotonoya and Sommerville, 1998]. Designing tests for requirements can reveal problems with the requirements. If the requirement is unclear, it may be impossible to define a test for it.

The activity of assuring that the software meets its requirements and ensuring that the requirements documents conform to specified standards is referred to as verification.

Verification addresses the question of "Have we got the requirements right?" [Abran, Moore (eds) et al., 2004]. In other words, the validation is the activity linking the requirements to the intention, while verification links the requirements to the implementation.

It is not simple to define what is meant by good requirement. The easiest way to approach the problem is to define features or characteristics of good requirement. According to Kotonoya and Sommerville [1988] the characteristics of good requirements are:

- valid in a way that they express only the real needs of the stakeholders
- complete conceptually and structurally and they specify all the features of the system and on the other hand the features that are not included
- consistent so that they don't contradict themselves
- necessary
- unambiguous
- verifiable
- understandable
- prioritized
- traceable.

Firesmith [2003] adds to features metadata so that each requirement should contain attributes or annotations that characterizes them. This metadata can include acceptance criteria, allocation, assumptions, identification, prioritization, rationale, schedule, status, and tracing information.

Although an initial set of requirements may be well documented, requirements will change throughout the software development lifecycle. Constant change addition, deletion and modification in requirements during the development life cycle impacts the cost, schedule, and quality of the resulting product. Thus the quality of requirements management is as important as requirements engineering. The typical reasons for change of requirements during the project are errors, conflicts and inconsistencies in requirements which must be corrected during analysis, validation or later development. Especially if the project is long there will be the project staff, customers and end-user gain more knowledge of the system which may cause changes in requirements. Technical, schedule or cost problems may cause need for change as well as different external environmental changes like changing customer priorities, changing business environment, emergence of new competitors, organizational changes and new legislation.

3.3.1. Prioritization of requirements

Prioritization of requirements is a way to add quality to project work. As noted before projects are always working under limited resources trying to fulfil customer’s expectations which very often are too high. Requirements prioritization means balancing the business benefit of each requirement against its cost and any implications it has for the architectural foundation and future evolution of the product. By priorization of the requirement it is easier to create the work plan, work breakdown structure (WBS), schedule and risk management for the project and the project management gets more flexibility to decision making [Sommerville and Sawyer, 1997]. If requirements are prioritized during the analysing stage time consuming discussions with the customer are not needed during the hectic implementation phase if shortage of any resources appear.

Prioritization scales is a common approach to prioritization [Wiegers, 1999]. Usually requirements are divided into three priority categories: high, medium and low. Another scale could be essential, conditional and optional. All scales are subjective and imprecise, so everyone involved must agree on the meaning of each level in the scale they use. The granularity at which to prioritize requirements is another issue. Even a medium-sized project can have hundreds or thousands of detailed functional requirements, too many to classify analytically and consistently. An appropriate level of abstraction has to be chosen for the prioritization. It could be at the use case level, the feature level, or the requirement list level.

Another prioritization method is prioritization models with cost-value approach like analytic hierarchy process (AHP). It is used in general as a problem solving process in which different solutions are evaluated. A decision is structured into a hierarchy, determining relative priorities for the elements in the hierarchy, and combining the numbers into overall weights estimating each decision outcome.

Total quality management (TQM) actually is quality method for the whole business. Total Quality is a description of the culture, attitude and organization of a company that aims to provide, and continue to provide, its customers with products and services that satisfy their needs. The culture requires quality in all aspects of the company's operations, with things being done right first time, and defects and waste eradicated from operations. With requirements prioritization it rates each requirement against several weighted project success criteria and computes a score to rank the priority of the requirements.

Kano method classifies customer preferences into five categories: attractive, one-dimensional, must-be, indifferent and undesired.

Quality function deployment model (QFD) uses the Kano model by structuring of the comprehensive QFD matrices [Zultner, 1992]. OFD is a comprehensive method for relating customer value to the features for a proposed product.

4. QFD

4.1. General

QFD is a methodology which concentrates on taking account of quality and its different dimensions during the product design process and integrate quality to a product from the beginning [Lillrank, 1990]. Unlike traditional quality systems which aim at minimizing negative quality in a product, QFD adds values to the product by maximizing the positive quality. Emphasis is on customers needs. It can be defined as a structured planning and decision making methodology for capturing customer needs and translating those requirements into product requirements, part characteristics, process plans and quality/production plans through a series of matrices. It is not originally a requirement engineering technique, but rather a systematic method for translating customer requirements into specific product design [Geras *et al.*].

The basics of QFD were developed in Japan. In the late 1960s by Shigeru Mizuno and Yoji Akao to design customer satisfaction into a product before it was manufactured [Shahin, 2005]. Kiyotaka Oshiumi who was working with Bridgestone Tire in Japan used a process assurance items fishbone diagram in 1966 to identify each customer requirement and to identify the design substitute quality characteristics and process factors needed to control and measure it. In 1972, with the application of QFD to the design of an oil tanker at the Mitshubishi Heavy Industries Ltd's Kobe shipyard, the fishbone diagrams turned out to be unwieldy. 1978 Toyota Autobody used QFD and 1983 the first QFD seminar was held in Japan. In 1990's the American car industry adopted QFD. Nowadays QFD has been widely applied in the products and manufacturing industry: machine building, consumer products and automobile body parts.

Since customer's role is important in software engineering there was a lot of interest to contrive QFD into that field. At the same time TQM was emerged as an important aspect of overall quality improvement programs in many organizations. Specifically, QFD seemed to be interesting method which could be elaborated to a implementation vehicle of TQM. QFD utilizes cross-functional teams and management to integrate the organization horizontally so that all departments work together to achieve the common goal of satisfying customer demands. QFD is driven by the concept of quality and results in the best possible product to market.

The methodological transfer of QFD technology from manufacturing industry to software engineering first took place in 1984, when the Japanese began to explore its use in the development of embedded software. Four years later, Digital Equipment Corporation (DEC) announced that QFD was being utilized for software development [Haag, Raja and Schkade 1996]. 1987 Richard Zultner first described how QFD could be integrated with software engineering. Betts combined QFD principles across the entire software development life cycle in 1989. While working in Hewlett Packard he applied QFD to a Corporate Quality Information System project, PRIMA. Further development was done in 1996 with requirements engineering with the advent of term Software Quality Function Deployment (SQFD). SQFD represents a transfer of the technology of QFD from its traditional manufacturing environment to the software development environment. Other organizations that have applied SQFD in software projects include AT&T, IBM, Texas Instruments and SAP [Liu et al., 2006].

4.2. The Four-Phased approach

Traditional QFD, which is mostly used in manufacturing industry, contains four phases:

1. House of Quality
2. Part deployment
3. Process planning
4. Production planning

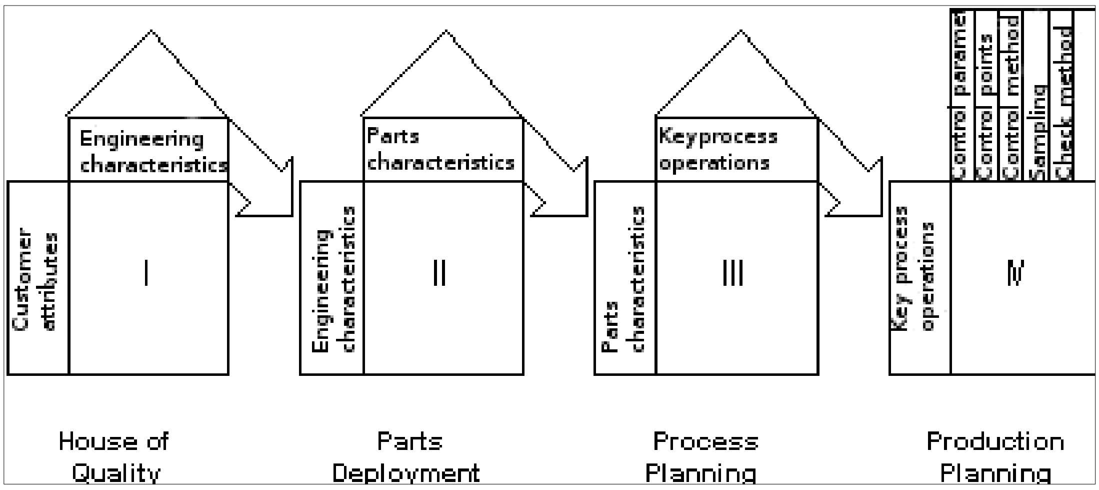

Figure 13. The traditional QFD Four-Phase-Method.

A four phases approach is accomplished by using a series of matrices [Chan and Wu, 2002]. During the product planning phase a matrix called House of Quality (HOQ) is made.

During part deployment phase the prioritized technical measures in the first phase are transformed into part characteristics. The critical parts and assemblies have to be identified as well as critical product characteristics. These have to be translated into critical part characteristics and target values.

During the process planning phase the different manufacturing process approaches are evaluated and the preferred approach. Selected critical processes and process flows are also determined. Also the production equipment requirements are developed and critical process parameters are established.

There has been various ways how traditional QFD has been correlated to software development process. 1989 Betts named the four phases as production planning, design planning, process planning and production planning. L. Cohen in his book "Quality function deployment – How to make QFD work for you" in 1995 named the four phases as product planning, part deployment, process planning and process/quality control.

During the process and quality control phase detailed planning related to process control, quality control, set -up, equipment maintenance and testing are made.

QFD is most known because of this matrix and it is considered a common mistake to assume that QFD means only House of Quality. Yet Haag, Raja and Schkade [1996] limited the focus of SQFD to requirements engineering. They see SQFD as a front-end technique, it is an adaptation of the A-1 matrix, the "House of Quality", which is the most commonly used matrix in the traditional QFD methodology. The SQFD process can illustrated as in Figure 14.

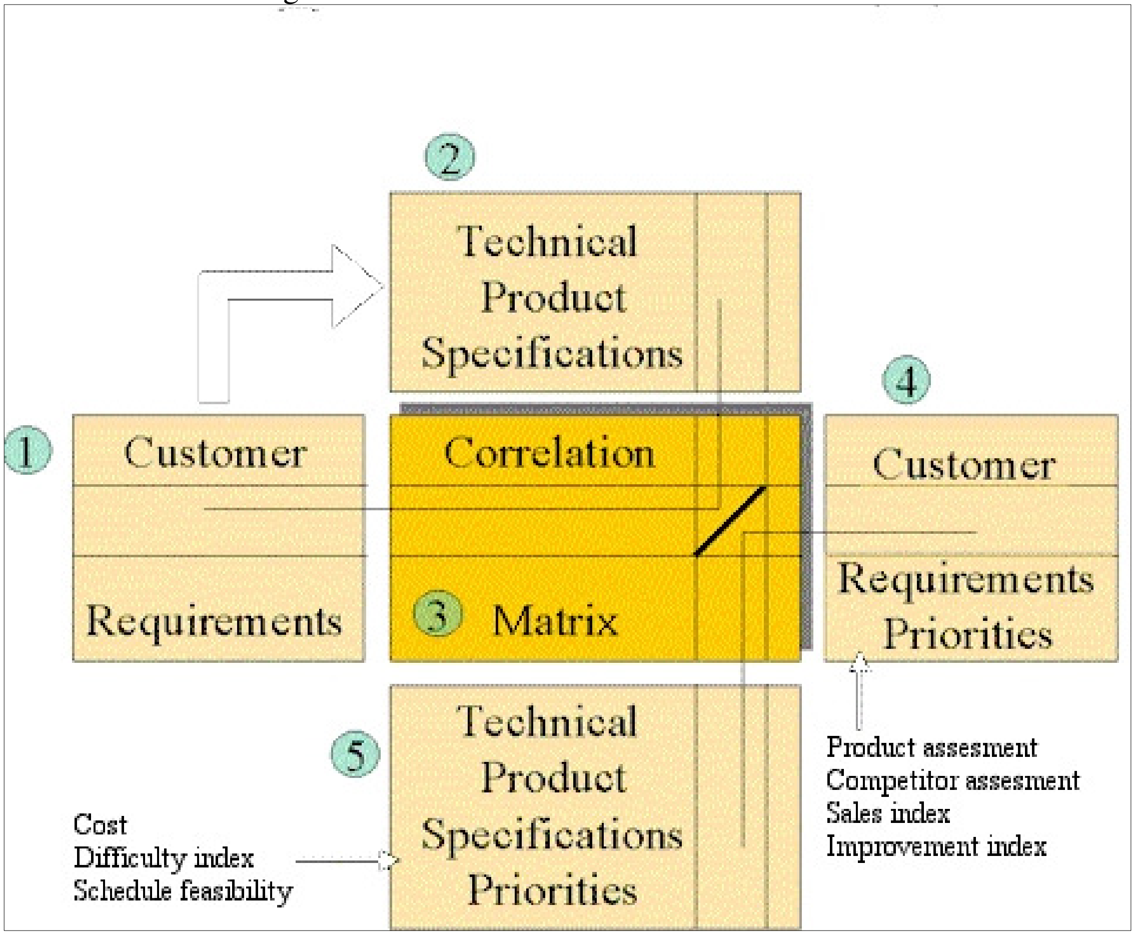

Figure 14. The SQFD process.

4.3. Types of requirements

Before introducing the House of Quality in detail it is important to understand different types of requirements. Traditionally requirements are divided into functional requirements (FR) and non-functional requirements (NFR).

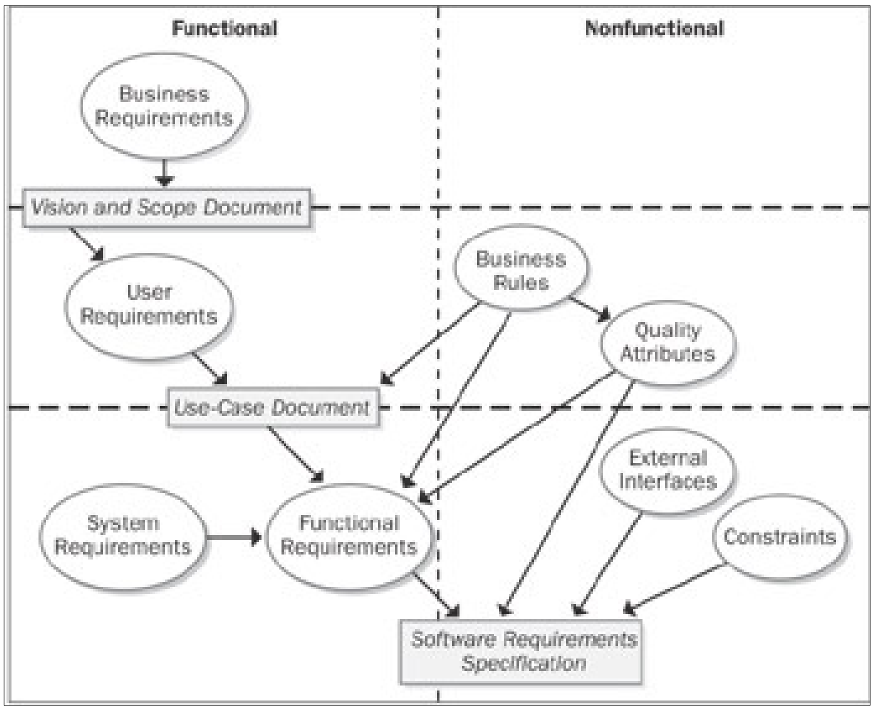

Figure 15. Relationship of several types of requirements information [Wiegers, 2003].

Business requirements represent high-level objectives of the organization or customer who requests the system. Business requirements describe why the organization is implementing the system—the objectives the organization hopes to achieve. Business requirements are usually including in a vision and scope document.

User requirements describe user goals or tasks that the users must be able to perform with the product. Valuable ways to represent user requirements include use cases, scenario descriptions, and event-response tables. User requirements therefore describe what the user will be able to do with the system.

The concept of functional requirements (FRs) means requirements that describe the functionality of the system. They specify the software functionality that the developers must build into the product to enable users to accomplish their tasks, thereby satisfying the business requirements. Sometimes they are called behavioural requirements. Usually the functional requirements are modelled with use cases.

Non-functional requirements (NFRs) describe system properties related to e.g. system performance, usability, security, maintainability etc. Other non-functional requirements describe external interfaces between the system and the outside world, and design and implementation constraints. Constraints impose restrictions on the choices available to the developer for design and construction of the product.

In QFD the user requirements are acquired from the customers. They are typically functional requirements which are grouped into higher level requirements. Technical requirements are regarded as requirements that answer the question how customer's requirements are fulfilled. In a way they are more detailed instructions how the requirements of customers should be implemented. They are gathered form the development team and they have to be measurable.

4.4. House of Quality

The House of Quality which is illustrated in Figure 16 is the most well know matrix of QFD. Its construction process can be divided into eight steps which are explained briefly in the following paragraphs.

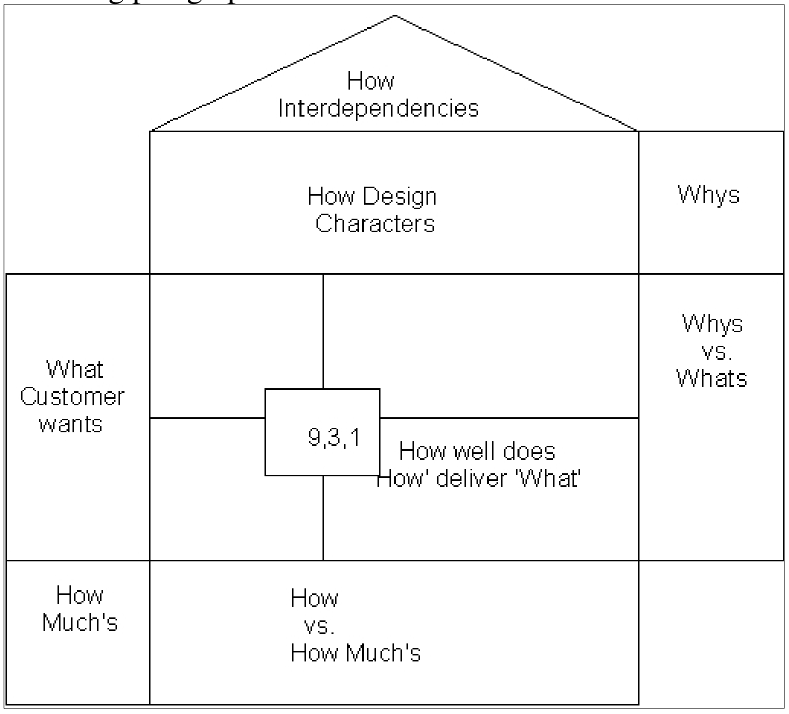

Figure 16. The architecture of the House of Quality.

4.4.1. Customer requirements

During the firs step the customer requirements are solicited through various methods and recorded on the left y-axis. The customers include end users, managers, system development personnel, and any people who would benefit from the use of the proposed software product. The requirements are usually short statements recorded and are accompanied by a detailed definition, the SQFD version of a data dictionary. After all the requirements are gathered, similar requirements are grouped into categories and written into a tree diagram.

4.4.2. Customer prioritization

The second step of the House of Quality is the planning matrix [Chan and Wu, 2002]. The customer requirements are analysed and prioritized. Customers are asked to determine the importance of various requirements. If a current system exists and is in use or there are possible commercial systems on the market the customers may be asked to analyze its functionality compared to requirements. Also market research can be arranged in order to determine the expected performance and functionality. Customer importance rating and market evaluation are the results of this phase.

4.4.3. Technical requirements

At the third step the technical requirements are generated. It must be noted that technical requirements are not understood here in a sense of functional versus non-functional requirements as described in chapter 4.3. In QFD they are technical in a sense that they no longer take on a voice of the customer but instead the voice of the company. These technical requirements should be controllable and measurable characteristics of the product and there can be more than one technical requirement corresponding one customer requirement [Haag, Raja and Schkade 1996]. The generation of technical requirements is a crucial part of the House of Quality. During this step the voice of the customer is translated into the design requirements to be implemented.

4.4.4. Requirements correlation

During the correlation of requirements each combination of customer requirement and a technical requirement, the QFD team must assign a weighting based on the question: "How important is technical requirement A is satisfying customer requirement B?" During this step consensus between customers and development should be reached on the requirements mapping.

4.4.5. Technical feature comparison

The purpose of this step is to consider the impacts that the technical requirements will have on each other. Unlike the previous step, customer requirements are not considered here. Also known as the "roof" of the House of Quality or A3 matrix, this step is critical in identifying engineering trade-offs between technical requirements.

For each pair of technical requirements, the QFD team must answer the following question:" Does improving one requirement cause deterioration or improvement in another requirement? "The "direction of improvement" that was generated in third step is important at this point as one needs to have a measurable quantity with which to determine improvement or deterioration. The impacts are recorded as positive (+), negative (-), or no effect.

4.4.6. Design Targets

The last step is composed of three tasks: technical requirement prioritization, competitive benchmarking, and technical design targets. Each of these relies on the information gathered in the previous steps. The purpose of this step is to integrate the results from all the previous into a set of targets to be used during design and implementation. The crucial difference between QFD and more traditional approaches is that QFD generates targets based on repeatable, statistical analysis. The prioritization of the technical requirements is performed by summing the product of the interrelationship weights with the overall weighting assigned during the planning step.

The House of Quality generates specific technical requirements that can be used during the design phase. The benefit of using the House of Quality is the requirements engineers can trace each requirement back to its source. In addition, the requirements engineer has also recorded the rationale behind each technical requirement. This process ensures that requirements engineers and developers effectively translate the voice of the customer. On the other hand the House of Quality can be an extremely time-consuming process for a large number of requirements. This can result in higher data storage, manipulation, and maintenance costs that require the use of CASE tool support to deal with. As well the entire process is very dependent upon the quality of customer requirements. Finally, without advanced CASE tool support, the process is inflexible to changing requirements.

4.5. QFD in software engineering

Interest in quality issues especially in software engineering started to grow during the late 1980. Several quality standards and methods TQM, ISO 9000, IEEE, ITIL etc. were taken into practice in many organizations. Haag, Raja and Schkade [1996] found out in their research that in organizations the transfer of the quality technology QFD to software development occurred after the implementation of the quality policies in other processes. Yet already in 1992 Zultner claimed that QFD has been applied to virtually every industry and business, including software development. However, one of the important components of QFD, which focuses on improving the process quality by assuring that organizational processes and actions are in compliance with established standards, has been neglected by most QFD followers in the business [Liu, Sun, Cane, 2005].

Several attempts have been made to integrate QFD into Software process improvement (SPI). Liu, Sun and Cane criticized CMM as a method to improve processes in an organization because it only provides practices which are needed to be performed without specifying how. Additionally those practices are not directly related to business goals and other requirements. The project Liu, Sun and Cane managed successfully this issue by using QFD as a tool to connect requirements within an organization to the action plan for its process improvement through KPA goals and KPs in CMM.

Richardson, Murphy and Ryan [2002] proposed a four-stage model for software process improvement in small companies. The measurements in the model are based on self assessment of software process. This model is unsuitable for large companies because self-assessment is difficult across groups.

SAP also uses QFD in SPI [Hierholzer, Herzwurm and Schlang, 1998]. They embedded QFD into Deming's Plan-Do-Check-Act cycle.

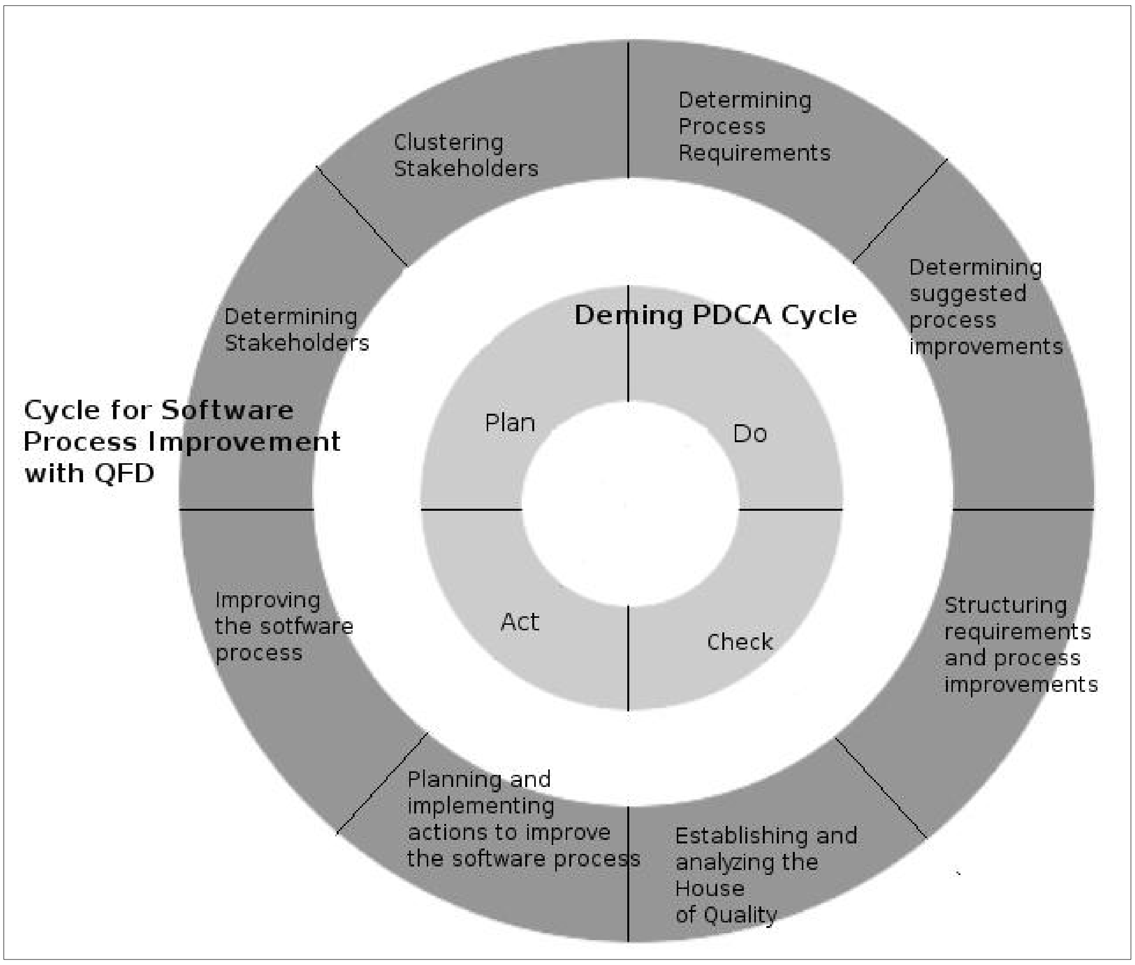
Clustering Stakeholders
Determining Process Requirements
Determining suggested process improvements
Deming PDCA Cycle
Determining Stakeholders
Cycle for Software Process Improvement with QFD
Plan
Do
Act
Check
Improving the sotfware process
Structuring requirements and process improvements
Planning and implementing actions to improve the software process
Establishing and analyzing the House of Quality

Figure 17. QFD in Deming's Plan-Do-Check-Act cycle.

According to their research QFD allowed for concentration of the improvement effort on the most important problems with the current process. This could not have been accomplished as well by using the ISO 9000 or CMM, which gave only general and less concrete suggestions for improvements that were not directly related to the process to be improved. The QFD approach described above allowed analysing not only the problems the stakeholders had with their current process but also the potential for improvement that could be achieved by solving them. The use of QFD efficiently structured the long-standing discussion about possibilities for process improvement at SAP. The structure of QFD allows for very efficient reuse of the information gathered during the project. When the Software Process QFD is repeated, most of the results from the project can be reused and easily merged with the new information gathered, thus greatly reducing the effort required for conducting the new iteration.

4.6. Software tools of QFD

There are a lot of tools which aid the process of using QFD.

PathMaker is a software framework which can be used to define all the steps required to run QFD. It contains management tools that assist in building the content of a QFD matrix. Brainstorm tool is user to collect customer requirements. The affinity diagram tool can be used to sort these requirements and build group of requirements. Rating the importance of the key customer requirements continues the discussion of the previous step. PathMaker contains a tool named Consensus Builder which is used to end up with a suitable analysis of the importance of the customer requirements. When the team needs to analyse the relationship between customer requirements and features/performances there is a Cause and Effects tool that can assist to visualise such relationships. Further the Consensus Builder can be employed to assign a relationship value, i.e. strong, weak or no relationship. As the results need to be recorded somewhere PathMaker provides forms on QFD that enable to create a QFD matrix**.**

QFDcapture Professional Edition is a support tool for any decision making process - from basic to complex. The software has a project focus, rather than a single matrix focus which means that a Roadmap of the lists, matrices, and documents which will be developed for each particular project can be set. The Roadmap indicates links between the matrices. Data which changes in one matrix will cause related changes in the upstream and downstream matrices. QFDcapture Professional Edition contains web page publishing features, a tool for generating customer surveys, a tool to generate market opportunity maps identifying the best opportunities for product improvement. It also contains a tool to generate relationship tree diagrams showing measures for each requirement in a graphical tree and branch format. There are also various templates to print out and work with blank chart templates which are useful as documents-in-progress during team meetings and print out spreadsheets as they appear in QFDcapture.

QFD Designer was the world's first Windows application for QFD. It contains various templates including strategic planning, customer segment analysis, voice of customer tables, House of Quality and failure analysis.

Actually since QFD is a group of matrices all spreadsheet software can be used to provide matrices. It can not be emphasized enough that QFD is a quality methodology and it requires quality thinking. It is very flexible due to its breadth and depth, so that what works for one company may be inappropriate for another. Ideally, QFD should be custom-tailored for each company. Thus it is more question of understanding the meaning and methods of QFD. The software is mostly a tool to gather and record the results of QFD process.

5. QFD in a small software development project – a case study

5.1. Introduction to code register

In large public administration organizations there usually are a lot of different codes which are commonly used in different processes. Typical examples of these codes are currency codes (EUR, USD, GBP etc.), country codes (FI, US, SE etc.), document codes etc. Since these codes are used in many processes it is reasonable to gather all code lists into one centralized database. Thus code lists have to be updated only to one registry and all the systems using codes get always real time information.

The scope of the project was to produce a new code register to the organization. There were a lot of problems with the current version: the framework that was used when building it was not supported any more, the database structure was disintegrated the amount of different database tables was over one hundred. Every time a new code list was created a new table was created, new user interface to manipulate codes of the code list was created and new interface for the other systems to get information of the codes in the code list had to be implemented. All things considered it was very time consuming to maintain the system.

The project to build a new system started as a student project at the University of Tampere in October 2006. The project was estimated to be a small scale project of about 1700 work hours. The scope was to implement a new code register which would be easy to maintain, where different kind of codes with different amount of different kind attributes could be saved. The user interface was also meant to be implemented during the project but the user interface to other systems was defined outside the project.

Originally there were seven members in the project: two project managers and five other members:

Role	**Responsibility**
Project manager	Project management, technical design
Project manager	Project management, technical design
User interface specialist	Design of user interface
Technical expert	Implementation of user interface
Technical expert	Programming, technical design
Technical expert	Database design

Technical expert	Programming, technical design

Table 2. Roles of the project group

From the customer’s side there were five people who contributed to project. One person was named to be the contact person and four others participated in requirements gathering. One of them was the administrator of the current system, two of them users of the user interface and contact people of the systems that used the services of code register. One person was the database manager.

The process method for the project was mostly waterfall model. The project was planned, the requirements were gathered, the design was planned and tested and construction and test phases followed each other sequentially.

The requirements phase was performed by interviewing the customer. Because of the geographical distance and need to limit expenses it was decided that no meetings with all customers was held. Instead several meetings were held with the contact person: the idea was that she gathered the requirements and needs of all the customers and brought their needs to the project group within these meetings. Two meetings where one of the project managers, customer's database manager and contact person participated were held. Two meetings were also held to plan the user interface of the new system. The users of the current code registers user interface, the contact person and the administrator of the current system attended to those meetings.

The functional and technical requirements were gathered and analysed by the project group with the help of the customer. They were also prioritized together with the customer. As a result of requirements elicitation and analysis the document Requirements specification was written. The customer reviewed and accepted it as all the documents during the project.

After the requirements elicitation phase one of the project members was not available for the project any more. It was not possible to get a substitute so the only possibility was to reduce the amount of work. It was difficult to decide what to leave out from the project and it turned out that the prioritization of the requirements was not waterproof. In the end it was decided the sub grouping features of the code lists were to be left out even though they were prioritized to 5.

The total work amount used for the project was 1046.

5.2. Motivation to use QFD in code register project

The Code register was chosen to be the object of case study for several reasons. First of all it was a small scale project: the amount of work was estimated to be only 1700 hours of work. The amount of functional requirements was 28 and the UI screen to be needed was about 10. The scope of the research was to concentrate especially on gaining experience of usage of QFD in small projects.

The other reason was that there is a current code register system which is in use. There also was the previous project only a year ago that created a new code register system with different architecture but the project couldn't produce a new complete system to be deployed. The requirements elicitation, analysis and prioritization weren't done well enough thus there is an opportunity to get experience how QFD would have affected to the result.

A lot of work was also done during the code register project which could be reused. Since most of the requirements were still valid the question was mostly to check them to find the real motivation for the requirement. The customers had to be interviewed also to confirm if some new requirements had occurred. All the requirements had to be analysed according to methods of QFD.

In this research the requirements have to be analysed again according to methods of QFD and using the House of Quality. The customers have to be interviewed in order to prioritize the requirements. The technical requirements have to be determined according to QFD method and correlated with the customer needs. The technical methods have to be prioritized and targets for them have to be set. Finally the technical interactions have to be identified. The difference in priorities compared to first project will be analysed and the time consumed in requirements analysis and QFD will be measured.

5.3. Research process and data collection

The research process is illustrated in Figure 18. The research did not proceed as a clear linear process. Some tasks were done iteratively and some parts of the process took place iteratively.

The study started by gathering information of QFD and software project processes and quality aspects in software project work. The theory was gained through that information. After the case for the case study was chosen the documentation from previous project and current system of the use case was gathered. The situation of current system was gained by interviewing the customer.

The QFD process had to be adjusted to the case study. The tasks of QFD are explained in detail in chapter 6.1 Adjustments were done parallel with implementing QFD.

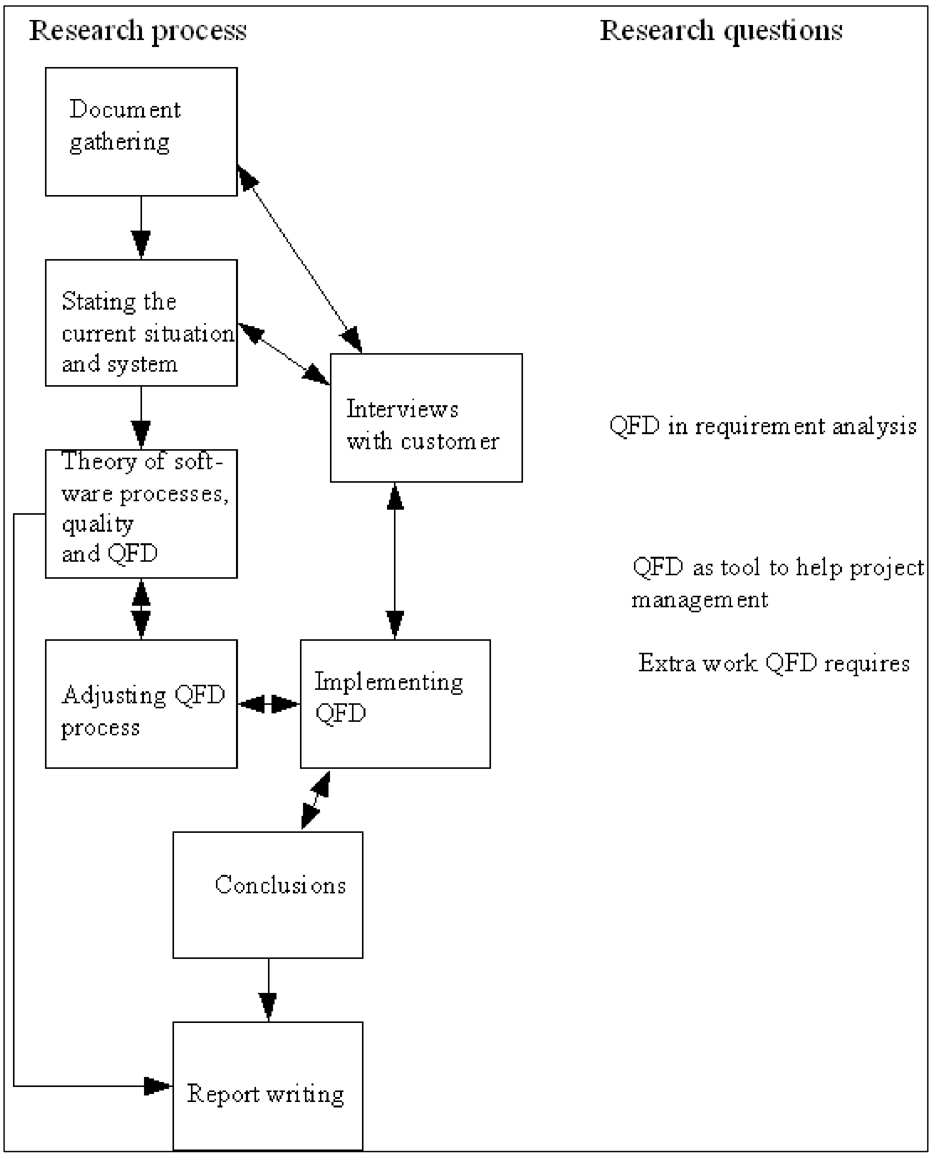

Figure 18. Research process.

Since the previous project had produced a lot of useful documentation it was actually the main source of data. The requirements specification was the most important source of information. A lot of useful information could also be found from the final report and from some notes of the project meetings. Another method to gain information was the communication with the customer. Most of the communication was arranged as reviews, since a lot of time could be saved doing work beforehand. Some reviews were held through email and some as face to face situations.

6. House of Quality in use

6.1. Description of the work flow

The use of QFD contains several tasks as described in chapter 4. While creating the House of Quality in new project constant interaction and interviews with customer are essential. Since there was a previous project not long ago with the same scope, a lot of written material can be used instead. The work is more adjusting the previous work into QFD and checking from the customer that they agree. The tasks of QFD process will be carried out as follows.

First the group of customers has to be checked in order to verify that all the aspects are taken into account. The gathering of customer requirements was already done, but they were analysed by the previous project team and some of the requirements were changed so that the real need of customer is not detectable any more. Also the requirements have to be grouped according to QFD method. It is not possible to have a QFD group for the work. Most of the work and sessions are arranged by me as well as creating and updating the matrices.

The prioritization of requirements has to be done again, because some new requirements might appear during the interviews with the customers.

Technical requirements will have to be gathered almost from scratch because the concept of technical requirement is different in QFD. Yet the requirement specification from previous project is useful.

Composing the House of Quality matrix and analysing the results are all new tasks and are done in sequence after developing the technical requirements. Chang and Wu [2002] have drawn the House of Quality matrix as in Figure 19.

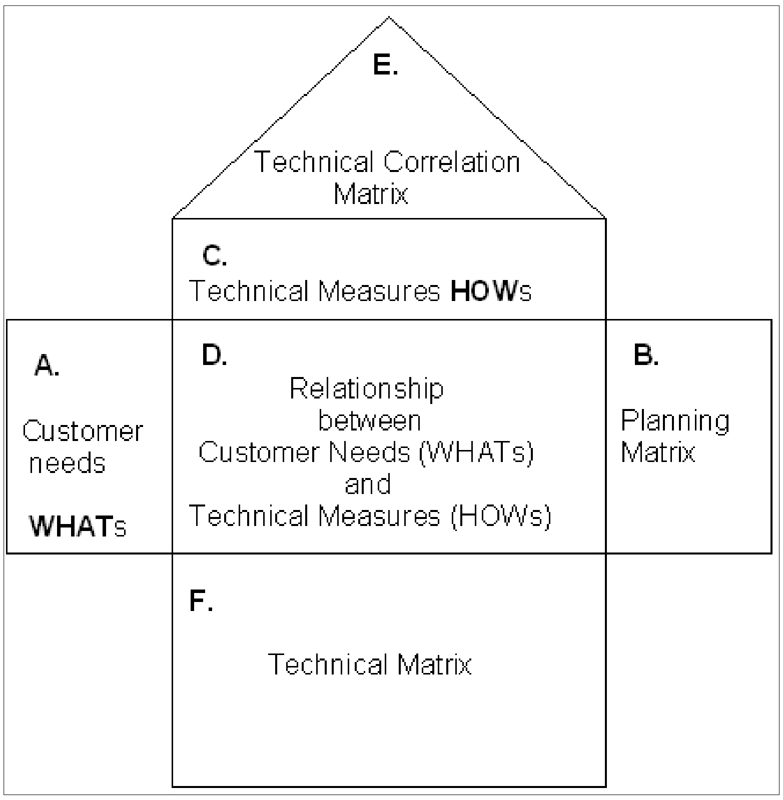
E.
Technical Correlation Matrix
C.
Technical Measures HOWs
A.
Customer needs
WHATs
D.
Relationship between Customer Needs (WHATs) and Technical Measures (HOWs)
B.
Planning Matrix
F.
Technical Matrix

Figure 19. House of Quality.

In the figure 19 the different tasks are identified more clearly than in figure 16 in previous chapter. The order of the letters from A to F also expresses the presumable order of work. The tasks A and B are described in chapter 6.3. The task C is described in chapter 6.4. Task D is described in chapter 6.5 and task E in chapter 6.6. The letters (A-F) are used in following chapters to indicate the different matrixes.

6.2. Stakeholders of Code register

The most important stakeholders of the code register could be classified into four different categories and types of stakeholders:

Category	**Type**	**Description**
Administrator	Customer/ Domain expert	Administrator of code register: adds new code lists, controls the access to code lists.
UI-users	End user	Users who maintain and add codes to code lists and maintain subgroups of code lists through user interface.
Administrators of other systems	Neighbouring systems	Users who are responsible of other it systems which use services of code register.
Database administrator	Customer	Technical person that is responsible of doing changes to databases and administering them.
Administrator of the interfaces to other systems	Customer	Technical person who is responsible for updating and maintaining the interfaces of code register to provide services to other systems.
Software Architect of the customer	Regulator	Provides guides and regulations of the architecture of software systems.

Table 3. Stakeholders of the Code register.

All categories of stakeholders were taken into consideration already in previous project.

Since the project was small there were not possibilities to have a group of people in QFD team. Typically in a small project there is only one or two IT people working with the requirements and they often are responsible for technical aspects and requirements. Thus most of the QFD work was done by me with the help of one person who works with current code register as an administrator of the interfaces to other systems. Her role was mostly to discuss and assess the results and ideas of QFD.

6.3. Determining the customer needs

Since during the previous project all the stakeholders were interviewed or taken into consideration otherwise, most of the requirements stated in the requirements specification document were still valid. The requirements from previous project are in Appendix A. They are grouped into main functions f. ex. *1. Logging* which contain three requirements: *1.1 Logging in, 1.2 Logging out* and *1.3 Authorization.* After each requirement the degree of importance of requirement is written in parentheses.

The problem with requirements was that the project group had analysed the requirements and the real need of the customer was not always evident - the voice of the customer was not quite audible or clear any more. Thus the old memos of the meetings with customer were useful source of information. Also some discussions with customers were necessary.

After listing the underlying requirements and interpreting them into clarified and more general requirements the amount of requirements was 35. A list of that size is difficult to analyse especially if it is wanted to remove redundancies or to find missing statements. It was also evident that some requirements were quite specific while others more general. In order to obtain better overview the requirements were arranged into groups and embedded together. The result is illustrated in Appendix B. There are two levels of requirements. The lower level contains all the previous and new requirements. They are identified as A.1 which means that the requirement belongs to higher level requirement A. The identification in parentheses (1.1) is from the previous project meaning that the requirement belonged to group 1 which can be identified in Appendix A.

The higher level consists of general items of the lower list or added topics which describe or generalizes the lower level requirements. The amount of higher level requirements is only 8. The reduced amount is easier to handle in matrices; it keeps the size of them decent. And in literature it is noted that it is reasonable to try to reduce the size of them. As a result of grouping it is evident that the grouping is different from grouping of previous project. In previous project the grouping of requirements was mostly done from the implementations point of view, not from the need of customer. The requirements after grouping are in Appendix B.

After regrouping of requirements the customers were interviewed again and they were also asked to determine the importance of the requirements. The preliminary value for each requirement was copied from requirements specification document of the previous project. The preliminary value was not added to new requirements. The scale to prioritize the requirements was the same as in previous project: linear from 1 to 5 - the most important requirements got the value of 5. The customers prioritized the lower level requirements. The value that was given to higher level requirements was counted as a rounded average of the values of lower level requirements.

After prioritizing the customer needs the present system was evaluated with the customer needs. Usually in QFD the competitive products are valued but since the code register is a customized system there is no products in markets where to compare to. The system evaluation contains two additional columns for each of the user requirement: system at present and at planned level. The same scale 1 to 5 was used in both columns. The customer needs with all the described attributes are presented in Table 4 and are also in B part of the House of Quality matrix.

Customer need	**Degree of importance**	**System now**	**Planned level**	**Improvement ratio**	**Importance weight**	**Relative weight**
A. Users are authorized to do different tasks	5	2	5	2,5	12,5	11,8
B. The system must be adaptable	5	1	4	4	20	18,9
C.The data is reliable	5	1	4	4	20	18,9
D. Codes or codelists that have been used by other systems or users must not be deleted except by administrator in case of severe errors.	4	4	4	1	4	3,8
E. It must be possible to group the codes in a codelist into subgroups	4	4	4	1	4	3,8
F. It must be simple to use	3	1	4	4	12	11,4
G. It must be possible to create an adaptable service interface through which other systems may interact with the code register	5	1	5	5	25	23,6
H. The administration of user roles must simple	5	3	5	1,7	8,3	7,8

Table 4. The customer needs.

In the first column of the table the customer needs are listed. The column Degree of importance indicates the customer's point of view on the relative importance of these needs. The column System now indicates how well the present system corresponds to the needs of customer. The column Planned level indicates how well the new system is planned to meet each of the requirements. This figure is given from the point of view of the implementing the requirements - also technical factors are taken into consideration. It determines the improvement goals.

The column in the Table 4 named Improvement ratio is a result of planned level divided by the system now of the same requirement:

Improvement ratio = Planned level / System now

The column Importance weight of requirements is determined by multiplying the degree of importance by the improvement ratio. In other words:

Importance weight = Degree of importance * Improvement ratio

Emphasis is therefore placed on the most important customer requirements.

The last column Relative weight is the importance weight normalized to 100 %. In other words:

Relative weight = Importance weight / Σ Importance weight *100

The relative weigh is used in matrix as a multiplier to the correlations. With this method the customer needs have weightings based on the customer's preferences and the level of needed improvement. Thus the weighting indicates not only customer's voice but also what is considered to require most improvement from the system developers' point of view.

6.4. Determining technical requirements

For achieving a complete list of technical requirements the requirements specification document from previous project can be used to some extent. Only few of the technical requirements can be used as they were. The customer had specific guides of the architecture of software and user interfaces and these turned out to be useful source of information which generated new technical requirements.

The problem with categorizing the requirements to technical and non technical is that the customer actually required some technical features which in the QFD method are not considered technical requirements f. ex. "The database should be planned so that it is possible to implement an efficient service interface through which other systems may interact with the code register".

The technical requirements have to be measurable so that can be controlled. Finding measurable technical requirements is not easy. But in order to be controlled each requirement has to have a target value that is planned to be achieved in future. To clarify the requirements an arrow is added for pointing which direction is to good. If the target is a fixed value it is written instead of an arrow. The technical requirements are listed in Table 5.

Technical requirement	Unit	Target
New roles can be added without changes in source code	Number of changes	↓
Speed of user interface to fetch data for other systems	Time	↓
Layout designed according to standard of customer	Number of deviations	↓
Ease of submitting	Minutes	↓
Flexibility to store different kinds of codes	Number of types of codes that can not be stored	↓
Efficient userinterface	Seconds for a page to	< 10 s
Flexible architecture according to standard of customer	Numeber of deviations	↓
Flexible interface to other systems	Number of chances to interface when new code is added	↓
Layered architechture	Changes to code if database is changed to another	↓
Support to IE 5.0 and newer versions	Number of differences in behavior of different versions	↓

↑	Aim to maximise
↓	Aim to minimise

Table 5. The technical requirements.

6.5. Composing the D part of the House of Quality matrix

The construction of the House of Quality matrix is done by placing the customer needs as rows and the technical requirements as the columns of the matrix. Each of the cells is filled with a symbol that indicates how strong the correlation between the customer need and the technical requirement is. By giving these symbols a numeric value the customer priorities to technical requirements can be calculated.

The matrix is created with the information of previous phases. The customer needs are placed as rows in the matrix and the technical requirements as columns. Also the indicators of the customer needs - degree of importance, system now, planned level, improvement ratio, importance weight and relative weight - are added as columns.

Each technical requirement can correlate customer need at one of four levels: strong, moderate, weak or non-existent. For non-existent relation the cell is left blank, weak is marked with ◘, moderate with ○ and strong with ●. Two sets of measurement scales may be employed to quantify the relationships: (0, 1, 3, and 9) and (0, 1, 3, 5) [Chan and Wu, 2002]. The former scale is more frequently used and seems more suitable since it assigns a much higher weight to the strong relationship that is indeed much more important in the product development process. Thus numeric values used for correlations are 9,3,1 and 0. The correlations are determined by asking how significant is technical requirement A in satisfying customer requirement B. It should be noted that the correlations can also be negative which means that improving the specific technical requirement hinders to achieve the corresponding customer need. The cells of the Figure 20 which have dark background indicate negative correlation.

The correlations are determined in two steps. First the matrix was filled and then discussed with a customer. Most of the correlations were easy to determine and the whole process took only 4 hours. The discussion with customer didn't lead into any changes yet it revealed some useful ideas how to improve the usability of the system.

Figure 20 presents the parts A,B,C and parts of the House of Quality matrix. The importance weight of the technical requirement is calculated by multiplying the correlation value of each cell with its corresponding relative weight and summing the result of each individual cell in the column:

Importance weight of technical requirement = Σ(Correlation value * Relative weight)

The negative correlation was not noted in the calculations - the absolute values were used. Thus the importance weight of the technical requirement means how important those requirements are to be controlled. It does not represent how important those are to be improved.

The relative weight of technical requirement was calculated as:

Relative weight of technical requirement = Importance weight of of technical req. / Σ Importance weight of technical req. *100

After completing the matrix it was verified and cross-checked with the customer to see that the information put in and the weightings it yielded made sense.

The matrix looked amazingly logical. There weren't similar rows which could have meant duplicate customer needs or not clearly defined requirements. Also the columns of table were unique which implied that the technical requirements were unique also.

There were only one technical requirement that had impact on customer needs "It must be possible to group the codes in a code list into subgroups" and "The administration of user
roles must be simple". Those customer needs had also the lowest priorities thus it is acceptable that there are few implementations supporting them.

	Technicaal requirements / **Customer needs**	1 New roles can be added without changes in source code	2 Speed of user interface to fetch data for other systems	3 Layout designed according to standard of customer	4 Ease of submitting	5 Flexibility to store different kinds of codes	6 Efficient userinterface	7 Flexible architecture according to standard of customer	8 Flexible interface to other systems	9 Layered architechture	10 Support to IE 5.0 and newer versions	**Degree of importance**	**System now**	**Planned level**	**Improvement ratio**	**Importance weight**	**Relative weight**
A	Users are authorized to do different tasks	●			○		◘ ▨					5	2	5	2,5	12,5	11,8
B	The system must be adaptable	●	○ ▨		○ ▨	●	○ ▨	●	●	●		5	1	4	4	20	18,9
C	The data is reliable				◘	○						5	1	4	4	20	18,9
D	Codes or codelists that have been used by other systems or users must not be deleted except by administrator in case of severe errors.		○ ▨		○		○ ▨					4	4	4	1	4	3,8
E	It must be possible to group the codes in a codelist into subgroups					●						4	4	4	1	4	3,8
F	It must be simple to use	○		●	●	○ ▨					○	3	1	4	4	12	11,4
G	It must be possible to create service interface through which other systems may interact with the code register		●			○ ▨			●			5	1	5	5	25	23,6
H	The administration of user roles must simple	●										5	3	5	1,7	8,3	7,8
															Total	105,8	100
	Importance Weight:	381	280	103	193	366	80	170	382	170	34						
	Relative weight (%):	17,6	13	4,8	8,9	16,9	3,7	7,9	17,7	7,9	1,6						

●	Strong
○	Moderate
◘	Weak
▨	Negative

Figure 20. The A,B,C and D parts of the House of Quality.

6.6. Technical feature comparison

The purpose of this phase is to find out how the technical requirements interact with each other. This phase is important especially for designers of the system because with it helps them to understand how the technical features are connected to each other: if one part of the system is changed how it does affect other parts? Each of the correlations can be either positive or negative and strong or weak or it is possible that technical features have no correlation at all. The positive correlation means that improving one technical feature improves also the other technical feature. The negative correlation means that improving the one technical feature dilutes the other technical feature.

The symbols used to describe correlations are:

+	weak positive correlations
++	strong positive correlations
-	weak negative correlation
- -	strong negative correlation
	no correlation

The matrix was filled the same way than previous parts of matrix. The matrix is shown in Figure 21.

	Unit	**Target value**	**Target movement**	New roles can be added without changes in source code	Speed of user interface to fetch data for other systems	Layout designed according to standard of customer	Ease of submitting	Flexibility to store different kinds of codes	Efficient userinterface	Flexible architecture according to standard of customer	Flexible interface to other systems	Layered architechture	Support to IE 5.0 and newer versions
New roles can be added without changes in source code	Changes		↓										
Speed of user interface to fetch data for other systems	Time		↓	-									
Layout designed according to standard of customer	Deviations		↓										
Ease of submitting	Minutes		↓	+		++							
Flexibility to store different kinds of codes	Codetypes		↓	++	-		-						
Efficient userinterface	Seconds	< 10 s		-			++	-					
Flexible architecture according to standard of customer	Deviations		↓	+	-	+		+	++				
Flexible interface to other systems	Changes		↓	++	-			+		++			
Layered architechture	Changes		↓			+		+	++	++	++		
Support to IE 5.0 and newer versions	Differences		↓			+	+		+				

Figure 21. Technical feature comparison.

During the discussion with the customer some technical requirements had to be rephrased in order to avoid misunderstandings. The requirements "Flexible architecture according to standard of customer" and "Layered architecture" caused some discussion, since the first requirement actually includes the second one. Yet they were left as they were since the layered architecture was especially important with this system.

The speed of user interface to fetch data for other systems was regarded quite important. Yet there are three technical features that can hinder the speed. Thus it is important to have close communication and collaboration while designing those features.

6.7. Analysing the results

The most useful result of QFD is the prioritized custom requirements and the technical requirements which are listed in Table 6.

No	Technical requirement	Weight	Weight(%)
1	Flexible interface to other systems	382	17,7
2	New roles can be added without changes in source code	381	17,6
3	Flexibility to store different kinds of codes	366	16,9
4	Speed of user interface to fetch data for other systems	280	13
5	Ease of submitting	193	8,9
6	Flexible architecture according to standard of customer	170	7,9
7	Layered architechture	170	7,9
8	Layout designed according to standard of customer	103	4,8
9	Efficient userinterface	80	3,7
10	Support to IE 5.0 and newer versions	34	1,6

Table 6. Technical requirements with their relative weights.

The relative weights of the technical requirements indicate how important it is to control these requirements in order to meet customer's expectations. The customers were not surprised by the priorities and they accepted them as a basis for future actions.

Combining the information of this table to information of Table 6 several conclusions can be made. The requirement number 1 "Flexible interface to other systems" turned out to be the most important technical feature to control and it also seems to be affecting positively to technical requirements numbers 1,5 and 7 as seen in Table 6 so it should be considered as a quite profitable requirement to fulfil. The requirement 7 "Layered architecture" is not specially important requirement but if checked from the Table 6 it has positive correlation to many other requirements, which makes it interesting to fulfil.

7. Summary

7.1. General experiences of QFD

The purpose of this study was to clarify what kind of benefits the use of QFD - especially the House of Quality model - offers during the requirements analysis phase in a small-scale project. For this purpose a QFD was used for a project which aimed at producing a new Code register for the customer. The activities of software project for which QFD was used were communication with stakeholders which included requirement acquisition and planning.

The Code register is customized software which is used by one customer organization. There had been a project year ago during which it was meant to be rebuilt. Unfortunately because the lack of resources the new system was not finished completely and it was not taken into production.

The typical situation for small scale projects is that there are not many resources available. Thus there were very limited resources available for using QFD method. The QFD analysis was carried out using the results of previous project and consulting mostly one member of customer who represented administration of interfaces to other systems as well as product development. However all the results of work were also discussed with the group of customers which involved product developers, users and administrators to ensure that the voice of customer was heard.

QFD fitted well to a small scale project since the amount of requirements was quite reasonable. In a large project the importance of grouping the requirements becomes more evident.

7.2. QFD as a method to improve requirements

The customer needs were collected from the previous project and interviewing the stakeholders. The documentation of the previous project turned out to be quite valuable and there weren't much changes compared to the previous customer requirements. The amount of requirements was quite large and in order to make matrices easier to handle the requirements were grouped. Grouping was not quite an easy task. It is difficult to generate requirements which are of same level in accuracy not too specific and not too general.

The requirements could also be acquired the other way - from general requirements to more detailed. That approach is more usual when new software is being developed. In practise it seems that requirements concerning software which is meant to be replacing existing software tend to be quite detailed. The reason for that is that the users know already the features and defects of existing software. But if new software was being under construction and the general requirements would have been there first, it would be necessary also in that case to bring them into more detailed level. The reason to that is that there might always be some underlying needs hiding behind the general requirements and in order to gain consistent and unambiguous requirements also the detailed requirements are necessary to obtain.

The decision to keep matrices rather small by arranging the customer needs in a hierarchical structure was justifiable yet it produced some difficulties. The customer had in mind the lower level requirements and it was not always obvious under which higher level requirement it belonged. The tree diagram had to be available always when discussing with the customer.

Yet the technical requirements were more difficult to develop since the results of previous project could be used only to some extent. The reason to that was that the concept of technical requirement is different in QFD. Most work was done by me because of the limited resources but more brainstorming with a group could have resulted into a better outcome.

7.3. QFD as a method to help project management

The determination of correlations between technical requirements and customer needs was interesting phase. It gave a new view of point to requirements which was totally new compared to previous project work. The result matrix was the most interesting result of QFD especially for the decision making in the project. The result revealed what are the real impacts that the technical requirements have on fulfilling the customer needs. During the previous project when it was obvious that some work had to be left undone because of the lack of resources the decision were based on discussions and 'hunch'. There was no other documentation to rely on than the prioritizations of requirements which were done by the customer. The decision in previous project to leave the user need E 'It must be possible to group the codes in a code list into subgroups' was actually not a bad decision even though it was prioritized to 4. From the correlation matrix can be seen that there was only one technical requirement that affected it and on the other hand that technical requirement could also be achieved through other customer needs as well.

Using QFD there is more material for the project manager to base the decisions on. One of the most valuable document to base decisions on is the Technical Correlation matrix (E). From it the affects of single technical requirement to other technical requirements is visible. On the other hand the technical requirements which affect a single technical need can also be seen. It is a valuable document to see the side effects the improvement of technical requirement causes.

7.4. Workload of QFD

In literature one of the downsides of QFD is often mentioned to be that is rather time consuming. And especially in a small scale there is no sense to spend much time using it. In previous Code register project the total hours used for the project by project team was 1046 from which requirements analysis took 118 hours. That amount does not include the hours that the customer used. The hours used for requirement analysis using QFD are in Table 7.

Phase	Time (h)
Customer requirements	
- collecting	8
- grouping & prioritizing	3
- meetings	4
Technical requirements	
- collecting	3
- prioritizing	4
- meetings	3
House of Quality	
- creating	5
- meetings	3
Tecnical feature comparison	
- creating	3
- meetings	2
Finalizing	2
Total	**40**

Table 7. Work hours used for requirements analysis using QFD.

The increase of work for requirements analysis because of using QFD is 25 % and if compared to total work of the previous project the increase of work is 3,6 % which is not much compared to benefits.

On the whole QFD seems to be a tool to offer additional systematics in the software development. It should be fitted tightly to software development process instead of having it as a separate analysis. It seems to be most valuable when creating general software but it offers also tool to systematise a project for producing customized software. It can also be used to compare how much improvement a new software is considered to produce according to customers compared to the previous system. It is a systematic method which can be considered as a tool to see the requirements from different aspects.

7.5. QFD and software process model

The development process model used in Code register project was waterfall model. QFD was easy to use in that model since all the requirements were gathered, grouped and analysed during one phase. QFD is a tool to analyze the requirements and if not all the requirements are not acquired and analyzed together the matrixes have to be redone every time new requirements appear.

Therefore QFD seems to fit well into development processes where all the requirements are gathered and analyzed once. Since agile method base on changeability and flexibility the use of QFD in agile process methods is more difficult and laborious. All matrixes have to be rechecked every time the requirements are added or changed.

7.6. Generalization of the results

This is a case study of one case thus generalisation of the results is not straightforward. The results should be considered more like an suggestive example. The main problem with empirical generalisation from studied to unstudied cases is that it is potentially subject to high and unknown levels of error. This problem obviously increases as the heterogeneity of unstudied cases increase. The problem with the software engineering is that the projects are never the same. In fact they can be considered very heterogeneous. Yet in this thesis QFD is used only to requirements analysis phase and the use is restricted to small-scale projects. Thus some suppositions are justified.

The case study is of a software project which had been done year ago and much of the results of that project could be reused although the software was never deployed. The scope of the previous project was to rebuild existing software. That situation affected to requirements so that the user could give quite specific requirements since they knew the system and its defect well. Also some of the users were quite technically oriented which also affected the result.

The benefits of the situation of previous project were that comparing the work amount was possible. On the other hand the customer requirements had to be analyzed again and if that work would have been done from the beginning according to QFD method it would have been unnecessary.

It would also have been interesting to have the work hours that the customer used for the project in order to get the whole picture. But it was impossible to get that information from the previous project.

This project was done mostly by one person so the result might have too one-sided.. QFD method is at its best if there are more people involved. Brainstorming sessions bring more aspects to problems.

References:

[Abran, Moore (eds) et al., 2004] Alain Abran, James W. Moore (eds) et al., *Guide to the Software Engineering Body of Knowledge: 2004 Edition: SWEBOK.* IEEE Computer Society, 2004.

[Ambler and Constantine 2000] Scott W. Ambler and Larry L. Constantine, *The Unified Process Elaboration Phase: Best Practices in Implementing the UP.* CMP Books, 2000.

[Bechtold, 1999] Richard Bechtold, *Essentials of Software Project Management.* Management Concepts, 1999.

[Bentley, 2006] Colin, Bentley, *PRINCE2 Revealed: Including How to use PRINCE2 for Small Projects.* Butterworth-Heinemann, 2006.

[Boehm, 1986] B. A Boehm, Spiral Model for Software Development and Enhancement. *Computer* **11,** 4, (May 1988), 61–72.

[Boehm, 1981] B. A Boehm, *Software Engineering Economics.* Prentice-Hall, Englewood Cliffs, NJ, 1981.

[Buehring, 2006] Simon Buehring, Managing small projects. ITtoolbox Research White paper. 2006. Also available as *http://hosteddocs.ittoolbox.com/SB10306smallprj.pdf.*

[Capability Maturity Model® Integration, 2002] Capability Maturity Model® Integration (CMMISM). Version 1.1, Software Engineering Institute, 2002.

[Chan and Wu, 2002] Lai-Kow Chan and Ming-Lu Wu, Quality Function Deployment: A Comprehensive Review of Its Concepts and Methods. *Quality Engineering* **15,** 1 (2002), 23–35.

[Chatfield and Johnson, 2004] Carl Chatfield and Timothy Johnson. *Microsoft Office Project 2003 Step by Step*, Microsoft Press, 2004.

[Connors, 1992] Danny T. Connors, Software development methodologies and traditional and modern information systems. *Software Engineering Notes* **17,** 2 (Apr 1992).

[Davis, 1993] Alan M. Davis, *Software Requirements: Objects, Functions, and States.* Prentice-Hall, 1993.

[Duggan and Reichgelt, 2006] Evan W. Duggan and Han Reichgelt, *Measuring Information Systems Delivery Quality.* IGI Publishing, 2006.

[ESA Board for Software Standardisation and Control,1996] *Guide to applying the ESA software engineering standards to small software projects*. BSSC(96)**2** Issue 1. ESA Board for Software Standardisation and Control (BSSC), European Space Agency, 1996. Also available as *http://styx.esrin.esa.it/premfire/Docs/Bssc962.pdf.*

[Firesmith, 2003] Donald G. Firesmith, Specifying Good Requirements. *Journal of object technology* **2**, 4 (July-August 2003), 77-87.

[Forselius, 1999] P.Forselius, Ohjelmistojen koon mittaaminen erityyppisissä hankkeissa. *Systeemityö* **1** (1999).

[Geras *et al.*] Geras Adam, Zannier Carmen and Davis Guy, Quality Function Deployment for Requirements Engineering. SENG 613:QFD Group. Available as *http://www.guydavis.ca/seng/seng613/group/qfd.shtml#1.1.*

[Haag, Raja and Schkade 1996] Haag, S., Raja, M.K., Schkade, L.L., Quality function deployment usage in software development. *Communications of the ACM* **39**, 1 (1996) 41-49.

[Hierholzer, Herzwurm and Schlang, 1998] Andreas Hierholzer, Georg Herzwurm and Harald Schlang, Applying QFD for Software Process Improvement at SAP AG. *In Proceedings of the World Innovation and Strategy Conference in Sydney, Australia,* August 2-5, 1998, 85-95.

[Huber, 2003] Nick Huber, Hitting targets? The state of UK IT project management. *Computer Weekly*, 2003. Also available as *http://www.computerweekly.com/Articles/2003/11/05/198320/hitting-targets-the-state-of-uk-it-project-management.htm.*

[Jacobson, Booch and Rumbaugh, 1998] I. Jacobson, G. Booch, and I. Rumbaugh, *The Unified Software Development Process*. Addison-Wesley, Boston, 1998.

[Javed, Maqsood and Durrani, 2004] Talha Javed, Manzil e Maqsood, Qaiser S. Durrani, A Study to Investigate the Impact of Requirements Instability on Software Defects. *ACM Software Engineering Notes,* ACM Press **29,** 3 (2004).

[Koch, 2005] Alan S. Koch *Agile Software Development: Evaluating the Methods for Your Organization.* Artech House, 2005.

[Kotonoya and Sommerville, 1998] Kotonoya Gerald and Sommerville Ian, *Requirements engineering.* John Wiley & Sons, 1998.

[Leon, 2000] Alexis Leon*, A Guide to Software Configuration Management.* Artech House, 2000.

[Lichter, Schneider-Hufschmidt and Züllighoven, 1993] Horst Lichter, Matthias Schneider-Hufschmidt and Heinz Züllighoven, Prototyping in Industrial Software Projects - Bridging the Gap Between Theory and Practice. *Proceedings of the 15th international conference on Software Engineering ICSE '93.* IEEE Computer Society Press. (May 1993).

[Lillrank, 1990] Paul Lillrank, *Laatumaa: Johdatus japanin talouselämään laatujohtamisen näkökulmasta.* Gaudeamus 1990.

[Liu et al., 2006] Frank Liu, Kunio Noguchi, Anuj Dhungana, V.V.N.S.N. Srirangam A. and Praveen Inuganti, A quantitative approach for setting technical targets based on impact analysis in software quality function deployment (SQFD). *Software Qual J* **14,** 2 (2006), 113–134.

[Liu, Sun, Cane, 2005] Xiaoqing (Frank) Liu, Yan Sun and Gautam Kane, QFD Application in Software Process Management and Improvement Based on CMM. In: *International Conference on Software Engineering*, 1 – 6.

[Luckey and Phillips, 2006] Teresa Luckey and Joseph Phillips, *Software Project Management for Dummies.* John Wiley and Sons, 2006.

[Mangione, 2003] Carmine Mangione, Software Project Failure: The Reasons, The Costs. Available as *http://www.cioupdate.com/reports/article.php/1563701.*

[Masum, Morshed and Mitsuru, 2004] Shaikh Mostafa Al Masum, A.S.M. Mahbub Morshed, and Ishizuka Mitsuru, Object Oriented Hybrid Software Engineering Process (SEP) Model for Small Scale Software Development Firms. *Proc. 2nd*

Int'l Conf. on Computer Science and its Applications(ICCSA-2004), San Diego (2004.6) . Also available as *http://www.miv.t.u-tokyo.ac.jp/papers/mostafa/ ProcessModelSSS_Mostafa_ICCSA2004.pdf.*

[McConnell, 2006] Steve McConnell, *Software Estimation: Demystifying the Black Art.* Microsoft Press, 2006.

[Perry, Sim and Easterbrook, 2004] Dewayne E. Perry, Susan Elliott Sim and Steve Easterbrook, *Case study for software engineers.* Available as *http://users.ece.utexas.edu/~perry/work/papers/DP-04-icse04.pdf.*

[Pohl, 1993] Klaus Pohl, Requirements Engineering: An Overview. Informatik V, RWTH Aachen, Germany, 1996. Also available as *http://ftp.informatik.rwth-aachen.de/ftp/pub/packages/CREWS/CREWS-96-02.pdf.*

[Potter, 2005] Neil Potter, Tracking project size attributes to monitor project progress. *The Process group Post* **12,** 1 (2005). Also Available as *www.processgroup.com.*

[Project Management Institute, 2004] Project Management Institute, *A Guide to the Project Management Body of Knowledge (PMBOK® Guide), Third Edition.* Project Management Institute , 2004.

[Rajala, 2004] Erkki Rajala, Onnistuminen projektissa. *HETKY, Helsingin Tietojenkäsittely-yhdistys ry:n jäsenlehti,* p. 12 3/2004.

[Richardson, Murphy and Ryan, 2002] Ita Richardson, Eamonn Murphy and KevinRyan, Development of generic quality function deployment matrix. *Quality Management Journal* **9,** 2 (2002), 25-43.

[Schmidt, 2000] Michael E.C. Schmidt, *Implementing the IEEE Software Engineering Standards*. Sams, 2000.

[Shahin, 2005] Arash Shahin, Quality Function Deployment: A Comprehensive Review. 2005. Available as *http://www.dci.ir/ravabet/f/shahin.pdf.*

[SMS, 2004] Software Measures Ltd. (SMS), 2004. Available as *http://www.measuresw.com/library/Papers/Rule/RulesRelativeSizeScale%20v1b.pdf.*

[Snedaker, 2005] Susan Snedaker, *How to Cheat at IT Project Management.* Syngress Publishing, 2005.

[Sommerville and Sawyer, 1997] I. Sommerville and P. Sawyer, *Requirements Engineering: A Good Practice Guide.* John Wiley and Sons, 1997.

[Thayer and Christensen, 2005] Richard H Thayer and Mark J. Christensen, *Software Engineering, Volume 1: The Development Process, Third Edition.* John Wiley and Sons, 2005.

[Tomayko and Hazzan, 2004] James E. Tomayko and Orit Hazzan, *Human Aspects of Software Engineering.* Charles River Media, 2004.

[Tsui, 2004] Frank Tsui, *Introduction - What is Software Project Management? Managing Software Projects.* Jones and Bartlett Publishers, 2004.

[Trengove and Dwolatzky, 2004] E. Trengove and B. Dwolatzky, A software development process for small projects. *International Journal of Electronical Engineering Education* **41,** 1 (2004) 10-36.

[White and Fortune, 2002] Diana White and Joyce Fortune, Current practise in project management - an empirical study. *International Journal of Project Management* **20,** 1 (2002) 1-11.

[Whittager, 1999] Brenda Whittager, What went wrong? Unsuccessful information technology projects. *Information Managent & Computer security* **7,** 1 (1999) 23-29.

[Wiegers, 1999] Karl E. Wiegers, First Things First: Prioritizing Requirements. *Software Development*, September 1999. Also available as *http://www.tarrani.net/linda/prioritizing.pdf.*

[Wiegers, 2003] Karl E. Wiegers, *Software Requirements, Second Edition.* Microsoft Press, 2003.

[Wysocki, 2006] Robert K. Wysocki, *Effective Software Project Management.* John Wiley & Sons, 2006.

[Zultner, 1992] Richard Zultner, Quality Function Deployment (QFD) for software. American Programmer, 1992.

Appendix A: Customer requirements from previous project

Requirements

These requirements are the original requirements which were original acquired from the customer but analysed by the project group. The requirements were prioritized into five categories: critical(5), high(4), normal(3), low(2), no importance(1). The prioritization value is marked after the title of requirement.

1. Logging

1.1. Logging in (5)

Users have log in to the system. Unauthorized users are not allowed to enter any parts of the system.

1.2. Logging out (5)

Users can log out from the system at any moment.

1.3. Authorization (5)

Authorization information is saved and controlled in another system which is connected to the system

2. Creating a new code list

2.1. Creation of new code list (5)

The administrator of the system can create a new code list and enter general attributes (code, name, start date, end date, user name, chance date, explanation).

2.2. Adding extra attributes (5)

At the creation of code list the administrator may add extra attributes to code list. Each attribute has a title and the type of attributes can be either date, text, numeric or decimal.

2.3. The code lists have created in two phases (4)

The creation of code list has to be done in two phases. First the code list along with its attributes is created. After creation it has to be confirmed. The code list can be used only after confirmation and no attributes can be deleted any more.

3. Adding codes to code list

3.1. Adding code (5)

The user that is authorized to add codes to a code list may add codes.

3.2. Adding many codes (3)

The user that is authorized to add codes to a code list must be able to save several codes at a time.

3.3. Changing the end date of the code (5)

After confirmation of the code the authorized user may change only the end date of the code, no other attributes.

4. Browsing the code lists

4.1. Listing the code lists (5)

All users must be able to list all code lists and information connected to them.

4.2. Listing codes in code lists (5)

All users must be able to list all the codes and their attributes of the chosen code list.

4.3. Reorganizing the list view (2)

The users must be able to sort the order of codes of a code list according to attribute columns.

5. Updating the code lists (administrator)

5.1. Listing codes (5)

The administrator of the code register must be able to list all the code lists and their contents (also hidden code lists).

5.2. Adding attributes to code list (3)

The administrator of the code register has to able to add attributes to code lists.

5.3. Hiding code list (5)

The administrator of the system has to be able to hide a code list from the users and other systems. A hidden code list can not be users by other systems through interface.

5.4. Updating code (5)

The administration of the code register can update contents of code attributes if the other user updating the attributes makes errors or information changes.

5.5. Deleting code (3)

The administrator of the system can delete codes from code lists. This feature is meant to be used only in case of errors. The codes are not meant to be deleted in case a code is not needed any more the end date should indicate that it not in use any more. The history of changes has to be saved to system.

6. The authorization of code lists (administrator)

6.1. Listing the user roles (5)
The administrator of the code register must be able to list all user roles and their authorization information.

6.2. Adding a user role (5)
The administrator of the code register must be able to add existing user role or roles as administrators to a code list.

6.3. Deleting authorization from a user role (5)
The administrator of the code register must be able to delete rights to a code list from a user role.

6.4. Listing the systems using code register (3)
The administrator of the code register must be able to list the information of systems using a code list.

6.5. Adding systems using code register (3)
The administrator of the code register must be able add new system to use a code list.

6.6. Deleting a system using code register (3)
The administrator of the code register must be able delete a system using a code list.

7. The subsets of codes

7.1. Listing subset of codes (5)
All users of the code register must be able to list all the subsets of codes of code list if he has are authorized to use the code list.

7.2. Creating a new subset
The users authorized to update a code list can add a new subset of codes that belong to the code list.

7.3. Arranging the order of codes in subset (4)
The users authorized to update the code list are allowed to arrange the codes of a subset into a certain order and save the information of the order.

7.4. Updating a subset of codes (4)
The users authorized to update a code list can change the contents of subset.

8. Technical requirements

8.1. It has enough performance

The generation of web pages should not take more than 10 seconds. The database should be planned so that there are no delays. Especially the interface to provide information to other systems at the same time should be fast.

8.2. It is secured, reliable, recoverable and usable

The user interface must be usable. The user authorization is done in another system.

8.3. It is maintainable and easy to operate

8.4. Layered architecture according to standard of customer has to be used

8.5. It is portable

8.6. Service interface should able to be built

The database should be planned so that it is possible to implement an efficient service interface through which other systems may interact with the code register.

Appendix B: The Customer requirements in Tree diagram

Customer need (new grouping)	Degree of importance	Degree of importance	Customer needs in detail
		5	A.1 (1.1) Users have to be identified
		5	A.2 (1.1) Users have be authorized to the system. The user roles are administrator, code list manager and viewer.
		5	A.3 (1.2) Users can log out at any time
		5	A.4 (1.3) Authorization information is saved and controlled in an other system
		5	A.5 (2.1) The administrator of the system can add a new code list
		5	A.6 (6.1) The administrator of the code register must be able to list all user roles and their authorization information.
A. Users are authorized to do different tasks	5	5	A.7 (6.2) The administrator of the code register must be able to add existing user role or roles as administrators to a code list.
		5	A.8 (6.3) The administrator of the code register must be able to delete rights to a code list from a user role.
		3	A.9 (6.4) The administrator of the code register must be able to list the information of systems using a code list.
		3	A.10 (6.5) The administrator of the code register must be able add new system to use a code list.
		3	A.11 (6.6) The administrator of the code register must be able delete a system using a code list.
		3	A.12 (5.2) The administrator of the code register can add attributes to code lists.
B. The system must be adaptable	5	5	B.1 (2.2) New codelists can be created without changes in system
		5	B.2 (2.2) A new codelist can have at least 20 different attributes of different types
		5	C.1 (2.3) Users must not add codes to list before it is allowed
		5	C.2 (5.3) It must be possible to prevent use of a codelist
		5	C.3 (3.3) It must be possible to add codes so that they are not available to other systems or users.
		5	C.4 (6.1) It must be possible to allow only some users to update and add codes to a codelist
C.The data is reliable	5	5	C.5 (New) The form of the data inserted must be validated
		5	C.6 (4.1) All users must be able to see the usable code lists

		5	C.7 (4.2) All users must be able to see all codes and their attributes of usable code lists
		5	C.8 (5.1) The administrator of code register must be able to see all code lists also hidden ones.
		5	C.9 (5.4) The administrator of the code register must be able to update codes of the code lists.
		5	C.10 (7.1) All users of the code register must be able to list all the subsets of codes of code list if he has is authorized to use the code list.
D. Codes or codelists that have been used by other systems or users must not be deleted except by administrator in case of severe errors.	4	5	D.1 (3.4) After confirmation of the code the authorized user may change only the end date of the code, no other attributes.
		3	D.2 (5.5) The administrator of the system can delete codes from code lists. This feature is meant to be used only in case of errors. The codes are not meant to be deleted in case a code is not needed any more the end date should indicate that it not in use any more. The history of changes has to saved to system.
		5	E.1 (7.2) The users authorized to update a code list can add a new subset of codes that belong ta the code list.
E. It must be possible to group the codes in a codelist into subgroups	4	4	E.2 (7.3) The users authorized to update the code list are allowed to arrange the codes of a subset into a certain order and save the information of the order.
		4	E.3 (7.4) The users authorized to update a code list can change the contents of subset.
		3	F.1 (3.2) It must be possible to save several codes at a time.
		2	F.2 (4.3) The users must be able to sort the order of codes of a code list according to attribute columns.
F. It must be simple to use	3	4	F.3 (New)The user interface follows the standard of interfaces of the customer
		3	F.4 (New) All the functions allowed to users are active other are visible but nonactive
G. It must be possible to create service interface through which other systems may interact with the code register	5		G.1 (technical) The database should be planned so that it is possible to implement an efficient service interface through which other systems may interact with the code register
H. The administration of user roles must simple		5	H.1 (3.1) The users that are give the authority to manage a codelist can add new codes to the code list