LAPPEENRANTA UNIVERSITY OF TECHNOLOGY
Department of Industrial Engineering and Management

Sami Kaukavuori

Requirements Management in the Software Development Process

The subject of this thesis was approved by council of the Department of Industrial Engineering and Management in the meeting of May 3, 2000.

Supervisor:	Prof. Seppo Pitkänen
Instructor:	Petri Nissinen

Espoo, August 30, 2000

Sami Kaukavuori
Vallikallionkuja 4 D 41
02600 Espoo
Tel. +358-40-7586841

ACKNOWLEDGMENTS

This Master's Thesis was written in Nokia Information Management during the year 2000.

I would like to thank my instructor Petri Nissinen for the possibility to make this thesis and for the interesting subject. I'd also like to thank all the other people who have helped me during this project.

Special thanks to my parents for the confidence in me, and all the support during, and most of all prior to, the making of this thesis.

Espoo, August 30, 2000

Sami Kaukavuori

ABSTRACT

Author: Sami Kaukavuori **Title: Requirements Management in the Software Development Process**	
Department: Industrial Engineering and Management	
Year: 2000	**Place**: Espoo
Master's Thesis. Lappeenranta University of Technology. 107 Pages, 21 Figures, and 3 Tables Supervisor Professor Seppo Pitkänen	
Keywords: Requirements Management, Software Development, Software Engineering **Hakusanat**:Vaatimustenhallinta, ohjelmistokehitys	

Software development is a complex process, and has a lot to do with the requirements for the software product. These are several different kinds of requirements, and they are presented in various levels; from intended functionality of a certain part of the software to very detailed requirements.

Managing these requirements is also very complicated, although in literature it is presented as a simple straightforward process, which consists of several distinct phases.

The emphasis of this thesis was on how to handle changes in these requirements, other feedback after the software has been released, and how the overall process could benefit from using a requirements management tool.

Using a requirement management tool (RMT) does not solve any problems, but it gives the means to improve requirements management considerably. Some advantages of using RMT are: centralised storing of the requirements, using different kind of access rights for different users concerning access to and changing the data, structured handling of the change management process, impact and traceability analysis, and access to the data using a web browser.

TIIVISTELMÄ

Tekijä: Sami Kaukavuori

Työn nimi: **Vaatimustenhallinta ohjelmistokehityksessä**

Osasto: Tuotantotalous

Vuosi: 2000 | **Paikka**: Espoo

Diplomityö. Lappeenrannan Teknillinen Korkeakoulu.
107 sivua, 21 kuvaa ja 3 taulukkoa
Tarkastajana professori Seppo Pitkänen

Hakusanat: Vaatimustenhallinta, ohjelmistokehitys
Keywords: Requirements Management, Software Development, Software Engineering

Ohjelmistokehitys on monimutkainen prosessi. Yksi keskeisistä tekijöistä siinä on ohjelmistolle asetettavat vaatimukset. Näitä vaatimuksia on hyvin monenlaisia, ja eri tasoisia; toivotusta toiminnallisuudesta hyvinkin yksityiskohtaisiin vaatimuksiin.

Näiden vaatimusten hallinta on myöskin hyvin monitahoista, vaikkakin se on kirjallisuudessa esitetty selkeänä prosessissa, joka on sarja toisistaan erottuvia vaiheita.

Työn painopiste oli näiden vaatimusten muutoksen ja valmiiseen ohjelmistoon kohdistuvan palautteen hallinnassa, ja kuinka vaatimustenhallintaohjelmisto voisi olla avuksi näissä prosesseissa.

Vaatimustenhallintatyökalun käyttö ei sinällään ratkaise mitään ongelmia, mutta se suo puitteet parantaa vaatimusten hallitsemista. Työkalun käytöstä on muun muassa seuraavia etuja: vaatimusten keskitetty varastointi, käyttäjäoikeuksien märittely koskien eri käyttäjiä ja heidän pääsyään näkemään tai muuttamaan tietoa, muutoksenhallintaprosessin hallinta, muutosten vaikutuksen analysointi ja jäljitettävyys ja pääsy tietoihin web-selaimella.

INDEX

TABLE OF FIGURES

LIST OF TABLES

ABBREVIATIONS AND ACRONYMS

AD	Applications Development
API	Application Programming Interface
APS	Application Services
BMS	Business Unit Marketing System
	Business Unit Logistics Management System
BU	Business Unit
CAD	Computer Aided Design
CASE	Computer-Aided Software Engineering
CD	Committee Draft
CIO	Chief Information Officer
CPU	Central Processing Unit
CR	Change Request
DIS	Draft International Standard
DOORS	Dynamic Object Oriented Requirements System (product)
DPA	Delivery Process Application Services
DXL	DOORS eXtension Language
FSP	Financial Services Platform
GSC	Global Support Concept
HCI	Human-Computer Interaction
HTML	HyperText Markup Language
IEC	International Electrotechnical Commission
IM	Information Management
ISO	International Organization for Standardization
IS	Information System
IT	Information Technology
MLS	Modular Logistics System
NET	(Nokia) Networks
NMP	Nokia Mobile Phones
NSC	Nokia Sales Configurator
OODB	Object-Oriented DataBase

OPC	Operations Center
PDTR	Proposed Draft Technical Report
RAD, R&D	Research and Development
RM	Requirements Management
RMT	Request Management Tool
	Requirements Management Tool
RTF	Rich Text Format
RTM	Requirements Traceability Management
SAP	Systems, Applications and Products (company)
SBM	Service Business Management
SC	Sales Configurator
SGML	Standard Generalized Markup Language
SI	Systems Integrators
SIR	System Investigation Request
	Support/Investigate Request
SLA	Service Level Agreement
SLATE	System Level Automation Tool for Engineers (product)
SP	Service Pack
SPICE	Software Process Improvement and Capability dEtermination
SQL	Structured Query Language
WH	Warehouse
QA	Quality Assurance
QSS	Quality Systems & Software (company)

1. INTRODUCTION

1.1 Background

The problem is how to handle thousands of requirements that are coming through different channels and are constantly changing, and do not necessarily meet the users' needs.

The present requirements management process is mostly based on feedback from the developers and users. Methods for gathering feedback are: Request Management Tool (RMT), Change Request (CR) and System Investigation Request (SIR) processes, Key User Workshops, Deployment phase, and training sessions. Business Units (BUs) also have requirements concerning different configurators and they have the rules and the product information for the configurations. The role of the BUs has a more long-term effect on the process than the other ways of collecting information.

1.2 Goals and Objectives

The goal of this thesis was to find out from the literature what are in theory the characteristics of good software, and how the software engineering process and requirements management are conducted.

Then it analyses the current process of requirements management in the case company. The emphasis will be on how to manage changes to requirements and on requirements coming from the actual users of the software.

One of the goals was to evaluate different requirements management tools, i.e. software products designed to handle requirements, and find out how they could be used to improve the management of requirements.

1.3 Scope and Limitations

The goal was to examine the current requirements management process, compare it with the theoretical process presented in the literature, evaluate requirements management tools, and see if they could help to solve the problems.

1.4 Structure of Thesis

The second chapter is about software quality evaluation, what are the components of a good software product, how they are defined, and how they can be achieved. They are the reason for any requirement during the software development process.

The third chapter deals with issues concerning software engineering and design, i.e. what are the different theoretical approaches and phases in the software engineering process, and what each phase includes.

The fourth chapter is about requirements management in general, and benefits of requirements management tools.

The fifth chapter is about feedback and the importance of it in the software development.

The empirical part first presents the current situation; what are the procedures regarding feedback and requirements from users and other stakeholders for the development team, and how are different kinds of issues handled differently.

And finally there is a model for the requirements management process, the advantages of using a requirements management tool in the current situation, and how could it be used.

2. SOFTWARE QUALITY EVALUATION

There is a close analogy between different interpretations of the term usability and comparable interpretations of the term quality. Although the term quality seems self-explanatory in everyday usage, in practice there are many different views of what it means and how it should be achieved as part of a software production process. (Bevan, 1994)

2.1 Quality

Garvin (1984) distinguishes between five overall approaches to defining quality. A traditional view is that quality is transcendent: a simple unanalyzable property which is recognized through experience. Although the term quality often raises this pre-conception, it is an ideal view which does not provide any indication of how quality can be achieved in practice. Garvin distinguishes four other practical approaches to quality:

Product quality: an inherent characteristic of the product determined by the presence or absence of measurable product attributes.

Manufacturing quality: a product which conforms to specified requirements.

User perceived quality: the combination of product attributes which provide the greatest satisfaction to a specified user.

Economic quality: a product which provides performance at an acceptable price, or conformance to requirements at an acceptable cost. (Bevan, 1994)

Rather than debate which (if any) of these definitions of quality is correct, they should be recognized as distinctly different approaches, each of which has value for its own purpose. (Bevan, 1994)

Quality is generally treated as a property of a product, thus the product view of quality seeks to identify those attributes which can be designed into a product or evaluated to ensure quality. ISO 9126 takes this approach and categorises the attributes of software quality as: functionality, efficiency, usability, reliability, maintainability and portability (Figure 1). (Bevan, 1994)

functionality: the capability of the software to provide functions which meet stated and implied needs when the software is used under specified conditions.

reliability: the capability of the software to maintain its level of performance when used under specified conditions.

usability: the capability of the software to be understood, learned, used and liked by the user, when used under specified conditions.

efficiency: the capability of the software to provide the required performance, relative to the amount of resources used, under stated conditions.

maintainability: the capability of the software to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in the environment, and in requirements and functional specifications.

portability: the capability of the software to be transferred from one environment to another.

Figure 1. ISO/IEC CD 9126-1 definitions (van Veenendaal, 1997)

In order to evaluate software it is necessary to select relevant quality characteristics. This can be done using a quality model which breaks software quality down into different characteristics. ISO/IEC 9126 (1991) provides a general-purpose model which defines six broad categories of software quality: functionality, reliability, usability, efficiency, maintainability and portability.

These are further broken down into subcharacteristics which have measurable attributes (Figure 2). (van Veenendaal, 1997)

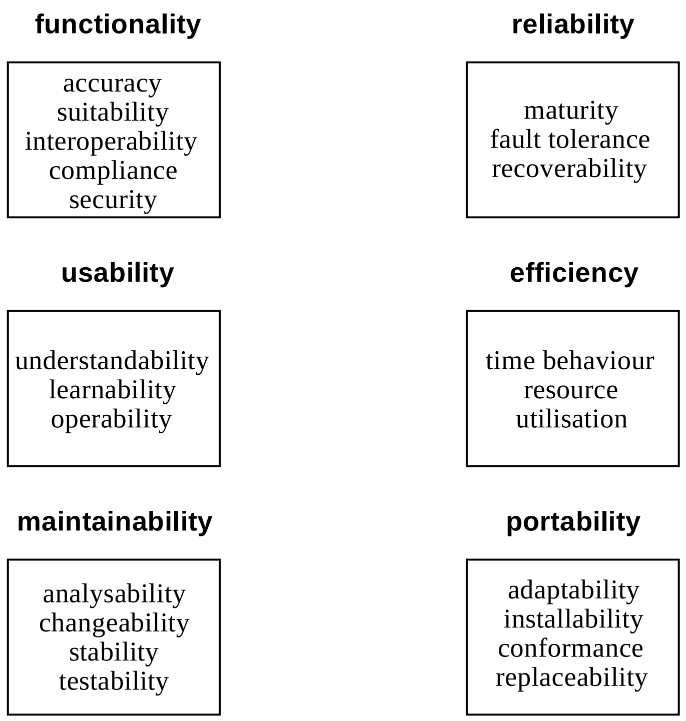

Figure 2. ISO/IEC 9126 quality model (van Veenendaal, 1997)

Another approach to quality which has been widely taken up is the use of the ISO 9000 standards to achieve manufacturing quality. (Bevan, 1994)

Economic quality is a broader approach which takes account of the need to make trade-offs between cost and product quality in the manufacturing process, or price and product quality when purchasing. (Bevan, 1994)

ISO 8402 defines quality as: *Quality*: the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs. This definition is in terms of the characteristics of a product. To the extent that user needs are well-defined and common to the intended users, it implies that quality is an inherent attribute of the product. However, if different groups of users have different needs, then they may require different characteristics for a product to have quality, so that assessment of quality becomes dependent on the perception of the user. (Bevan, 1994)

2.2 Reliability

There is no doubt that the reliability of a computer program is an important element of its overall quality. If a program repeatedly and frequently fails to perform, it matters little whether other software quality factors are acceptable. (Pressman, 1992, p. 581)

Software reliability, unlike many other quality factors, can be measured directly and estimated using historical and developmental data. Software reliability is defined in statistical terms as "the probability of a failure free operation of a computer program in a specific environment for a specific time." (Pressman, 1992, p. 581)

Whenever software reliability is discussed, a pivotal question arises: What is meant by the term "failure"? In the context of any discussion of software quality and reliability, failure is nonconformance to software requirements. Yet, even within this definition there are gradations. Failures can be merely annoying or they can be catastrophic. One failure can be corrected within seconds while another requires weeks or even months to correct. Complicating the issue even further, the correction of one failure may in fact result in the introduction of other errors that ultimately result in other failures. (Pressman, 1992, p. 581)

2.3 Efficiency

In well-engineered systems, there is a natural tendency to use critical resources efficiently. Processor cycles and memory locations are often viewed as critical resources, and the coding step is seen as the last point where microseconds or bits can be squeezed out of the software. Although efficiency is a commendable goal, three maxims should be stated before we discuss the topic further. First, efficiency is a *performance requirement* and should, therefore, be established during software requirements analysis. Software should be as efficient as required, not as efficient as is humanly possible. Second, efficiency is improved with good design. Third, code efficiency, and code simplicity go hand in hand. In general, don't sacrifice clarity, readability, or correctness for nonessential improvements in efficiency. (Pressman, 1992, p. 540)

2.4 User Perceived Quality and Quality of Use

Most approaches to software quality do not deal explicitly with user-perceived quality. User-perceived quality is regarded as an intrinsically inaccurate judgement of product quality. For instance Garvin, 1984, observes that "Perceptions of quality can be as subjective as assessments of aesthetics." (Bevan, 1994)

However, there is a more fundamental reason for being concerned with user-perceived quality. Products can only have quality in relation to their intended purpose. For instance, the quality attributes of a racing car will be very different from a family car. For conventional products this is assumed to be self-evident. For general-purpose products it creates a problem. A text editor could be used by programmers for producing code, or by secretaries for producing letters. Some of the quality attributes required will be the same, but others will be different. Even for a word processor, the functionality, usability and efficiency attributes required by a trained user may be very different from those required by an occasional user. (Bevan, 1994)

Work on usability has led to another broader and potentially important view of quality which has been outside the scope of most existing quality systems. This embraces user-perceived quality by relating quality to the needs of the user of an interactive product: *Quality of use:* the extent to which a product satisfies stated and implied needs when used under stated conditions. (Bevan, 1994)

This moves the focus of quality from the product in isolation to the particular users of the product, the tasks and the context in which it is used. The purpose of a product is to help the user achieve particular goals, which means that measures of quality of use can be defined as: *Quality of use measures:* The effectiveness, efficiency and satisfaction with which specified users can achieve specified goals in specified environments. (Bevan, 1994)

A product meets the requirements of the user if it is effective (accurate and complete), efficient in use of time and resources, and satisfying, regardless of the specific attributes it possesses. (Bevan, 1994)

Specifying requirements in terms of performance has many benefits. This is recognized in the rules for drafting ISO standards (ISO, 1992), which suggest that to provide design flexibility, standards should specify the performance required of a product rather than the technical attributes needed to achieve the performance. (Bevan, 1994)

Quality of use is a means of applying this principle to the performance which a product enables a human to achieve. (Bevan, 1994)

2.5 Software Quality of Use

The same principle can be applied to software. Software quality attributes will determine the quality of use of a software product when it is used in a particular context. Software quality attributes are the cause, quality of use the effect. Quality of use is (or at least should be) the objective, software product quality is the means of achieving it. (Bevan, 1994)

Experience has shown that it is almost impossible to accurately specify a set of internal software attributes which will ensure that the requirements for quality of use are met (i.e. that a given group of users will be able to carry out a specified set of tasks effectively, efficiently and with satisfaction). (Bevan, 1994)

2.6 Context of Use

The quality of use is determined not only by the product, but also by the context in which it is used: the particular users, tasks and environments. The quality of use (measured as effectiveness, efficiency and satisfaction) is a result of the interaction between the user and product while carrying out a task in a technical, physical, social and organisational environment (Figure 3). (Bevan, 1994)

Measures of quality of use can be used to evaluate the suitability of a product for use in a particular context. However the measures of quality of use also depend on the nature of the user, task and environment - they are a property of the whole "work system" (ISO, 1981). Measures of quality of use can thus also be used to assess the suitability of any other component of the context. For instance whether a particular user has the necessary training or skill to operate a product, which tasks a product should be used for, or whether changes in the physical environment (such as improved lighting) improve quality of use. (Bevan, 1994)

Similarly the focus of the evaluation (element to be varied) may be a complete computer system, the complete software, a specific software component, or a specific aspect of a software component. Any relevant aspect of software quality may contribute to quality of use, but for interactive software ease of use is often a crucial issue. Quality of use thus provides a means of measuring the usability of a product, and usability is defined in this way in ISO 9241-11. (Bevan, 1994)

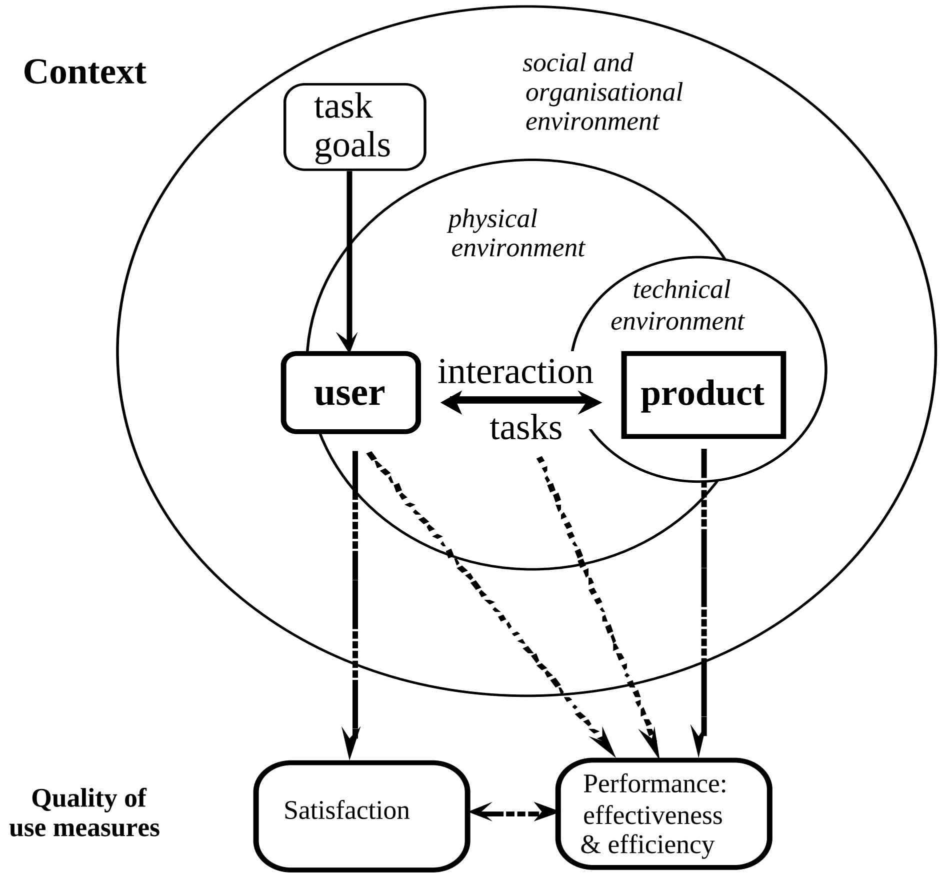

Figure 3. Quality of use measures determined by the context of use (Bevan, 1994)

The definition of quality in ISO 8402 (Quality vocabulary): *Quality*: the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs. This is a “product” oriented view of quality (Garvin, 1984): “an inherent characteristic of the product determined by the presence or absence of measurable product attributes”. In this view, the quality of a software product can be specified and built in as specific attributes of the code. The ISO/IEC 9126 definitions acknowledge that the objective of these attributes is to meet user needs in the form of functionality, reliability, usability, efficiency, maintainability and portability. But ISO 8402 makes it clear that a product-oriented view of quality should not be confused with measures of the “degree of excellence” resulting from the presence or absence of required attributes. Yet the objective of quality from the user’s perspective is to achieve a degree of excellence in a particular context of use. Despite the apparent user orientation of ISO/IEC 9126, the definitions in terms of attributes imply that software quality should be specified and measured on the basis of attributes of the source code. (van Veenendaal, 1997)

2.7 Approaches to Software Quality

The link between the ISO 9241-11 (chapter 2.8) and ISO/IEC 9126 views of usability, and “quality in use” was incorporated as a high level quality objective into the revision to ISO/IEC 9126-1, and the related ISO/IEC 14598-1 standard (Software product evaluation - General guide): *Quality in use*: the extent to which a product used by specified users meets their needs to achieve specified goals with effectiveness, productivity and satisfaction in a specified context of use. The revised ISO/IEC CD 9126-1 now distinguishes three broad approaches to improving the quality of a product (Figure 4):

Set criteria for process quality: attributes of the software development processes, e.g. by application of ISO 9001, or ISO 15504 (SPICE).

Set criteria for product quality: attributes of the software (internal measures) or the behaviour of the software when tested (external quality).

Set criteria for quality in use: the extent to which the code meets user needs for effectiveness, productivity and satisfaction in use. (van Veenendaal, 1997)

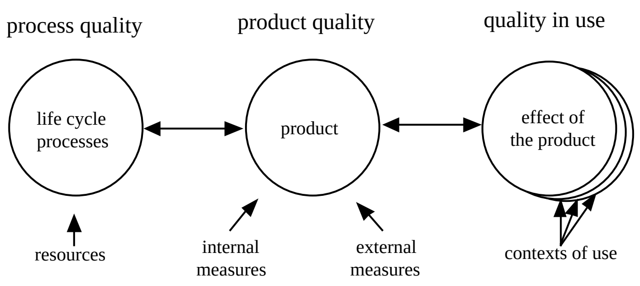

Figure 4. Approaches to software quality (van Veenendaal, 1997)

Software product quality can be measured internally (typically by static measures of the code), or externally (typically by measuring the behaviour of the code when executed). The objective is for the product to have the required effect in a particular context of use. Quality in use is the user's view of quality. Achieving quality in use is dependent on meeting criteria for external measures of the relevant quality sub-characteristics, which in turn is dependent on achieving related criteria for the associated internal measures (Figure 5). (van Veenendaal, 1997)

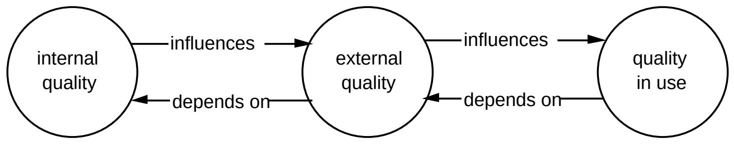

Figure 5. Relationship between different types of quality (van Veenendaal, 1997)

Measures are normally required at all three levels, as meeting criteria for internal measures is not usually sufficient to ensure achievement of criteria for external measures, and meeting criteria for external measures of sub-characteristics is not usually sufficient to ensure achieving criteria for quality in use. (van Veenendaal, 1997)

The software quality characteristics in the revision of ISO/IEC 9126 (Figure 1) have been redefined in terms of “the capability of the software”, to enable them to be interpreted as either an internal or an external perspective. The definitions also refer to “use under specified conditions” to make it clear that quality is not an absolute property, but depends on the context of use. (van Veenendaal, 1997)

2.8 Usability

In ISO 9241-11, the ISO software ergonomics committee defined usability based on the degree of excellence of a product: *usability*: the extent to which a product can be used by specific users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use. (van Veenendaal, 1997)

ISO 9241-11 explains how usability can be measured in terms of the degree of excellence in use: effectiveness (the extent to which the intended goals of use are achieved), efficiency (the resources that have to be expended to achieve the intended goals), and satisfaction (the extent to which the user finds the use of the product acceptable). ISO 9241-11 also emphasises that usability is dependent on the context of use and that the level of usability achieved will depend on the specific circumstances in which a product is used. The context of use consists of the users, tasks, equipment (hardware, software and materials), and the physical and social environments which may influence the usability of a product in a work system. Measures of user performance and satisfaction thus assess the overall work system, and, when a product is the focus of concern, these measures provide information about the usability of that product in the particular context of use provided by the rest of the work system. (van Veenendaal, 1997)

The term *usability* is sometimes used to indicate a particular approach to the issues of Human-Computer Interaction (HCI). With this in mind, the concepts that make up *usability* are considered, and a number of definitions of usability are outlined and discussed. The usability approach is concerned with both obtaining user requirements in the early stages of design, and with evaluating systems that have been built. (Booth, 1989, p. 103)

The usability perspective might be characterized as an approach that first addresses the practical issues and second theoretical issues, although some might dispute this, and argue that the two go hand-in-hand. This focus on usability does not just include information technology (IT) products, but also other types of systems, devices, machinery or work environments. This may, in part, account for why *usability* is such a commonly used term. (Booth, 1989, pp. 103-104)

It seems as though the issue of usability has grown more important as greater numbers of technically complicated products have become available to a wider population of, what Eason (1976) has termed, *naïve users*. While manufacturers have concentrated upon increasing the functionality of their products (increasing the numbers of things they can do), users have grown steadily more confused and frustrated that they cannot operate the machinery that they have bought. (Booth, 1989, p. 104)

Within the IT industry this problem has been even more serious. Many software products have had to be abandoned, not because they did not work, but because the users could not or would not use them. This may be because IT products are generally more complicated than household products such as video recorders and washing machines. (Booth, 1989, p. 104)

Usability problems appear to afflict all manner of complicated products, from complex IT systems to everyday household items. The issue of concern is how to mitigate the effects of these usability difficulties, or better still, how to ensure that usability problems never arise. This challenge is best expressed in the statement: *Today we are just as capable of producing an unusable product or system as we have always been*. In other words, the challenge is this: although we might recognize usability as a central issue in the design of complex products, how can we ensure that future products do not suffer from these problems? (Booth, 1989, p. 104)

The usability approach has been characterized as one that begins by analyzing the user's needs and setting usability goals for the intended system (or product). The idea of setting usability goals for products has been well accepted within both academia and industry. Unfortunately, the question of who sets usability goals and how they are set, has received less attention. One argument is that a system might only be as usable as its usability goals. In other words, if we choose inappropriate goals then, no matter how well we meet these goals, the system will fall short of being usable. Furthermore, the degree to which a system fails to meet usability demands may be proportionate to the gulf between the goals we set and the needs of the user. (Booth, 1989, p. 127)

2.9 Measurable Human Factors Goals

For each user and each task, precise measurable objectives guide the designer, evaluator, purchaser, or manager. These five measurable human factors are central to evaluation:

Time to learn: How long does it take for typical members of the target community to learn how to use the commands relevant to a set of tasks?

Speed of performance: How long does it take to carry out the benchmark set of tasks?

Rate of errors by user: How many and what kinds of errors are made in carrying out the benchmark set of tasks? Although time to make and correct errors might be incorporated into the speed of performance, error making is such a critical component of system usage that is deserves extensive study.

Subjective satisfaction: How much did users like using aspects of the system? This can be ascertained by interviews or written surveys that include satisfaction scales and space for free comments.

Retention over time: How well do users maintain their knowledge after an hour, a day, or a week? Retention may be closely linked to time to learn; frequency of use plays an important role. (Shneiderman, 1987, pp. 14-15)

Every designer would like to succeed in every category, but there are often forced tradeoffs. If lengthy learning is permitted, then task performance speed may be reduced by use of complex abbreviations and shortcuts. If the rate of errors is to be kept extremely low, then speed of performance may have to be sacrificed. In some applications, subjective satisfaction may be the key determinant of success, while in others short learning times or rapid performance may be paramount. Project managers and designers must be aware of the tradeoffs and make their choices explicit and public. Requirements documents and marketing brochures should make clear which goals are primary. (Shneiderman, 1987, p. 15)

2.10 Documentation

Learning anything new is a challenge. Although the challenge is usually joyous and satisfying, when it comes to learning about computer systems many people experience anxiety, frustration, and disappointment. Much of the difficulty flows directly from the poor design of the commands, menus, display formats, or prompts that lead to error conditions or simply from the inability of the user to know what to do next. (Shneiderman, 1987, p. 358)

Documentation of a software product can be critical to the success or the failure of the product (Galitz, 1984). Indeed, many software packages do fail in the marketplace because they are very poorly documented. *Documentation* here refers to users' manuals, tutorials, quick reference guides, job aids, and any other materials (hardcopy or online) that inform the user about how to access and exercise software functions and features. (Hartson, 1988, p. 145)

Even though increasing attention is being paid to improving the user interface design, there will always be a need for supplementary materials that aid the user. These materials include:

1. Traditional user manual: a paper document that describes the features of the system. Many variations on this theme include:
   a. Alphabetic listing and description of the commands
   b. Quick reference card with a concise presentation of the syntax
   c. Novice user introduction or tutorial
   d. Conversion manual that teaches the features of the current system to users who are knowledgeable about some other system.
2. Computer-based material, such as the:
   a. Online user manual – an electronic version of the traditional user manual. The simple conversion to electronic form may make the text more readily available but more difficult to read and absorb.
   b. Online help facility – the most common form of online help is the hierarchical presentation of keywords in the command language, akin to the index of a traditional manual. The user selects or types in a keyword and is presented with one or more screens of text about the command.
   c. Online tutorial – this potentially appealing and innovative approach uses the electronic medium to teach the novice user by showing simulations of the working system, by attractive animations, and by interactive sessions that engage the user.

   (Shneiderman, 1987, pp. 358-359)

Other forms of instruction or information acquisition include classroom instruction, personal training and assistance, telephone consultation, videotapes, instructional films, and audio tapes (Francas et al., 1982). (Shneiderman, 1987, p. 359)

All users of interactive computer systems require some training. Many users can learn from another person who knows the system, but training materials are often necessary. Traditional printed manuals are sometimes poorly written, but this medium can be very effective if properly prepared (Price, 1984). Many designers are enticed by the notion of online help facilities and tutorials that use the same interactive system to provide training and reminders about specific features and syntax. (Shneiderman, 1987, p. 358)

Hartson (1988) represents a list of evaluation criteria to measure documentation quality. The criteria fall into five categories, as follows:

Organisation – Thoughtful organisation greatly enhances the usefulness of software documentation. Every manual should have a table of contents, index, and tabs. Glossaries are extremely useful in defining new terms for the first time or casual user. Chapter headings and introduction and summary sections are more effective if they are presented in a task-oriented manner. For example, use 'Saving a File' rather than 'File Storage Procedures'. (Hartson, 1988, p. 145)

Typography and Legibility – Even the best documentation efforts will fall short if presented improperly to the user. Printed materials should be carefully typeset, with attention given to font style, font size, layout, and use of highlighting characteristics (italics, bold, etc.). (Hartson, 1988, p. 145)

Language – The style and level of written documentation can have an impact on how quickly and accurately information is read and understood. Readability is measured through a variety of indicators and readability formulas which focus on the number of syllables in words, the number of words in sentences, commonality of words, and so on. Whether or not sentences are in passive or active voice can also have an impact on readability and comprehension. Sentences using the passive voice can often be more difficult to understand. (Hartson, 1988, p. 145)

Graphics and Illustrations – Illustrations, half-tones, and other graphic images play a key role in documentation. They help to break up monotonous text and can, in some cases, actually provide a more effective vehicle for communicating ideas. (Hartson, 1988, pp. 145-146)

Physical Characteristics – One of the most obvious (but often overlooked) features of software documentation is its size and shape. Large, bulky documents are often perceived as uncomfortable and clumsy. Such documents can intimidate or annoy users and discourage effective use. Documentation materials should be easy to store and update, and should be made of durable materials. (Hartson, 1988, p. 146)

2.11 Summary

There are several different ways to understand the term quality. Garvin (1984) distinguishes five approaches to defining quality. A traditional view is that quality is a simple unanalysable property which is recognised through experience. Four other practical approaches are: product quality (determined by measurable attributes), manufacturing quality (product conforms to specific requirements), user perceived quality (greatest satisfaction to a specified user), and economic quality (performance at an acceptable price). They are distinctly different approaches, and each of them has value for its own purpose.

ISO 9126 categorises the attributes of software quality as: functionality, reliability, usability, efficiency, maintainability, and portability. These are attributes that can be designed into a product or evaluated to ensure quality. In order to evaluate software quality, these categories can be broken down into subcharacteristics which have measurable attributes (e.g. fault tolerance, stability, installability).

ISO 8042 defines *quality* as: the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs. If different groups have different needs, they may require different characteristics for a product to have quality, so that assessment of quality becomes dependent on the perception of the user. For example a racing car and a family car have different kinds of quality characteristics. Bevan (1994) defines *quality of use* as: the extent to which a product satisfies stated and implied needs when used under stated conditions (measured as effectiveness, efficiency, and satisfaction); and c*ontext of use:* quality of use is not determined only by the product, but also by the context in which it is used: the particular users, tasks and environments, thus dependent for example on the user's training and skills.

ISO 9241-11 defines *usability*: the extent to which a product can be used by specific users to achieve specific goals with effectiveness, efficiency and satisfaction in a specific context of use.

Software products have been abandoned, not because they did not work, but because users could not or would not use them. Usability is very important, the question is how can it be ensured. Shneiderman (1987) describes *measurable human factors*: time to learn, speed of performance, rate of errors by user, subjective satisfaction, retention over time. The goal is to succeed in all of these factors, but naturally there are forced trade-offs for example between time to learn and speed of performance.

When learning to use a new product, documentation is also very important, it includes: users' manuals, tutorials, quick reference guides, job aids.

3. SOFTWARE ENGINEERING

3.1 Computed-Based Systems

The elements of a computer-based system (depicted in Figure 6) often include the following:

Software: Computer programs, data structures, and related documentation that serve to effect the logical method, procedure or control that is required.

Hardware: Electronic devices (e.g. CPU, memory) that provide computing capability, and electromechanical devices (e.g. sensors, motors, pumps) that provide external work functions.

People: Users and operators of software and hardware.

Database: A large, organized collection of information that is accessed via software and is an integral part of system function.

Documentation: Manuals, forms, and other descriptive information that portrays the use and/or operation of the system.

Procedures: The steps that define the specific use of each system element or the procedural context in which the system resides. (Pressman, 1992, p. 132)

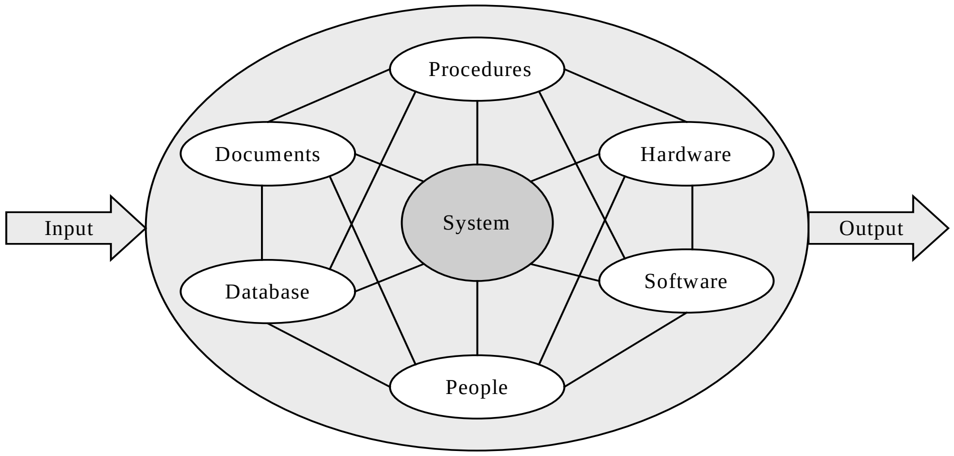

Figure 6. System elements (Pressman, 1988, p. 133)

3.2 Computer Systems Engineering

Computer system engineering is a problem-solving activity. Desired system functions are uncovered, analyzed, and allocated to individual system elements. The computer system engineer begins with customer-defined goals and constraints, and derives a representation of function, performance, interfaces, design constraints, and information structure that can be allocated to each of the generic system elements. (Pressman, 1992, p. 134)

The genesis of most new systems begins with a rather nebulous concept of desired function. Therefore, the system engineer must bound the system by identifying the scope of function and performance that are desired. The questions focus on function, performance, and information flow and content. The system engineer does not ask the customer *how* the task is to be done; rather, the engineer asks *what* is required. (Pressman, 1992, pp. 134-135)

The following trade-off criteria govern the selection of a system configuration based on a specific allocation of function and performance to generic system elements:

Project considerations. Can the configuration be built within preestablished cost and schedule bounds? What is the risk associated with cost and schedule estimates?

Business considerations. Does the configuration represent the most profitable solution? Can it be marketed successfully? Will ultimate pay-off justify development risk?

Technical analysis. Does the technology exist to develop all elements of the system? Are function and performance assured? Can the configuration be adequately maintained? Do technical resources exist? What is the risk associated with the technology?

Manufacturing evaluation. Are manufacturing facilities and equipment available? Is there a shortage of necessary components? Can quality assurance be adequately performed?

Human issues. Are trained personnel available for development and manufacture? Do political problems exist? Does the customer understand what the system is to accomplish?

Environmental interfaces. Does the proposed configuration properly interface with the system's external environment? Are machine-to-machine and human-to-machine communication handled in an intelligent manner?

Legal considerations. Does this configuration introduce undue liability risk? Can proprietary aspects be adequately protected? Is there potential infringement? (Pressman, 1992, pp. 136-137)

3.3 Phases in Software Engineering

Computer programs, the software that is becoming an ever-larger part of the computer system, are growing more and more complicated, requiring teams of programmers and years of effort to develop. As a consequence, a new subdiscipline, software engineering, has arisen. The development of a large piece of software is perceived as an engineering task, to be approached with the same care as the construction of a skyscraper, for example, and with the same attention to cost, reliability, and maintainability of the final product. The software-engineering process is usually described as consisting of several phases, variously defined but in general consisting of: (1) identification and analysis of user requirements, (2) development of system specifications (both hardware and software), (3) software design (perhaps at several successively more detailed levels), (4) implementation (actual coding), (5) testing, and (6) maintenance. (Britannica)

Function and performance are allocated to software during system engineering. In some cases, function is simply the implementation of a sequential procedure for data manipulation. Performance is not explicitly defined. In other cases, function is the internal coordination and control of other concurrent programs, and performance is defined explicitly in terms of response and wait times. (Pressman, 1992, p. 140)

To accommodate function and performance, the software engineer must build or acquire a set of software components. Unlike hardware, software components are rarely standardized. In most cases, the software engineer creates custom components to meet the allocated requirements for the software element of the system that is to be developed. (Pressman, 1992, p. 140)

The software element of a computer-based system is comprised of programs, data, and documentation that is categorized as *application software* and *system software*. Application software implements the procedure that is required to accommodate information processing functions. System software implements control functions that enable application software to interface with other system elements. (Pressman, 1992, p. 140)

Figures 7, 8, and 9 illustrate the generic steps in the software engineering process. The figures illustrate the steps that must be accomplished and the various representations of software that are derived as it evolves from concept to realization. (Pressman, 1992, p. 141)

3.3.1 Definition Phase

The definition phase of software engineering, depicted in Figure 7, begins with the software planning step. During this step a bounded description of the scope of software effort is developed; risk analysis is conducted; resources required to develop the software are predicted; cost and schedule estimates are established. The purpose of the software project planning step is to provide a preliminary indication of project viability in relationship to cost and schedule constraints that may have already been established. A S*oftware Project Plan* is produced and reviewed by project management. (Pressman, 1992, p. 141)

The next step in the definition phase is software requirements analysis and definition. During this step, the system element allocated to the software is defined in detail. Requirements are analyzed and defined in one or two ways. Formal information domain analysis may be used to establish models of information flow and structure. These models are then expanded to become a software specification. Alternatively, a prototype of the software is built and evaluated by the customer in an attempt to solidify requirements. Performance requirements or resource limitations are translated into software design characteristics. Global analysis of the software element defines validation criteria that will serve as the basis for test planning and will be used to demonstrate that requirements have been met. (Pressman, 1992, p. 143)

Software requirements analysis and definition is a joint effort conducted by the software developer and the customer. A S*oftware Requirements Specification* is the deliverable document produced as a result of this step. (Pressman, 1992, p. 143)

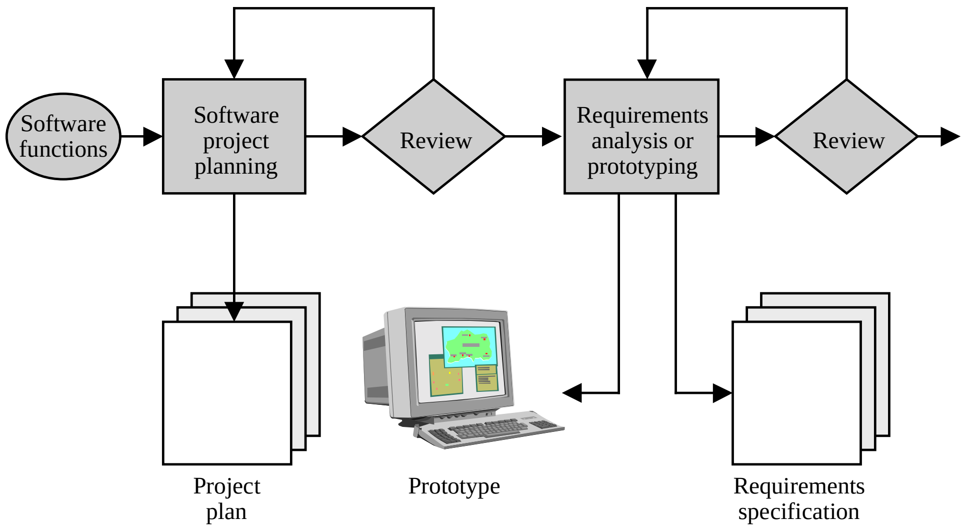
Software functions
Software project planning
Review
Requirements analysis or prototyping
Review
Project plan
Prototype
Requirements specification

Figure 7. Software engineering – definition phase (Pressman, 1992, p. 142)

3.3.2 Development Phase

The development phase (Figure 8) translates a set of requirements into an operational system element that we call *software*. The first step of the development concentrates on design. The design process for software begins with a description of architectural and data design. That is, a modular structure is developed, interfaces are defined, and a data structure is established. Design criteria are used to assess quality. This preliminary design step is reviewed for completeness and traceability to software requirements. A first-draft *Design Specification* is delivered and becomes a part of the software configuration. (Pressman, 1992, p. 143)

Procedural aspects of each modular component of the software design are considered next. Each detailed procedural description is added to the *Design Specification* after review. (Pressman, 1992, p. 143)

Coding occurs after design is complete. Software engineering methodology views coding as a consequence of good design. Code is reviewed for style and clarity, but should otherwise be directly traceable to a detailed design description. A source language for each modular component of software is the configuration deliverable for the coding step. (Pressman, 1992, pp. 143-144)

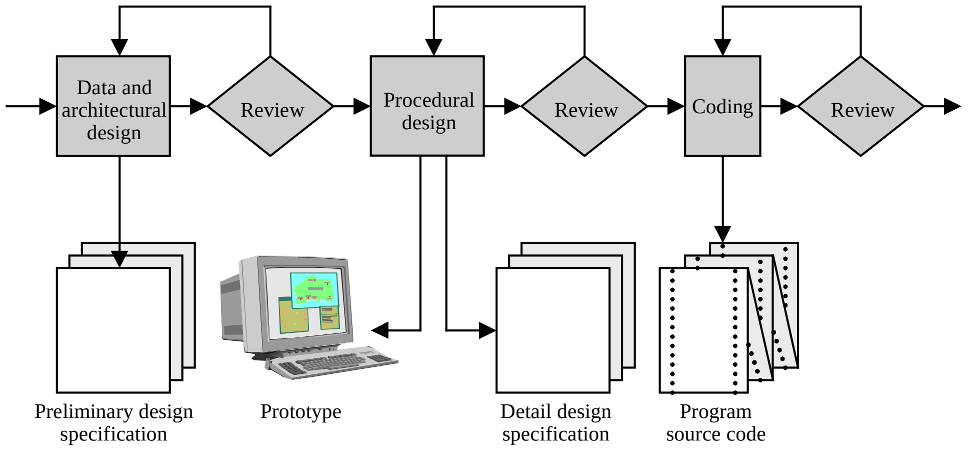
Data and architectural design
Review
Procedural design
Review
Coding
Review
Preliminary design specification
Prototype
Detail design specification
Program source code

Figure 8. Software engineering – development phase (Pressman, 1992, p. 142)

3.3.3 Verification, Release, and Maintenance Phase

During the last phase in the software engineering process (Figure 9), the software engineer tests the software to find the maximum number of errors before shipment, prepares the software for release, and then maintains the software throughout its useful life. (Pressman, 1992, p. 144)

After the source code has been generated, a series of verification and validation activities are conducted. Unit testing attempts to verify the functional performance of individual modular component of software. Integration testing provides a means for the construction of the software architecture, while at the same time testing function and interfaces. Validation testing verifies that all requirements have been met. After each of these testing steps, debugging – the diagnosis and correction of defects – may occur. A *Test Plan and Procedure* may be developed for the testing steps. A review of test documentation, test cases, and results is always conducted. (Pressman, 1992, p. 144)

Once software testing is completed, the software is almost ready for release to end users. However, before release occurs, a series of quality assurance (QA) activities are conducted to ensure that appropriate records and internal documents have been generated and cataloged, high-quality user documentation has been developed, and appropriate configuration control mechanisms have been established. The software is then distributed to end users. (Pressman, 1992, p. 144)

As soon as software is released to end users, the software engineer's job changes. Now, the focus changes from construction to maintenance – error correction, environmental adaption, and function enhancement. Recognition of this fact is the first step toward lessening the impact of a task that devours 50 to 70 percent of budget for many large software organizations. The tasks associated with software maintenance depend upon the type of maintenance to be performed. Modification of the software includes the entire configuration (i.e., all programs, data, and documents developed in the definition and development phases), not just the code. (Pressman, 1992, p. 144)

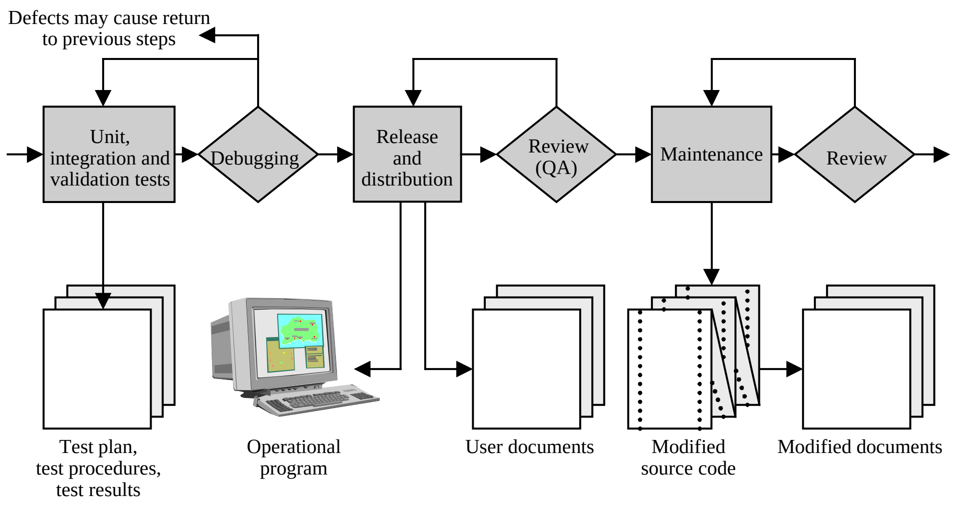

Figure 9. Software engineering – verification, release, and maintenance phase (Pressman, 1992, p. 142)

3.4 Different Approaches to Software Engineering

There are many different approaches to software engineering in the literature. This chapter introduces some of them. Even though the models are in some senses very different, the phases included are somewhat the same.

3.4.1 The Classic Life Cycle

Figure 10 illustrates the classic life-cycle paradigm for software engineering. Sometimes called the "waterfall model", the life-cycle paradigm demands a systematic, sequential approach to software development that begins at the system level and proceeds through analysis, design, coding, testing, and maintenance. (Pressman, 1992, pp. 24-25)

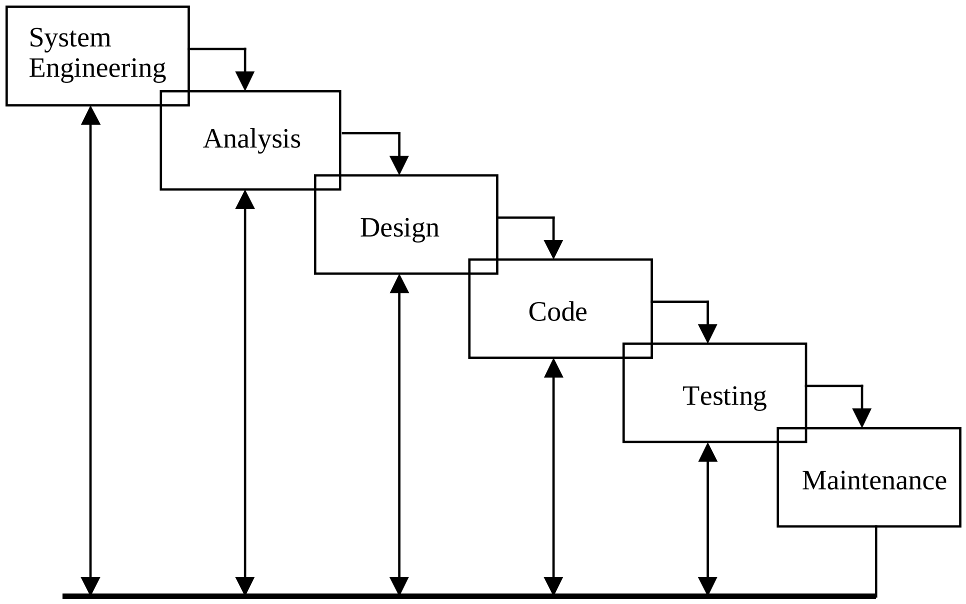

Figure 10. The classic life-cycle (Pressman, 1992, p. 25)

3.4.2 Prototyping

Often, a customer has defined a set of general objectives for software, but has not identified detailed input, processing, or output requirements. In other cases, the developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system, or the form that human-machine interaction should take. In these, and many other situations, a prototyping approach to software engineering may offer the best approach. (Pressman, 1992, pp. 26-27)

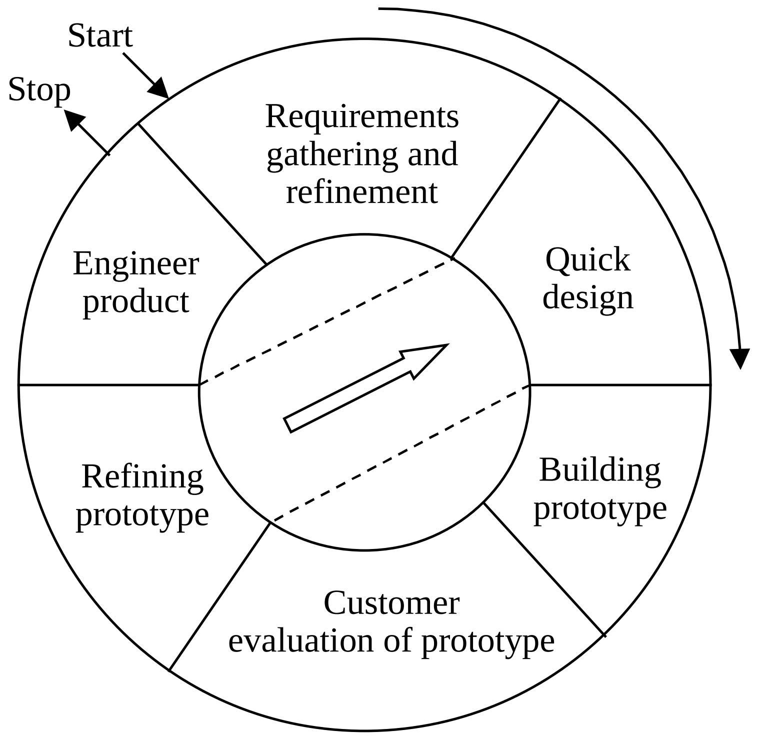

Figure 11. Prototyping (Pressman, 1992, p. 27)

3.4.3 The Spiral Model

The spiral model for software engineering (Boehm, 1988) has been developed to encompass the best features of both the classical life cycle and prototyping, while at the same time adding a new element – risk analysis – that is missing in these paradigms. The model represented by the spiral in Figure 12, defines four major activities represented by the four quadrants of the figure:

1. *Planning* – determination of objectives, alternatives and constraints
2. *Risk analysis* – analysis of alternatives and identification/resolution of risks
3. *Engineering* – development of the "next-level" product
4. *Customer evaluation* – assessment of the results of engineering (Pressman, 1992, p. 29)

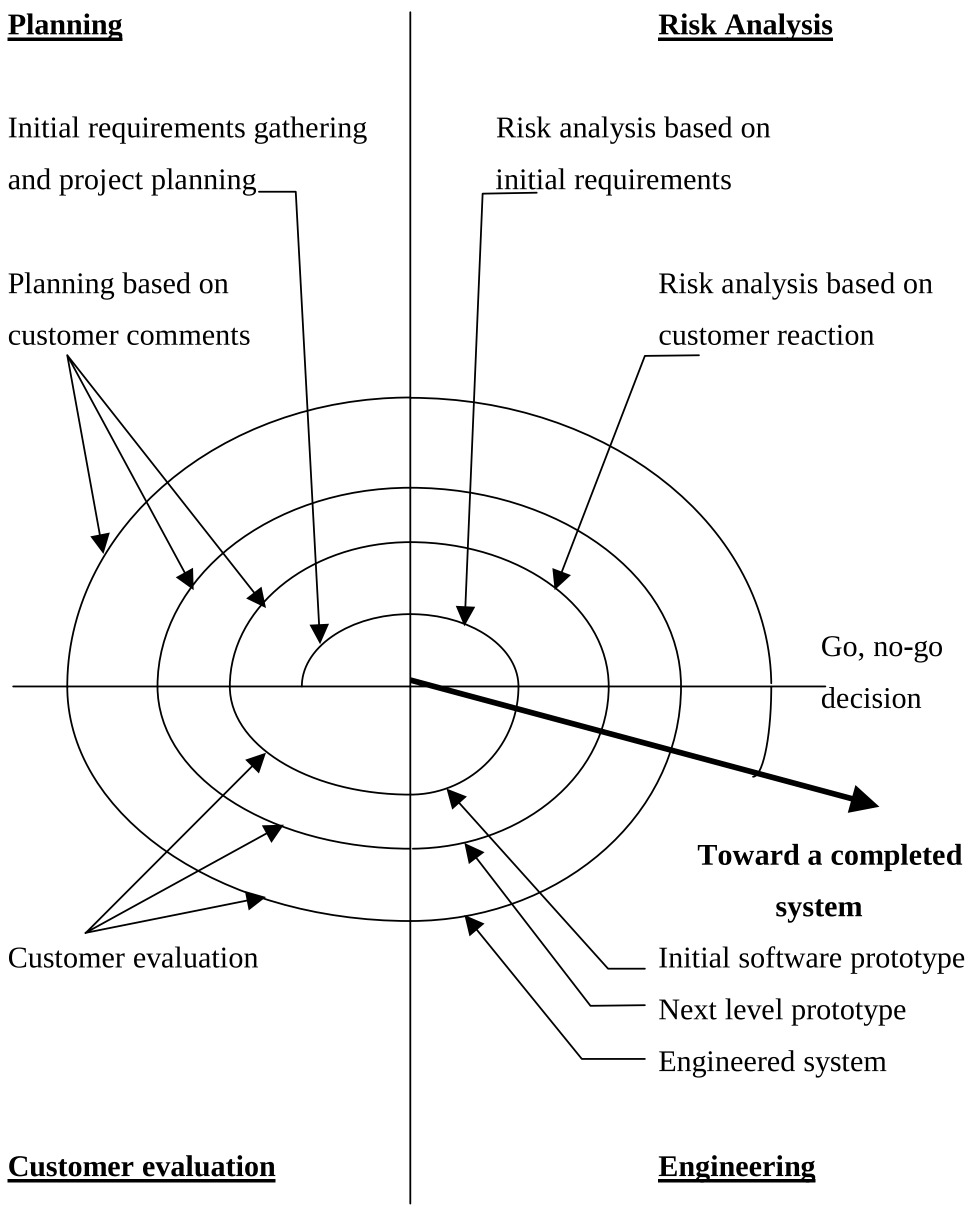
Planning
Risk Analysis
Initial requirements gathering and project planning
Risk analysis based on initial requirements
Planning based on customer comments
Risk analysis based on customer reaction
Go, no-go decision
Toward a completed system
Customer evaluation
Initial software prototype
Next level prototype
Engineered system
Customer evaluation
Engineering

Figure 12. The spiral model (Pressman, 1992, p. 29)

An intriguing aspect of the spiral model becomes apparent when we consider the radial dimensions depicted in Figure 12. With each iteration around the spiral (beginning at the center and working outward), progressively more complete versions of the software are built. During the first circuit around the spiral, objectives, alternatives and constraints are defined and risks are identified and analyzed. If risk analysis indicates that there is uncertainty in requirements, prototyping may be used in the engineering quadrant to assist both the developer and the customer. Simulations and other models may be used to further define the problem and refine requirements. (Pressman, 1992, pp. 29-30)

The customer evaluates the engineering work (the customer evaluation quadrant) and makes suggestions for modifications. Based on customer input, the next phase of planning and risk analysis occur. At each loop around the spiral, the culmination of risk analysis results in a "go, no-go" decision. If risks are too great, the project can be terminated. (Pressman, 1992, p. 30)

In most cases, however, flow around a spiral path continues, with each path moving the developers outward toward a more complete model of the system, and, ultimately, to the operational system itself. Every circuit around the spiral requires engineering (lower right quadrant) that can be accomplished using either the classical life-cycle or prototyping approaches. It should be noted that the number of development activities occurring in the lower right quadrant increases as activities move further from the center of the spiral. (Pressman, 1992, p. 30)

The spiral model paradigm for software engineering is currently the most realistic approach to the development for large scale systems and software. It uses an "evolutionary" approach (Gilb, 1988) to software engineering, enabling the developer and the customer to understand and react to risks at each evolutionary level. It uses prototyping as a risk reduction mechanism, but, more importantly, 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. (Pressman, 1992, p. 30)

But like other paradigms, the spiral model is not a panacea. It may be difficult to convince large customers (particularly in contract situations) that the evolutionary approach is controllable. It demands considerable risk assessment expertise, and relies on this expertise for success. If a major risk is not discovered, problems will undoubtedly occur. (Pressman, 1992, p. 30)

Muench, et al. describe yet another spiral model for software development with four cycles and quadrants, as illustrated in Figure 13. The idea in this spiral model is quite similar to that in Pressman's model:

- Proof-of-concept cycle – capture business requirements, define goals for proof-of-concept, produce conceptual system design, design and construct the proof-of-concept, produce acceptance test plans, conduct risk analysis and make recommendations.
- First build cycle – derive system requirements, define goals for first build, produce logical system design, design and construct the first build, produce system test plans, evaluate the first build and make recommendations.
- Second build cycle – derive subsystem requirements, define goals for second build, produce physical design, construct the second build, produce system test plans, evaluate the second build and make recommendations.
- Final cycle – complete unit requirements, final design, construct final build, perform unit, subsystem, system, and acceptance tests.

(Duncan, 1996, p. 15)

-

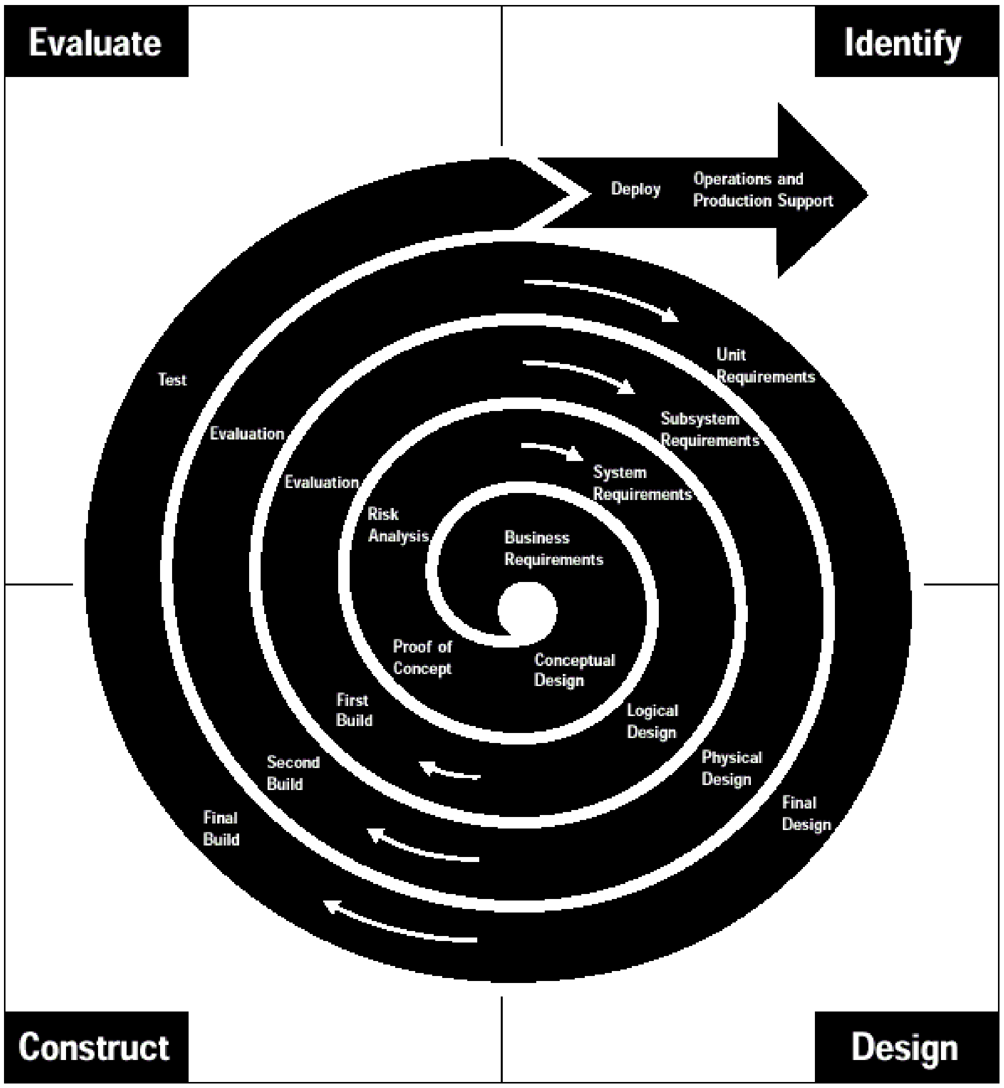

Figure 13. Representative software development life cycle (Muench, 1994)

3.5 Summary

The software-engineering process is usually described as consisting of several phases, variously defined but in general consisting of: (1) identification and analysis of user requirements, (2) development of system specifications (both hardware and software), (3) software design (perhaps at several successively more detailed levels), (4) implementation (actual coding), (5) testing, and (6) maintenance. (Britannica)

There are three different kinds of approaches to how to combine these phases.

The classic life-cycle paradigm, also called the "waterfall model", demands a systematic, sequential approach to software development that begins at the system level and proceeds through analysis, design, coding, testing, and maintenance in a straightforward manner (illustrated in Figure 10).

Often, a customer has defined a set of general objectives for software, but has not identified detailed input, processing, or output requirements. In other cases, the developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system, or the form that human-machine interaction should take. In these, and many other situations, a prototyping approach to software engineering may offer the best approach. (Figure 11) (Pressman, 1992, pp. 26-27)

The spiral model for software engineering (Boehm, 1988) has been developed to encompass the best features of both the classical life cycle and prototyping, while at the same time adding a new element – risk analysis – that is missing in these paradigms. The model represented by the spiral in Figure 12, defines four major activities represented by the four quadrants of the figure: planning, risk analysis, engineering, and customer evaluation (Pressman, 1992, p. 29)

4. REQUIREMENTS MANAGEMENT

According to Stokes, requirements are: "Collection of statements that describe in a clear, consistent and unambiguous manner all aspects of a proposed system". (McDermid, 1991)

4.1 Requirements Management Tools

Static requirements documents are not much help in evaluating the impact of suggested changes, or in ensuring thorough testing and documentation. Requirements management tools and improved practices can help. (Light, 1998)

What project management best practices will assist Applications Development (AD) organizations in maximizing return on investment for their AD projects while reducing the potential for cost overruns, late delivery and scope creep? (Light, 1998)

AD organizations are increasingly finding that simply gathering static, text-based requirements without automated support is of little use when changes are suggested. Similarly, software requirements as output from a business process-modeling tool are seldom static reference items. A static, nonautomated approach to requirements fails to streamline development teams' analysis of requirements, and unnecessarily increases the future burden of enhancing and maintaining systems - a burden that cripples the responsiveness of many AD groups to new development demands. (Light, 1998)

Therefore, requirements generation - whether in documents or process models - should not be viewed as just a step in development that, once completed, feeds the next step. Rather, it should be part of ongoing requirements management - a process much simplified if requirements are captured in a database-based tool to enable collaborative review for completeness, use-case creation, test-case creation, traceability, and to facilitate versioning/change control. (Light, 1998)

Since the introduction of tools from Quality Systems Software (DOORS) and TD Technologies (SLATE) in the early 1990s, the RM tool market - once limited to Unix workstation tools used by technical engineers on highly complex aerospace, defense or manufacturing systems – has more than doubled to about $60 million for 1998. New tools for the Windows platform have appeared in recent years. Approaches increasingly interfacing with modeling tools also promise to further stimulate the market. Here, we give an overview of the leading vendors, profile the market, and identify key criteria in selecting an RM tool. Chapter 4.1.1 provides short profiles of the leading high-end requirements tools used most often in building very large, complex systems. Chapter 4.1.2 profiles other leading requirements tools used on projects of varying complexity. (Light, 1998)

Windows-based tools feature relative ease of setup and use, and target the low end of the market, i.e. small to midsize organizations (e.g., of fewer than 250 developers), whose project durations are typically less than 18 months. (Light, 1998)

Along with RTM and SLATE, market share leaders include DOORS and RequisitePro. Quality Systems Software with DOORS and Integrated Chipware with RTM together account for most of the overall market, and about 80 percent of the Unix market. They compete mainly with TD Technologies' SLATE. However, in the growing Windows segment, the relative newcomer Rational Software, with RequisitePro, emerged as a close second to DOORS in 1997. Another newcomer, Technology Builders, entered the market in 1998 with its Windows-based Caliber-RM tool that, like SLATE, uses the Versant OODB. RTM's weak Windows implementation (improved in RTM Workshop) has achieved only scant market penetration, less than 5 percent. (Light, 1998)

Bottom Line: How requirements are initially gathered and stored often reveals the level of an IS organization's engineering discipline. Those that provide teams with an automated requirements environment will better support change control efforts, gain testing efficiencies, and potentially reduce their future maintenance burden. (Light, 1998)

4.1.1 High-End Requirements Management Tool Profiles

RTM Workshop: Integrated Chipware's RTM (Requirements Traceability Management) utilizes an Oracle database integrated with the tool and features significant in structured analysis capability. Largely focused on the embedded systems market for RM tools, Integrated Chipware has particularly targeted Unix-based development at high-end manufacturing, aerospace and defense firms that develop and produce such products. It mainly targets government contracts requiring high levels of documentation and requirements traceability. RTM Workshop requires substantial training and, often, significant ongoing support. (Light, 1998)

SLATE: TD Technologies' System Level Automation Tool for Engineers (SLATE) was first developed by Texas Instruments, which sold it to TD Technologies in 1994. Mainly used on large defense projects, SLATE was made for use by distributed, concurrent-engineering design teams for which requirements begin mainly as system design elements, not text; the tool enables document generation as a byproduct of the design capture process, and features Internet access to documents published on the Web. Formerly Unix-only, Windows NT support was added in 1997. (Light, 1998)

High-end tools are also available from Ascent Logic (RDD-100, for Requirements Driven Development), Compliance Automation (Vital Link) and Teledyne Brown Engineering (XTie-RT, for Requirements Tracer). (Light, 1998)

4.1.2 General-Purpose Requirements Management Tools

DOORS: Quality Systems Software's Dynamic Object-Oriented Requirements System (DOORS) ships with a proprietary object-oriented database, enabling broader support of large design drawings and nontext requirements, and quicker requirements-oriented queries. DOORS is also easier to learn and use, and has a strong, market-leading Windows implementation. QSS is increasingly targeting the commercial IS market, as also indicated by its marketing alliances with Microfocus/Intersolv and Platinum Technology, and DOORS' recent integration with Rational Rose. (Light, 1998)

RequisitePro: Requisite, founded in 1996, was acquired by Rational Software last year. From its inception, RequisitePro has targeted the IS/AD market more than the product development or systems engineering arenas. RequisitePro features interfaces with Rose and other Rational tools for software testing, configuration management and documentation (SQA Suite, ClearCase, SoDA), and with Microsoft's Visual SourceSafe, Word and Project 98. Thus users can test requirements by tracing them to test procedures, thereby improving test coverage, and trace requirements to associated software code, as well as partially automate documentation. (Light, 1998)

Caliber RM: Technology Builders designed its new tool with a three-tier architecture featuring both a Windows-based and a thin Web-browser client. Distributed groups can alter or comment on individual requirements without locking a project's entire requirements document and, by storing user responsibility information with a requirement, the tool enables Caliber's automatic notification of changes to affected team members. The tool features interfaces with Mercury Interactive's Test Director and Select Software Tools' Select Enterprise. (Light, 1998)

4.2 Taming Scope Scourge

AD groups often lack project plans, activities and defined deliverables that are consistent with a project's shifting requirements. Effective requirements management takes both traceability and fully authorized project managers. (Light, Conway, 1997)

What strategies, processes and techniques will assist AD organizations in reducing their exposure to project failures? (Light, Conway, 1997)

Despite widespread awareness of the dangers of scope creep in AD, inquiries to Gartner Group show that many IS organizations still suffer from this dreaded scourge. Unfortunately, the conviction to avoid the problem seldom yields actual project management mechanisms, so that changes and additions often drive projects over budget or beyond their due dates. (Light, Conway, 1997)

Most AD groups have some type of initial agreement with the application's intended users as to specified intended functionality. However, these software requirements typically "creep" upward by about 1 percent a month, according to Capers Jones ("Assessment and Control of Software Risks," 2nd edition, 1996). Requirement changes are generally duly documented and allocated among hardware, software and other system components, but their effect on project scope is seldom well-established, so that project plans, activities and defined deliverables stray from the initial budget and schedule. (Light, Conway, 1997)

Requirements management extends the AD group's initial review of the requirement allocations, to maintain the initial agreement throughout development, along with realistic budgets and schedules. To understand the impact of changes, planners must be able to trace the effects of those changes on the rest of the system and the project overall. Some large Systems Integrators' (SIs') AD methodologies closely track changes to maintain the accuracy of the project estimates that are the basis of their bids. Rigorous traceability is also practiced by U.S. Department of Defense and healthcare industry contractors, and by makers of high-reliability systems and manufactured goods, to ensure that user needs are fully met and that no unintended system behavior occurs. Each documented user requirement must be traced to a software function; and each software function must be traceable to a user requirement, or it will yield unspecified behavior. Traceability links requirement attributes to show that user needs are met and that the system will not work in unexpected ways. (Light, Conway, 1997)

Examples of Requirement Attributes

- Reason for requirement
- Resource to meet requirement
- Time created or updated
- Version number, as requirement is revised
- Status (e.g., proposed, in progress or tested)
- Parent and child requirements, dependencies, including links to non-IS projects
- Priority of requirement (in function, budget or schedule)
- Owner, person or team to work on requirement

(Light, Conway, 1997)

Specifically, we advise that the project manager - as the person responsible for bringing the project in on time and on budget - should be fully and formally authorized to negotiate regarding requirement changes that would affect the project's resources, budget and schedule. Changes rejected by the project manager should not be subject to appeal to a vice president or CIO unless seconded by an executive-level sponsor. (Light, Conway, 1997)

Estimating project costs and delivery due dates depends critically on requirements management, the key to which is traceability. IS organizations should train project managers in scope management and empower them to negotiate requirements changes, especially any that could lead to project delays, cost overruns or cancellations. (Light, Conway, 1997)

5. FEEDBACK

5.1 Positive Feedback

Information System (IS) organizations often receive negative feedback and are faulted for even minor service gaps. With simple steps, they can reverse this tendency, creating positive feedback and gaining support for their critical role. (Gabler, 1999)

Service-oriented IS organizations seeking to encourage positive user feedback should:

- Set the foundation for service visibility
- Set the context by reporting the magnitude of services provided
- Be honest about services provided and not provided
- Solicit feedback from users to ensure synchronized perceptions

(Gabler, 1999)

Historically, Information Technology (IT) system implementations were the focal point of user and IS congratulations as new capabilities went live - IS staff were heroes. As IT matured, users and management began to realize that IS staff also provide service and maintain unique tools that serve the business needs. This necessary shift from one-time implementation goals to ongoing service goals requires a mind-set change. Now, IS staff are often faulted for "creating" problems whenever there are service gaps. Without positive reinforcement, any task can become laborious and demotivating, and IS staff, typically highly motivated, will lose interest, causing service to degenerate. Since the reason for investing in IT is to address a business need, IS's ongoing service role has become vital to the business team. The next chapter addresses the first step toward attaining consistent, quality IS service - communicating the volume and status of service requests through service activity reports. (Gabler, 1999)

5.1.1 Service Activity vs. SLA Reports

Service activity reports align users perceptions with the IS organization's reality, setting a common context for service volume and status (how big the job is). Although activity reporting is *not* performance-level reporting, it does expose IS to the organization. Exposure can induce improved service, but only if IS management understands that its value can only be based on business effectiveness. Once users and IS understand service volume and status, IS organizations must mature to SLA reporting.

SLA reports align the IS organization's perceptions with users' reality, comparing the level of service delivered with the level of service expected (how well the job is done). For example, SLA reports could show, for a specific time frame, the average duration of an open service request and the average time from request to initial response compared with agreed-upon thresholds. Thus, SLAs are likely to include activity volumes but probably not the status of specific service requests.

Service activity and SLA reports are complementary. Both parties (IS organization and users) have different perspectives and objectives. Assessing performance levels without an appreciation for volume is impractical. Volume and available resources are the primary factors in determining performance in a well-run service organization. Establishing a common volume/status context is the first step toward establishing a common performance-level context. (Gabler, 1999)

Problem: "Good" service goes unnoticed because it is expected. Thus, service gaps are highly visible, creating a "negative feedback loop." For example, the telephone is hardly noticed until it does not work as expected, at which time it becomes a major obstacle. Why should IS service be viewed differently? The problem stems from perception, since we all tend to remember negative events more readily than positive events. Solution: The IS organization must track service issues and problems, then establish accurate perceptions of the resulting service activities based on four principles:

Set the Foundation. Set up regular, recurring service activity reports to be issued at least monthly. Users soon begin to expect these reports, establishing a foundation for presenting a service message.

Set the Context. The number of service requests received and handled is often much greater than users and their managers realize. Just publishing these counts creates a significantly different perception of service gaps. Managers often admit that they had no idea their staff were generating so many requests. This often results in training or procedural changes, which decrease some service activities. Reporting the magnitude of service activities establishes the context.

Be Honest. Including the number of open requests underscores the IS organization's honesty in publicizing the service load. The number of requests opened minus the number of requests closed quantifies both work in progress (open) and work completed (closed). Both numbers create positive perceptions that otherwise could be negative. Honesty establishes credibility.

Solicit Feedback. Detail closed service requests and make it clear that, if the user views a request otherwise, the IS organization wants to know. Closing a request that a user feels has not been resolved erodes credibility. Synchronizing perceptions establishes further credibility and a sense of teamwork and honesty between both parties. (Gabler, 1999)

For example: The IS organization should track and monitor all calls to a help desk. It should generate monthly reports by function unit, showing the total requests open at the end of the last reporting period; total requests received for the current time period; total requests closed in the current time period; and total requests still open at the end of the current time period. This summary dispels most negative perceptions and focuses user attention on the amount of service actually delivered. A second part of the report should detail each closed service request so the user knows the status. The report should invite the user to notify IS of any discrepancies. Publication of this report allows both parties to know the exact status of the service load and IT problems, diffusing misunderstandings and providing a healthy communication vehicle. (Gabler, 1999)

The IS organization's service role is critical for users to use IT systems smoothly, but if accurate perceptions are not created by IS, users will develop negative perceptions based on personal experience. Through simple reports that quantify requests and their resolution, the IS organization can begin to step into its vital role as part of the business team. SLA reporting can then be added as the next maturation step. (Gabler, 1999)

5.2 User Satisfaction Monitoring

Measurement of user satisfaction with IT-delivered products or services provides an opportunity to focus on the feedback and develop action plans that yield the greatest improvement in user satisfaction level. (Redman, 1998)

Unfortunately, many IS organizations do not really know how satisfied their users are with the services being provided. Moreover, many IS organizations do not have a good understanding of what their users' top IT concerns are. How can IS organizations address such important issues and significantly improve user satisfaction within their resource constraints? (Redman, 1998)

Traditionally, IS performance metrics have been based on efficiency, technology and budgetary guidelines. Although such measurements are important, measuring internal customer satisfaction is emerging as a significant opportunity as well as a business requirement. The concept of user feedback may appear trivial, but collective user perception can be so powerful that it can make or break the credibility and future success of the IT services provider and its management. (Redman, 1998)

5.2.1 Focused Feedback

With respect to time and money, it is generally prohibitive to embark on fixing everything that is perceived to be "wrong" with the IS organization and the services it delivers. However, measurement of internal customer satisfaction with IT-delivered products or services provides an opportunity to focus on the feedback and develop action plans that will yield the greatest improvement in user satisfaction levels. (Redman, 1998)

The analysis, strategies for improved performance, and continued monitoring of improvement can result in the following benefits:

- Heightened awareness of user frustrations
- Better alignment of priorities
- Sharper IT management focus
- Improved IT requirements planning
- More effective IT resource allocation
- Enhanced quality of IT services
- More competitive IT services
- Increased IT customer satisfaction
- Greater productivity and return on investment

(Redman, 1998)

CASE NOKIA

6. NOKIA

Nokia is a global company whose key growth areas are wireless and wireline telecommunications. A pioneer in mobile telephony, Nokia is the world's leading mobile phone supplier as well as a top supplier of mobile and fixed telecom networks and services.

Nokia also creates solutions and products for fixed and wireless datacommunications. Multimedia terminals and computer monitors round out our expertise in communications technology. (Nokia In Brief)

6.1 Nokia IM

Nokia Information Management (IM) is a global Nokia function, which creates, deploys and delivers information management services for all Nokia businesses and employees. IM services include applications and services for business communications, demand/supply chain, management and support, and product creation. IM also offers standard infrastructure services and end-user support for Nokia applications and users.

IM´s aim is to become a trusted strategic partner to Nokia businesses by
- Proactively creating enterprise-wide IM services that meet the business needs
- Rapidly deploying and efficiently delivering these services
- Internally piloting and creating a showcase for Nokia's own products

As a global Nokia function, Nokia IM operates close to the business, in all the same locations where Nokia has business operations. Nokia IM's geographical areas and their central locations are:

- Americas, Dallas
- APAC (Asia Pacific), Singapore
- China, Beijing
- Continental Europe, Düsseldorf
- Finland North, Oulu
- Finland South, Espoo
- Finland West, Salo
- UK&Ireland, Camberley

In addition, Nokia IM Service/Help Desks and On-Site Support operations are in most major countries and cities in the world. (This is Nokia IM)

6.2 Delivery Process Application Services (DPA)

Delivery Process Application Services (DPA), (formerly Demand/Supply Chain Application Services) is one of the three application service groups in Nokia Information Management (IM)/Application Services (APS).

Based on Nokia strategy, DPA creates and deploys end-to-end demand/supply chain IM services to Nokia Business Groups. That means IM services for Nokia´s product delivery process.

In the creation and deployment of IM services, DPA responds to the rapidly changing needs of Nokia Businesses. Organizationally, DPA supports Nokia business processes in the following application areas: Demand Creation and Account Management, Demand/Supply Planning Applications, Demand/Supply Chain Transaction Applications, Service Support Applications and NET Operations Applications. Nokia-wide e-business development is also part of DPA.

DPA Application Deployment is organized and carried out by Business Groups. DPA Advanced Application Support, which is part of the Nokia IM Global Support Concept, is responsible for Advanced Application Support and implementation of the Global Support Concept for DPA. Infra Support for Application Services is a part of DPA as well.

Tapio Niskanen, Director, IM Application Services - is heading the DPA organization, which employs 360 people as of the end of March 2000. The organization has locations in Espoo, Salo, Oulu, Haukipudas and Bochum, Germany. (DPA Intro)

6.3 Sales Configurator

Nokia Sales Configurator (NSC) is an application developed by Nokia IM for Nokia Networks Oy Business Units and National Organisations. With Nokia SC, users can create product configurations based on customer requirements in both Sales (tendering) and Delivery processes (sales order processing), see Figure 15. SC provides users with an easy-to-use interface for accurate and up-to-date creation of product configurations. SC will replace some of the current configurators, and it provides interfaces to other applications such as MLS, see Figure 14.

Nokia Sales Configurator covers the following functionalities:

- Tendering
- Product configurations
- Sales order processing
- Pricing

(SC Home)

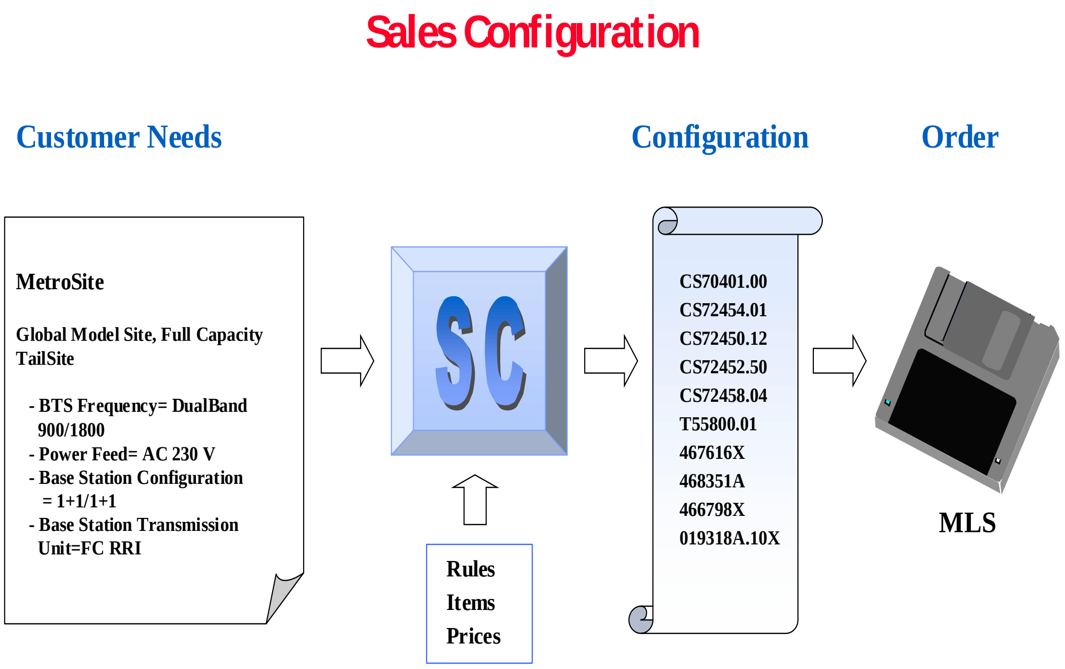

Figure 14. Sales configuration

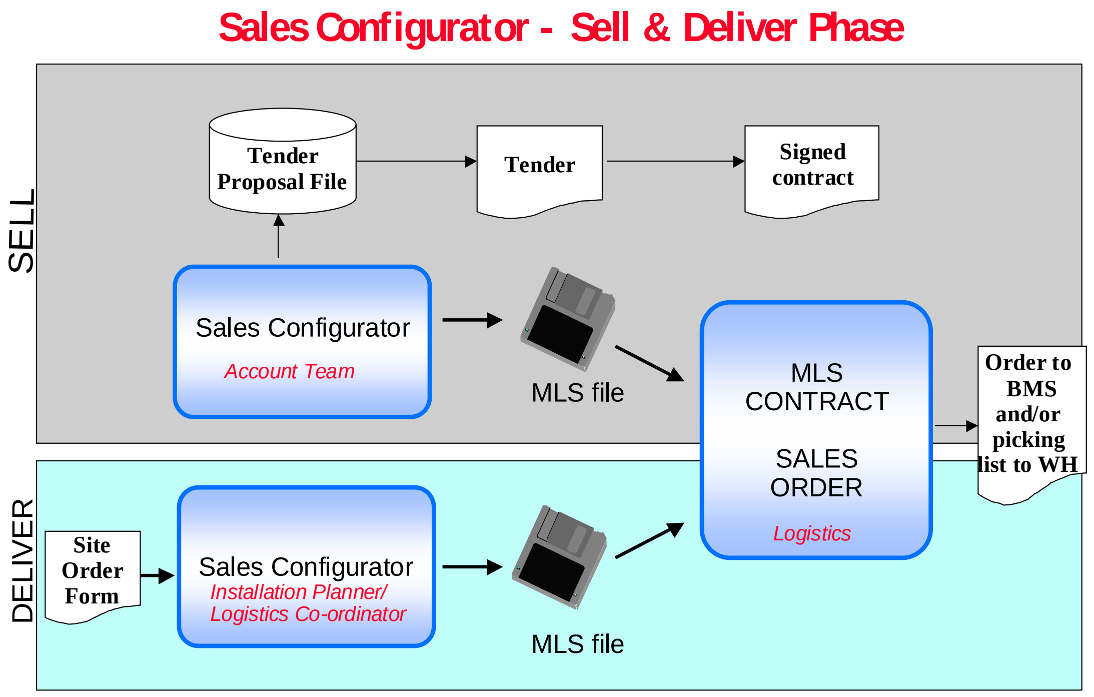

Figure 15. Sales configurator; sell & deliver phase

Figure 14 illustrates the sales configuration in principal. Customer needs, i.e. what products/product families are to be included in the desired system, are input to SC. SC makes an item list with prices according to the rules; it can be transferred to MLS (Modular Logistic System).

The upper part of Figure 15 describes the use of Sales Configurator in the tender phase. The account team can make the tender using the pricing information in the SC. In the lower part, the configurator is being used to produce the part list to be used in the logistics.

BMS - Business Unit Logistics Management System - is an operative logistics tool used by Nokia Networks Business Units. Currently Locos is the only BMS system in use.

WH - Warehouse

7. PRESENT PROCESSES

In the beginning of the software development process, requirements are handled with Word documents; these Requirements Specification documents are static. Once the software has been released, these documents are meaningless. After that, in practice, all the feedback to the development team comes in the form of Change Requests (CRs) or System Investigation Requests (SIRs). The different channels through which these requests come are illustrated in Figure 16.

Live release means the latest published version of the software. Release X is the next release to be published, and sometimes there are already requirements that affect the releases after that, due to a tight schedule. A release plan is used to decide which features will be implemented in each release.

Other parts of the figure are described in detail in the following chapters: Deployment (chapter 7.10), Key User Workshop (chapter 7.9), Key User (chapter 7.4.3), Service Desk (chapter 7.4.2), User (chapter 7.4.1), CR/SIR processes (chapter 7.7).

SPs are Service Packs that are minor releases to fix some of the defects between releases.

7.1 Feedback Channels

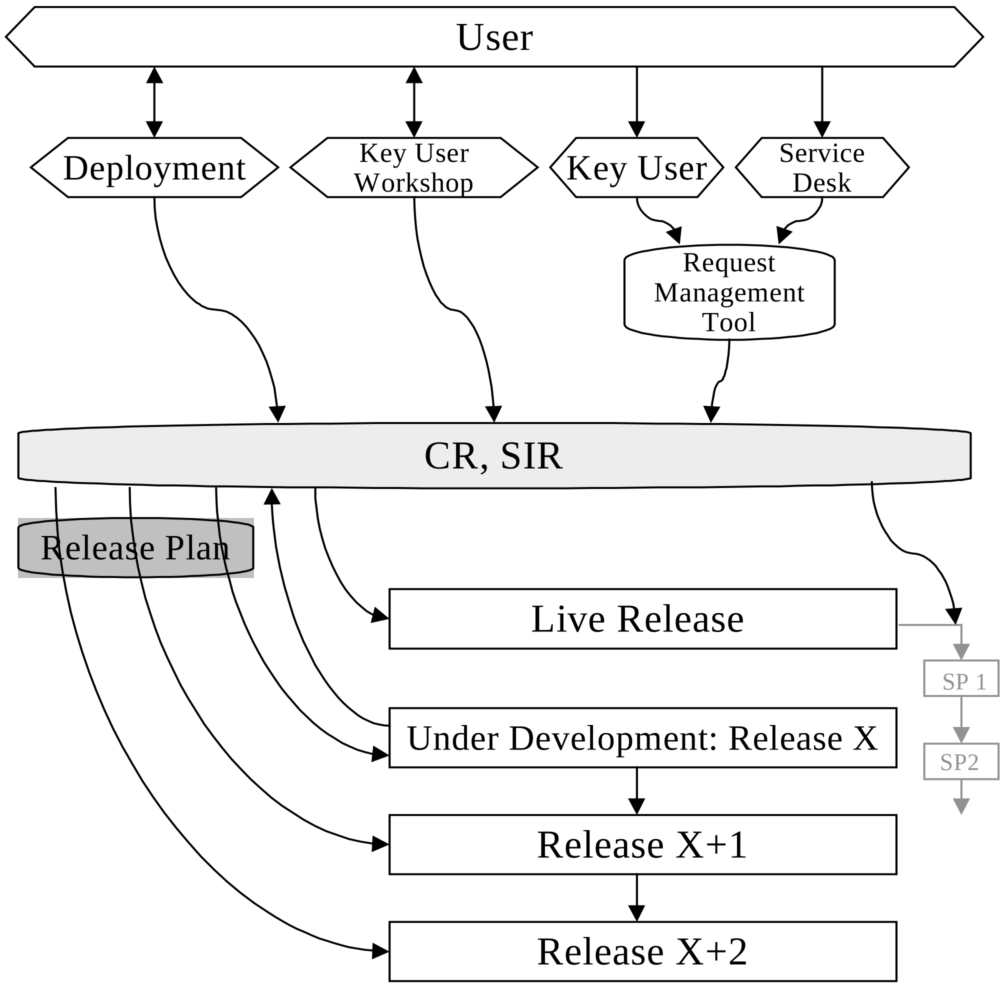

Figure 16. Feedback channels

7.2 Service Delivery

Service delivery is made within a Global Support Concept. The concept comprises Service Level Agreements (chapter 7.3) and a Global Support Model (chapter 7.4).

The Service Level Agreement document presents in detail the products, services, service levels and fees provided by Nokia IM. It explains the responsibilities of different organisations and describes measures for service performance. Site-specific needs are taken into consideration in a Local Implementation Document.

The Global Support Model presents the teams and organisation that will provide the services. It consists of three tiers:

Tier one involves Key Users and Service Desk services for the receiver organisation or site. Tier one is the primary contact point for end users.

Tier two involves application support, operations center and on-site support and local computing services.

Tier three involves advanced application support and infrastructure support at a global level.

Concept owners interact with the tier three and tier two support and configuration owners by defining business needs and priorities. (Services 2000 Catalogue)

7.3 Service Level Agreement

A Service Level Agreement (SLA) is an agreement between Nokia IM and Nokia Businesses/ Global Functions. The SLA states the delivery terms and conditions for IM Services (applications). The SLA defines the following tasks:

- Service content
- Responsibilities of both parties, IM and Business
- Service performance
- Target levels for service performance
- Measurement
- Corrective actions

Why do we need SLAs? An SLA is a good way of harmonizing service expectations. When the above-mentioned tasks are defined, both business and IM know what is included and what doesn't belong in the service.

Who needs SLAs? The service level defined in an SLA affects many Nokia employees. It is important for the users to know what kind of IM service they can expect and where to contact in order to get problems solved.

Nokia businesses and Global Functions have the possibility to plan their own operations and follow IM costs (since they are paying for the services IM provides) with SLAs.

Also Nokia IM itself can concentrate on those services our customers require. IM has better possibilities to plan its operations and resources and in the long run achieve cost-efficiency.

An important and interesting part of SLAs are the metrics. The performance of IM is measured in the following ways:

User Satisfaction Surveys (IM/Application specific surveys)

- Service quality, through
  - On-time deliveries in incident resolution
  - Completed service requests within SLA time targets
  - Percentage of first-pass cases in incident resolution
- Availability of Service (system working)

The quality of service is monitored with periodical SLA monitoring reports.

SLAs have been signed for the following IM services:

- Messaging for Nokia (Outlook)
- SAP R/3 Logistics service for NMP
- MLS (Modular Logistics System) service for NET
- SBM Platform Support for NET Engineering Services
- BMS (Business Unit Marketing System) service for NET
- Infrastructure services for Nokia
- Planning and Reporting Application Service for Nokia

- SAP R/3 FSP (Finance Platform) for Nokia FSP to be signed by the end of June

(Himmanen, 2000)

7.4 Global Support Model

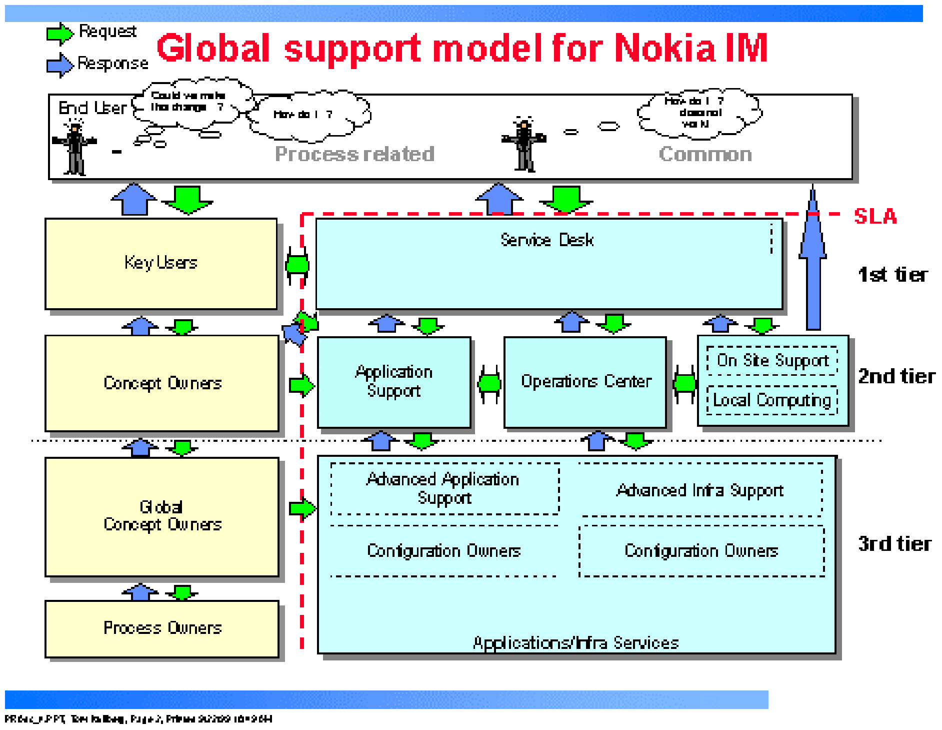

Figure 17. Global support model for Nokia IM

When an end user has a problem, if it is process related he/she should contact his/her key user; if the problem is 'common' he/she contacts the service desk. A key user can also solve the problem himself/herself; if this is not possible, he/she contacts the concept owner.

For application-related problems, for example how to use a certain application, a user contacts the nearest service desk, which is able to solve trivial problems. In the case of SC they normally just write it down and the handling responsibility is transferred to the 2nd tier. Each geographical area has its own Application Support, where application related problems are handled; at the moment there are two people in Dallas, one in China, one in Sydney, and one in Finland. Operations Centers are also area specific and they handle issues concerning operations. On-Site Support means people in place who can come to the person with the problem if they think they can solve it. Local Computing people handle for example server-related problems. If the 2nd tier is unable to solve the problem, it is transferred to the 3rd tier, which is global. Advanced Application Support handles application and platform-related problems, and Advanced Infra Support infra related. The last option in this chain is Configuration Owners, if none of the preceding steps has solved the problem. That is the development group of the application under consideration. In this process, RMT is used to monitor the progress of each issue.

In product and process-related issues, there are no tools for making requests or questions, but e-mail or telephone are used to communicate between the different people involved in the process.

7.4.1 Service User

Service Users are Nokia employees whom Service Desks and Key Users support in IT-related requests. A Service User may have two different kinds of problems: **process related** (I don't know how this application works or could I do this in a different way) and **common** (my computer/network etc. does not work).

Service Desk is the single point of contact for Service Users in common problems. Key Users are contacted for a requested solution to a business process-specific application, e.g. SAP Logistics, MLS, SAP FSP. Each request is logged into the request management tool, to follow up the solution progress within the agreed service level.
(Nokia Intranet)

7.4.2 Service Desk

Service Desk is a single point of contact offering support e.g. by phone or e-mail for IM Service Users. It is responsible for providing day-to-day care and problem resolution.

The main tasks of Service Desk are:

- Informing end users regarding:
  - service up-time and availability,
  - recovery & performance,
  - coaching and training
- Monitoring performance of IM
- Logging service requests into the request management tool, which provides statistics about their service level, offering information about user satisfaction
- Informing the service user of a service request's status
- Resolving requests if possible (e.g. over the phone)
- If immediate resolution is not possible, escalating requests to second tier personnel
- Communicating with the users in the local language (and English) if possible
- Proactively looking for root causes of continual requests by root cause and trend analysis
- Tier 1 event monitoring of application software, system software, hardware and networks
- Managing the site inventory of assets for end user terminals (e.g. hardware, licenses)

(Nokia Intranet)

7.4.3 Key User

The Key User is responsible for the effective use of the following business process-specific applications in the local business organisation: Demand/Supply Chain Applications, Management Support Applications, Product Creation Applications, and Business Communications Applications.

The main tasks of the Key User are:

- Answering application users' questions, assisting request management by logging and escalating application user requests. When needed she/he also raises application change requests, forwards them to the Concept Owner and maintains local reporting.
- Training application users, participating in testing new application releases and acting as local communicator of the above-mentioned applications.

(Nokia Intranet)

7.4.4 Concept Owners (Regional)

The Regional Concept Owner represents the regional business process perspective in the global concept development, together with the concept owner network.

The main tasks of the Regional Concept Owner are:

- Managing regional concept specification work when the regional process owner has accepted the concept specification.
- Refining process requirements for regional systems support, and supporting application implementation in her/his area, providing feedback to the Global Concept Owner.

(Nokia Intranet)

7.4.5 Application Support

Application Support is responsible for combining local process understanding with business-specific application expertise.

The main tasks of Application Support are:

- Monitoring and solving escalated requests, supporting Service Desks, Key Users and Concept Owners.
- Informing Key Users of functionality changes and offering training materials.
- Assisting in coordinating upgrade services, maintenance tasks and delivering advanced training programs (applications, processes, tools) for Key Users.

(Nokia Intranet)

7.4.6 Operations Center

Operations Center is responsible for providing a centralised backbone of systems and services for the delivery of common Nokia-wide IM Services.

The main tasks of the Operations Center are:

- Solving and monitoring escalated requests
- Performance, capacity and event monitoring of systems which are OPC's responsibility
- Computing and networking operations
- Insulation, change control and operations for defined IM applications and infrastructure services, e.g. Exchange servers, routes

- Software and data distribution for Local Computing, On-Site Support and Service Desk
- Infrastructure capacity planning and implementation support, hardware maintenance planning
- Disaster recovery and contingency planning in the OPC area
- Username operations for main computing platforms and System Management operations for all services
- Security
- Management of the site inventory of assets (servers, networks)
- Purchasing of hardware, networking, system software and infrastructure for server and network platforms which are not covered by global procurement

(Nokia Intranet)

7.4.7 On-Site Support

On-Site Support is responsible for offering personal support to the Service User at the user's workplace or at the Service Point, if the request cannot be solved in the Service Desk by phone or email.

The main tasks of On-Site Support are:

- Providing a local Service Point where users can get immediate service, e.g. a standby laptop or desktop, cables, cartridges and diskettes.
- Installing terminals and defined software for laptops, desktops and communicators, as well as other Service User devices like printers, scanners and plotters, also taking care of the change control.
- Providing IT facilities on the site, e.g. network connections at the workplace.

(Nokia Intranet)

7.4.8 Local Computing

Local Computing is responsible for supporting locally operated systems, including both local and partially global systems. It provides system support on-site when remote management tools cannot be used.

The main tasks of Local Computing are:

- Performance, capacity and event monitoring for locally operated systems, their hardware maintenance and fault management, as well as disaster recovery and contingency planning.
- Purchasing of local hardware, networking, systems software and infrastructure equipment for Service User needs (e.g. laptops, desktops) which are not covered by global procurement.
- Managing the site inventory of computing and networking assets (hardware, licenses).
- Installing and operating e.g. file servers, network printers, computer rooms and taking care of their change control.

(Nokia Intranet)

7.4.9 Concept Owner (Global)

The Global Concept Owner is responsible for transforming global business process requirements into concept specifications.

The main tasks of the Global Concept Owner are:

- Defining global process requirements for systems support when the process owner has accepted the concept specification
- Maintaining the concept functionality map and facilitating and supporting regional concept specification work

- Implementing solutions: e.g. tests pilot & local system functionalities and training concepts
- Managing the regional concept owner network, facilitating the local key user network, and maintaining the list of key users

(Nokia Intranet)

7.4.10 Advanced Application Support

Advanced Application Support is responsible for facilitating Nokia organizations globally to manage their applications, acting as an interface between application support and application creation teams.

The main tasks of Advanced Application Support are:

- Monitoring and solving escalated requests
- Creating change requests in response to raised requests
- Global coordination of application service levels, maintenance of global support documentation
- Planning and delivering advanced training for Application Support
- Planning and coordinating application support and software upgrade deployment
- End-to-end responsibility for service support
- Launching and attending selected development projects when appropriate (e.g. support tool development)

(Nokia Intranet)

7.4.11 Advanced Infra Support

Advanced Infrastructure Support is responsible for the technical solutions of applications.

The main tasks of Advanced Infrastructure Support are:

- Monitoring and solving escalated requests, execution of escalated requests
- Planning and delivering technical training for Operations Centers and support persons
- Infrastructure related product software upgrade deployment support (technical aspects)
- Global performance, capacity and event management
- Global infrastructure maintenance and contingency planning

(Nokia Intranet)

7.4.12 Configuration Owners

The Configuration Owner is responsible for the system configuration – the creation.

The main tasks of the Configuration Owner are:

- Consolidation of business needs from concept owners and configuration owners
- Defining required systems in business process analysis
- Application release and version management, new release training planning
- Configuration management of releases from development to production
- Development of integration interfaces and implementation of system integration tests
- Design, programming and testing management of new system features
- Control of user acceptance tests of the systems (Nokia Intranet)

7.5 User Support

The service provides resolutions to issues related to the use of SC in production and testing environments. User support also helps users with any questions they may have concerning the use of the application.

During working hours, service provision starts when an issue has been escalated to Nokia IM Service Desk or Application Support. The response time is the time taken for an issue to be assigned to a specific individual in Nokia IM and a notification message sent to the issue creator. The issue is classified as resolved once:

- A resolution has been successfully implemented by Application Support and communicated to the issue creator, or
- A resolution has been proposed to the issue creator/originator for them to implement, or
- A resolution for future implementation has been proposed to the issue creator and, where necessary, a work around has been proposed.

An issue is classified as closed once the issue originator is satisfied with resolution. (Services 2000 Catalogue)

7.5.1 Levels of User Support and Issue Resolution Times

Table 1 shows the promised response and resolution times for different kinds of problems.

Table 1. Levels of user support and issue resolution times:

Category	Attributes	Response Time	Resolution Time
<u>Critical</u> Level 1	· Complete system failure · No staff can work · Business impact at every level	Immediate	10 hours
<u>High</u> Level 2	· Complete failure of a critical system / application component · Some staff unable to work · Major business impact	2 hours	40 hours
<u>Medium</u> Level 3	· Complete failure of a non-critical system / application component · Partial failure of a system / application component · Specific staff affected · Minor business impact	4 hours	100 hours
<u>Low</u> Level 4	· Advice required · No noticeable business impact	8 hours	Min.150 hrs (user defined)

(Services 2000 Catalogue)

7.6 Training

Application training is available according to the target service level. After the initial training during the roll-outs, further training will be arranged in regional learning centers. The training components are:

- SC Tool training
- Product Configurator training
- Key User training
- New Release functionalities

(Services 2000 Catalogue)

7.7 CR and SIR Processes

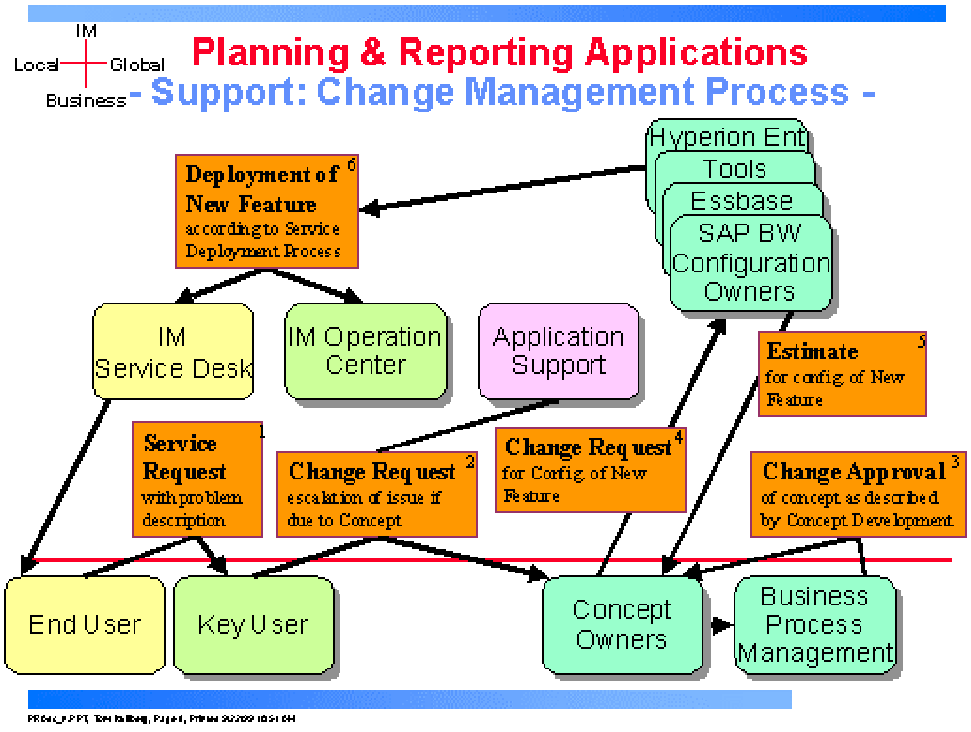
IM
Local
Global
Business
Planning & Reporting Applications
- Support: Change Management Process -
Hyperion Ent
Tools
Essbase
SAP BW
Configuration Owners
Deployment of New Feature according to Service Deployment Process 6
IM Service Desk
IM Operation Center
Application Support
Estimate for config. of New Feature 5
Service Request with problem description 1
Change Request escalation of issue if due to Concept 2
Change Request for Config. of New Feature 4
Change Approval of concept as described by Concept Development 3
End User
Key User
Concept Owners
Business Process Management

Figure 18. Change management process

Figure 18 represents the Change Request process when the CR is being initiated by an end user. There are also other channels through which CRs and SIRs are generated, e.g. Key User Workshops, Testing, Deployment, Development team, and Business Units.

At the moment, feedback from the users comes mostly through a complicated process which handles CRs (Change Requests) and SIRs (System Investigation Requests). SIRs are used for reporting bugs and other malfunctions of the software. CRs can be also used to make suggestions about how a certain feature should work.

A change or modification to the system is often referred to as a 'CR' or 'Change Request'. An enhancement is a change to the previously agreed functionality and is different from an 'SIR', which requires a fix to the system in order to meet the previously agreed functionality. A System Investigation Request is often referred to as a 'Bug'. An SIR is a recognised problem with the system that requires further investigation and may require a fix to the application code. Therefore SIRs are considered more urgent and are prioritised in terms of schedule and resources allocated.

The CR/SIR tool is being used via a Lotus Notes database, in which the originator completes a form including all the relevant information required. This form includes information about the originator, release, desired functionality, priority, status etc. (see appendix 1 for more detailed information).

7.7.1 SIR Process

SIR priority definitions:

Critical: The Error is classified as a Crash-level Error if the Software or Maintainable Deliverables cannot run, or service is crippled as to be useless, or a time critical user job is stopped, there are data corruption problems, or a critical malfunction or deficiency endangers business, and there is no workaround available.

High: The Error is classified as a High-level Error if an important operational user job is stopped, or a time critical user job is at hazard, or an important Software or Maintainable Deliverables component is unusable, or a system or product malfunction due to deficiency or non-usability has frequent or major end-user impact, or there is a frequent failure of an important service.

Medium: The Error is classified as a Medium-level Error if the Software or Maintainable Deliverable is hampering progress, or a non-urgent job does not run, or an intermittent fault is causing inconvenience, or a system or product malfunction due to deficiency or non-usability is having infrequent or minor user impact.

Low: The Error is classified as a Low-level Error if the Error has no current impact on the user, or there is a locally identified cure or workaround available. It is passed on for information purposes only to ensure registration of the problem and clearance as appropriate.

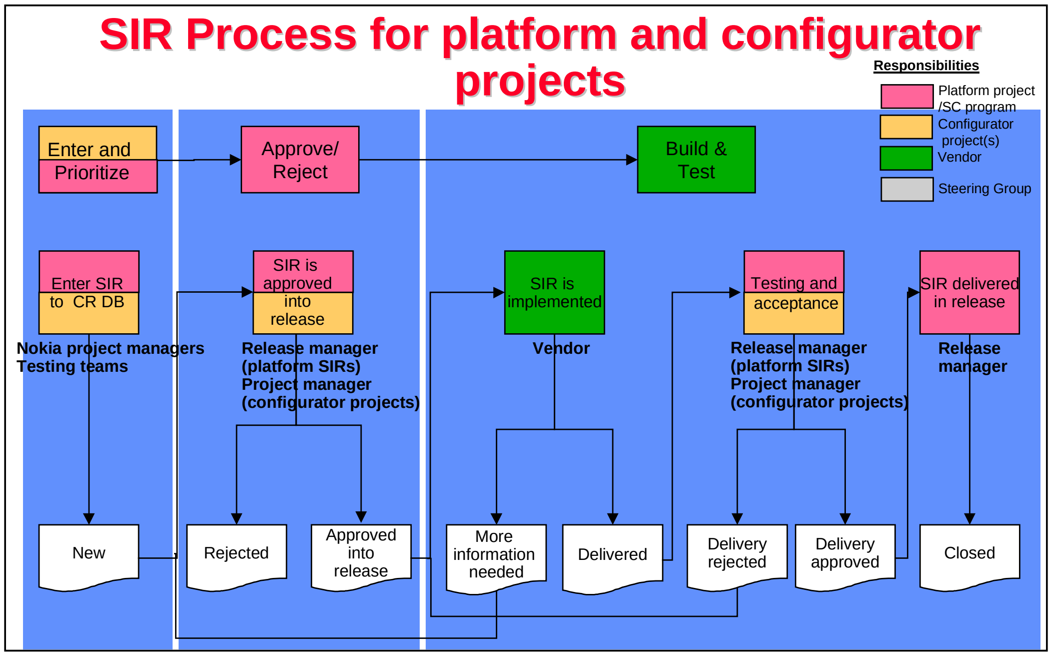

Figure 19. SIR process

The SIR process is represented in figure 19; table 2 defines the responsibilities in SIR handling in different phases of the process.

SIR statuses:

New: The originator has created the SIR. The release manager (for platform SIRs) or configurator project manager (for configurator SIRs) is working with the SIR.

Rejected: The release manager (for platform SIRs) or configurator project manager (for configurator SIRs) has rejected the SIR because it is not valid, has insufficient information, or is a duplicate of another SIR already in the database.

Approved into release: The release manager (for platform SIRs) or configurator project manager (for configurator SIRs) has approved the SIR into implementation. The vendor is implementing the SIR.

More information needed: The vendor has not been able to implement the SIR because there is not enough information provided in the SIR. The release manager (for platform SIRs) or configurator project manager (for configurator SIRs) is working to obtain the information needed.

Delivered: The vendor has implemented the SIR and it has been delivered in release. Testing is currently underway.

Delivery approved: The test team has approved the implementation of the SIR. The release manager is currently validating the delivery of the SIR.

Delivery rejected: The test team has disapproved the implementation of the SIR. The vendor is currently fixing the SIR.

Closed: The SIR has been delivered in release and the release is available from the release manager.

Not Repeatable: The SIR has not happened when trying to repeat the procedure causing the SIR.

Table 2. SIR process responsibilities:

#	Responsible person:	SIR status before action:	Action:	SIR status after action:	View used:
1	Project key persons, testers, SC support	None	Create the SIR in the CR database defining all necessary data in the SIR	New	None
2	Release manager (platform CRs) Configurator project manager (configurator CRs)	New More information needed	Verify that the SIR has all the necessary data and that no duplicate SIRs exist in the CR tool database. Either turn the SIR into status Approved into Release for implementation, or reject it by turning it into status Rejected and inform the creator by mail.	Approved into release Rejected	Release manager view, Configurator project manager view
3	Vendor	Approved into release Delivery rejected	Implement the SIR according to information provided in the SIR and turn the SIR into status Delivered. If the SIR cannot be implemented because necessary information is lacking, turn the SIR into status More information needed.	Delivered More information needed	Vendor view

4	Testing team (Release manager) (Configurator project manager)	Delivered	Test the SIR in the release where it is implemented. If SIR implementation is acceptable, turn the SIR into status Delivery approved. If SIR implementation is not acceptable, turn the SIR into status Delivery rejected and give the reason for rejection.	Delivery approved Delivery rejected	Testing view
5	Release manager	Delivery approved	Based on release notes provided by the vendor, verify that all SIRs in release have been delivered and tested. Verify that all SIRs have the needed information and documentation. Turn SIRs in release into status Closed.	Closed	Release manager view

7.7.2 CR Process

CR priority definitions:

Critical: Must be implemented. Even the release schedule can be changed to get these features.

High: Must be implemented. However, there is a (cumbersome) workaround, which could be used if the schedule is not sacrificed. If skipped, must be implemented in the next minor release.

Medium: Should be implemented. However, the schedule is not sacrificed. If skipped, must be implemented in the next major or minor release.

Low: 'Nice to have' feature. If the schedule allows, can be implemented. If skipped, prioritization must be considered again in the next (minor or major) release.

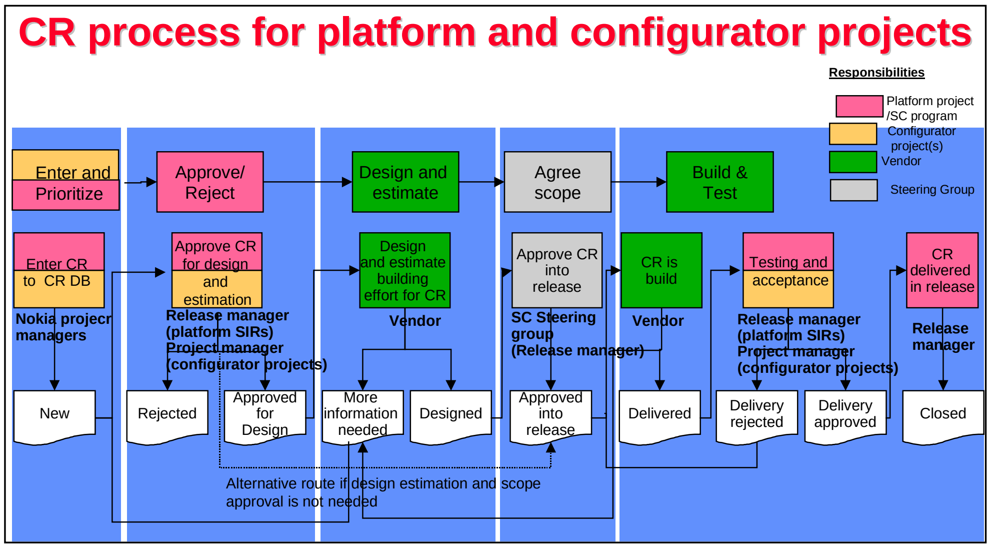

Figure 20. CR process

The CR process is represented in figure 20; table 3 defines the responsibilities in CR handling in different phases of the process.

CR statuses:

New: The originator has created the CR. The release manager (for platform CRs) or configurator project manager (for configurator CRs) is working with the CR.

Rejected: The release manager (for platform CRs) or configurator project manager (for configurator CRs) together with the originator of the CR has rejected the CR because it is not valid, has insufficient information, or is a duplicate of another CR already in the database.

Approved for design: The release manager (for platform CRs) or configurator project manager (for configurator CRs) has approved the CR into design and estimation. The vendor is currently working with design and estimation.

More information needed: The vendor has not been able to design or implement the CR because there is not enough information provided in the CR. The release manager (for platform CRs) or configurator project manager (for configurator CRs) is working to obtain the information needed.

Designed: The vendor has designed a solution and estimated implementation effort for the CR. Release scope definition is currently underway to decide if the CR will be implemented.

Approved into release: The SC Steering Group has approved the CR into SC release. The vendor is currently implementing the CR.

Delivered: The vendor has implemented the CR and it has been delivered in release. Testing is currently underway.

Delivery approved: The test team has approved the implementation of the CR. The release manager is currently validating the delivery of the CR.

Delivery rejected: The test team has disapproved the implementation of the CR. The vendor is currently fixing the CR.

Closed: The CR has been delivered in release and the release is available from the release manager.

Table 3. CR process responsibilities:

#	**Responsible person:**	**CR status before action:**	**Action:**	**CR status after action:**	**View used:**
1	Project key persons, testers, SC support, BU representatives	None	Create the CR into the CR database, defining all necessary data in the CR.	New	None
2	Release manager (platform CRs) Configurator project manager (configurator CRs)	New More information needed	Verify that the CR has all the necessary data and that no duplicate CRs exist in the CR tool database. Either turn the CR into status Approved for design to have it designed by the vendor, or reject it by turning it into status Rejected and inform the creator by mail. Note! It is also possible to turn the CR into status Approved into release for implementation if design and scope approval is not needed.	Approved for design Rejected (Approved into release)	Release manager view, Configur ator project manager view
3	Vendor	Approved for design	Design the CR according to information provided in the SIR and provide an estimate about implementation effort. If the CR cannot be designed because necessary information is lacking, turn the CR into status More information needed.	Designed More information needed	Vendor view

4	SC Steering Group (Release manager)	Designed	Decide the scope of the release. Turn all CRs to be implemented into status Approved into release and indicate the release in which the CRs are to be implemented.	Approved into release	Scope approval view
5	Vendor	Approved into release Delivery rejected	Implement the CR according to information provided in the CR and turn the CR into status Delivered. If the CR cannot be implemented because necessary information is lacking, turn the CR into status More information needed.	Delivered More information needed	Vendor view
6	Testing team (Release manager) (Configurator project manager)	Delivered	Test the CR in the release where it is implemented. If CR implementation is acceptable turn the CR into status Delivery approved. If CR implementation is not acceptable turn the CR into status Delivery rejected and give the reason for rejection.	Delivery approved Delivery rejected	Testing view
7	Release manager	Delivery approved	Based on release notes provided by vendor, verify that all CRs in release have been delivered and tested. Verify that all CRs have the needed information and documentation. Turn CRs in release into status Closed.	Closed	Release manager view

SIRs are usually implemented as soon as possible. When concerning a live environment, if the problem is in data it can be fixed by fixing the data in the database. If the problem is in the code of the software, it cannot be fixed before the user reinstalls the software; the fix will be (hopefully) in the next release. Some of the 'data' is in the code of the software, e.g. some rules concerning the products. If it is very important to fix the problem, and the next release is not coming out in the near future, there is one option left. To get the problem fixed between releases it can be done by using a patch, an executable program that fixes the problem when executed in the client machine. These patches are called Service Packs and sometimes there may be several Service Packs out between the complete releases.

There is a wide variety of Change Requests (CRs), some of which concern wholly new features to the software and some are pretty meaningless like 'the button should be green, not blue', and naturally everything between those extremities. Even if there is categorization regarding the importance of a CR, it may not always show the real situation, because it is decided by the originator of the CR and may be exaggerated to ensure that the CR is recognized. Most of the time the schedule is so tight that the CR may not be implemented, if at all, even in the next release.

7.8 RMT

The Request Management Tool is being used to monitor problem solving activities and people responsible within the support organisation. It monitors, among other things, handling times between different phases of the procedure, and the overall times. RMT is not the same as the CR tool and it is being used by the support organisation, and some of the issues handled in RMT may later become a SIR or a CR.

7.9 Key User Workshop

Key User Workshops are held after or simultaneously with the roll-out of a new release. The main goal is to train the key users to use the latest version of the software. Usually some feedback is gathered from these events.

7.10 Deployment

Three main events: server installation, client installation, user training.

During the deployment phase there are events regarding piloting and user training. User training events include case training, and user problems are documented for further analysis, compared to existing CRs/SIRs.

The SC Implementation service includes technical implementation project management and system specialist services which carry out the implementation project according to the SC implementation methodology. Implementation is done usually in a rollout country by teams consisting of IM and Business Unit people, where IM people carry out SC installation, training arrangements and system training, and BU people carry out configurator training. The required system consultation services are included. The SC Project includes the following phases:

- Project preparation (planning, scope definition, informing)
- Realisation (installation, training)
- Go live & support

(Services 2000 Catalogue)

8. RESULTS

The present processes are concentrated on fixing the problems and malfunctions of the software. There are no media to support the gathering of brand new ideas for developing the software better or adding new functionalities or features to it.

One problem is also that the creation and development people are so far away from the actual users of the software. Virtually none of the problems they have with the software are coming to their attention. That is because there are so many 'filters' on the way and most of the problems are solved by the key user or the service desk, and in such cases upper tier people are not notified of such incidents.

Gathering all the requirements in one place, i.e. a database, and giving different groups different kinds of access rights to the data would solve many of the current problems concerning requirements management. Requirements management tools provide the necessary functionalities to make it happen.

8.1 Requirements Management Tool Workshops

I participated in four different workshops organised by Osmo Vikman from Nokia Research Center. The tool vendors were there to represent their tools. It was not only about the tools themselves, but they also described the processes the tools are supposed to support.

These four vendors and their tools were: Quality Systems & Software: DOORS and DOORSnet, Rational Software: Analyst Studio & RequisitePro, Technology Builders, Inc.: Caliber-RM, and TD Technologies/SDRC: SLATE & TranSLATE.

There is a certain methodological background to almost every Requirement Management Tool. QSS have even written several white papers and books concerning their idea of requirements management and naturally how their tool (DOORS) fits the overall process of requirements management.

DOORS is being used by other Nokia BUs. One of the reasons this tool was selected to be evaluated more closely was the positive experiences of the other BUs. The purpose of the evaluation was to assess how it would fit our own purposes, i.e. managing the Sales Configurator's requirements.

8.2 Requirements Management Process

Figure 21 illustrates how a requirements management tool would fit into the current processes (see Figure 16) of gathering feedback and requirements from different stakeholders.

Using a requirements management tool would make it easier to document Business Units' concepts, i.e. the BUs' processes which the SC is supposed to support and their processes. The process roadmap refers to this development of BUs' processes. Technological progress also affects application development, and these are the technology and application roadmaps in the figure. One example of this kind of development is the transformation of SC to the web.

The advantage of a requirements management tool in this framework is the ability to handle totally different kind of requirements in the same database, and link these requirements to each other. It also allows the use of a hierarchy, so there can be requirements that are on different levels.

Different kinds of access rights can be assigned to different user groups or even individual users (more in chapter 8.3.1). Using DOORSnet, the database can be accessed through a web browser anywhere (chapter 8.3.2).

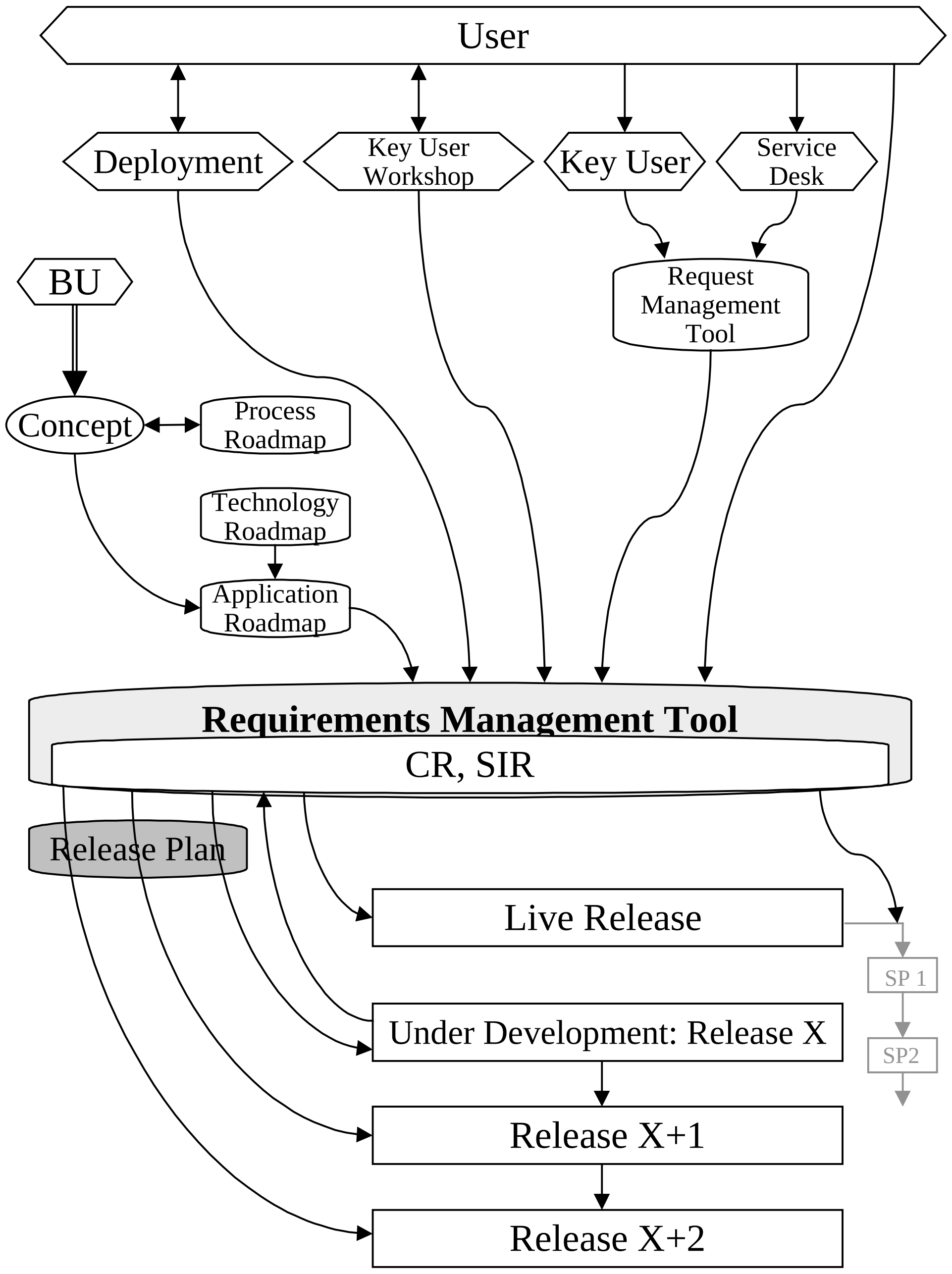

Figure 21. Requirements management process

8.3 Requirements Management Tool Advantages

There are several advantages of using a requirements management tool compared to handling requirements through static, e.g. Word documents. QSS Inc.'s DOORS was evaluated in more detail so the benefits are here described as seen in DOORS, although most of them apply to all of the requirements management tools.

8.3.1 Access Rights

There is a wide variety of different kinds of access rights that can be assigned to different user groups or even individual users. These include: read, write, and change rights.

These features make it easy to assure that a user is allowed only to modify relevant requirements, it can be restricted so that a user does not even see those requirements that are not accessible to him/her.

8.3.2 DOORSnet

Using DOORSnet makes it possible to publish selected requirements to the web, where they can be seen using a web browser. In DOORSnet 2 it is possible to make change proposals against these requirements, and suggestions concerning anything. In DOORSnet 3, there will be also full edit capabilities.

8.3.3 Change Proposal System

CR and SIR processes could be handled with DOORS very well. The DOORS change proposal system can be accessed either through DOORSnet or DOORS. Using DOORS, DOORSnet, and suitable access rights, for example, the user could be given rights only to make suggestions through DOORSnet, the user would not see the requirements or other suggestions (and probably doesn't even want to see them). The deployment team, people involved in Key User Workshops, Key Users, and Service Desk would have DOORSnet access to make, in addition, change proposals to existing requirements. BU people and the development team would have all the DOORS features, and access depending on their position in the project.

The change proposal system makes a change proposal when it is suggested, and the requirement remains unchanged until approved by the person responsible. It also has an option to send e-mail to the proposer when the status of the change proposal is changed, so the proposer knows all the time what the status is, and why it was changed.

8.3.4 Export and Import

DOORS has excellent import and export features. Word and other requirements documents are easy to import in just minutes. The basic import functions include importing plain text, Rich Text Format (RTF), spreadsheet, MS Project, Framemaker, and Interleaf. Export includes, in addition, SGML (Standard Generalized Markup Language), SQL (Structured Query Language), MS Office products (Excel, Outlook, PowerPoint, Word), and HTML (HyperText Markup Language).

8.3.5 History and Baselines

Every requirement is handled individually. Every single requirement or other object has its own history and all changes can be seen; who has changed it, when, what he/she has changed, and possibly additional comment/rationale. This allows easy cancellation of changes, because all the history is recorded.

There is also a possibility to make baselines, i.e. versions of the whole document can be frozen and given version numbers such as 0.1, 1.0, …. This allows the user to see what the whole project was like when a baseline was made. Baselining is a good way to take snapshots of the database, for example when a certain milestone in the project has been reached.

8.3.6 Links to Various Other Applications

DOORS has a possibility of integration with various different kinds of applications through their Application Programming Interface. At the moment there are about 40 integrations. These include, e.g., analysis and design tools (Rational's Rose), CAD tools, test and risk management tools, and configuration management tools (Continuus).

8.3.7 Attributes and Views

Every requirement has a set of basic attributes which are automatically generated: Absolute Number, Created By, Created On, Created Thru, Last Modified By, Last Modified On, Object Text, Object Heading, Object Short Text, Object Text.

A user can add whatever attributes wanted in addition to these basic attributes. There are several attribute types which can be used, e.g.: Boolean, Date, Integer, Real, String, Text.

Using different views it can be selected which attributes show on the screen; these views can be saved for later use. Requirements can also be sorted (e.g. on the basis of creation date) and filtered (e.g. show only the requirements whose status is critical).

8.3.8 Traceability and Impact Analysis

One of the most important features is impact and traceability analysis. Requirements can be linked to other requirements, and therefore it can be analysed what other requirements are affected if only one requirement is changed (impact analysis), and vice versa, which of all requirements have an effect on a certain requirement (traceability analysis). This is very important when there are thousands of requirements and it is very difficult to clarify the possible effects of changing something. Usually, if not properly conducted, it leads to more problems than solutions.

8.3.9 Modifiability

70 % of the software is made using DXL. The DOORS eXtension Language (DXL) is a scripting language for controlling and extending DOORS functionality. It is a powerful, feature-rich language that is syntactically similar to C or C++. DXL can be used to automate routine or complex tasks, such as calculating attribute values, or responding to events by triggering custom programs. Even one's own functions can be added to DOORS menus.

8.4 Possible Problems Implementing Requirements Management Tool

Although DOORS is quite a simple tool to use, it requires some training to be used efficiently. This may not be a major obstacle, but the problem is the fear of anything new. It is difficult to introduce a new way of doing things without any resistance. It would also require acceptance and commitment from a relatively large group of people. One of the problems concerns costs compared to benefits. It may be difficult to show how much money it would save in the long run to use such a tool.

9. CONCLUSION

Software development is a complex process, and has a lot to do with the requirements for the software product. These are several different kinds of requirements, and these are presented in various levels; from the intended functionality of a certain part of the software to very detailed requirements (e.g. some minor detail in the user interface).

Managing these requirements is also very complicated, although in literature it is presented as a simple straightforward process which consists of several distinct phases.

The emphasis of this thesis was on how to handle changes in these requirements, other feedback after the software has been released, and how the overall process could benefit from using a requirements management tool.

Using a requirement management tool (RMT) does not solve any problems, but it gives the means to improve requirements management considerably. Some advantages of using RMT are: centralised storage of the requirements, using of different kinds of access rights for different users concerning access and changing the data, structured handling of the change management process, impact and traceability analysis, and access to the data using a web browser.

The evaluation of the tool was conducted in quite a simple manner, with only one user, and a small database. A more realistic conception of the use of this kind of a tool would require a pilot project where the tool would be used in a more realistic environment with real requirements.

REFERENCES

LITERATURE

Booth, Paul, **An Introduction To Human-Computer Interaction**, 1989, Lawrence Erlbaum Associates Ltd., U.K.

Duncan, William, R., **A Guide To The Project Management Body Of Knowledge**, 1996, Project Management Institute, PMI Publishing Division, USA

Encyclopædia Britannica, www.britannica.com, (27.6.2000)

Galitz, W.O., **Humanizing office automation**, 1984, Wellesley, MA: QED Information Systems

Gilb, T., **Principles of Software Engineering Management**, 1988, Addison-Wesley

Hartson, H.Rex, and Deborah, Hix (ed.), **Advances in Human-Computer Interaction**, Volume 2, 1988, Ablex Publishing Corporation, USA

Mayhew, Deborah J., **The Usability Engineering Lifecycle: A Practitioner's Handbook For User Interface Design**, 1999, Morgan Kaufmann Publishers, Inc., USA

McDermid, John A., **Software engineer's reference book**, 1991, Oxford Butterworth-Heinemann

Muench, Dean, 1994, **The Sybase Development Framework**, Oakland, Calif., Sybase Inc, USA

Pressman, Roger, S., **Software Engineering: A Practitioner's Approach**, third edition, 1992, McGraw-Hill, Inc., Singapore

Shneiderman, Ben, **Designing the User Interface**, 1987, Addison-Wesley Publishing Company, Inc., USA

ARTICLES AND PAPERS

Bevan, Nigel, **Measuring usability as quality of use**, 1994, National Physical Laboratory, Teddington, UK, ftp://ftp.npl.co.uk/pub/hci/papers/quality.rtf (27.03.2000)

Boehm, B., **"A Spiral Model for Software Development and Enchantment"**, Computer, vol. 21., no. 5, May 1988

Francas, M., Goodman, D., and Dickinson, J**., Command-set and presentation method in the Training of Telidon Operators**, Proc. of the Human Factors Society – 26th Annual Meeting, Human Factors Society, Santa Monica, CA, USA, 1982

Gabler, J., **IS as Service Provider: End the 'Negative Feedback Loop'**, Research Note , Tactical Guidelines, 5. November 1999, GartnerGroup, Inc., http://gartner3.gartnerweb.com/ (23.5.2000)

Garvin, **What does "product quality" really mean?,** Sloane Management Review, Fall 1984

Light, M., **What We Need Is Requirements Management**, Research Note, Markets, 3. November 1998, http://gartner5.gartnerweb.com/ (27.6.2000)

Light, M., Conway, B., **Requirements Management: Taming Scope Scourge**, Research Note, Strategic Planning Assumption, 15 April 1997, http://gartner5.gartnerweb.com/ (27.6.2000)

Redman, B., **IS Organizations Benefit Greatly From User Satisfaction Monitoring**, InSide Gartner Group, 1. July 1998, Gartner Group, Inc., http://gartner3.gartnerweb.com/ (24.5.2000)

van Veenendaal, E, and McMullan, J, **Achieving software product quality**, 1997, Tutein Nolthenius, Netherlands, ftp://ftp.npl.co.uk/pub/hci/papers/qualusab.rtf (27.03.2000)

NOKIA INTERNAL DOCUMENTS

DPA Intro
http://connecting.nokia.com/NOKIA/im/home/dpa.nsf/document/ES22T4J4F6N (30.5.2000)

GSC Homepage, http://domino.ntc.nokia.com/nokia/im/dca/gsc.nsf (30.5.2000)

Himmanen Satu, Immediate DPA News, June 2000, **IM Responsibilities Materialize in SLA**

SC HOME
http://connecting.nokia.com/nokia/IM/home/dpa.nsf/document/00002DFA (30.5.2000)

SC Introduction to team

Nokia In Brief
http://www.nokia.com/inbrief/index.html (30.5.2000)

Nokia Sales Configurator CR Tool; Process and Instructions for Creating, Managing and Implementing SIRs and CRs

Services 2000 Catalogue
http://domino.ntc.nokia.com/NOKIA/IM/DCA/gsc.nsf/document/00002272 (24.5.2000)

This is Nokia IM
http://connecting.nokia.com/NOKIA/IM/home/customer.nsf/document/ES22T2156 (30.5.2000)

ISO STANDARDS

ISO, 1981, ISO 6385: Ergonomic principles in the design of work systems

ISO, 1992, Directives Part 2 - Methodology for the development of international standards, ISO/IEC, Switzerland

ISO DIS 8402 (1994) Quality Vocabulary

ISO/IEC 9126 (1991) Software product evaluation - Quality characteristics and guidelines for their use

ISO/IEC CD 9126-1 (1997) Software quality characteristics and metrics - Part 1: Quality characteristics and sub-characteristics

ISO DIS 9241-11 (1996) Ergonomic requirements for office work with visual display terminals (VDT)s - Part 11: Guidance on usability

ISO/IEC 14598-1 (1997) Information Technology - Evaluation of Software Products - Part 1: General guide

ISO/IEC PDTR 15504 (1997) Software process assessment

INTERVIEWS

Hippeläinen Leo, Chief SW Architect, Nokia Radio Access Systems, Research & Development, 29.5.2000

Karjalainen Jukka, Application Specialist, Nokia Information Management, Delivery Process Application Services, Application Deployment, 18.4.2000

Katajamäki Harri, Project Manager, Nokia Information Management, Delivery Process Application Services, Demand/Supply Planning Applications, 7.4.2000

Kuusela Eero, Team Leader, Nokia Information Management, Delivery Process Application Services, Advanced Application Support, 18.4.2000

Räihä Marcus, Application Specialist, Nokia Information Management, Delivery Process Application Services, Advanced Application Support, 18.4.2000, 28.4.2000

Santakari Jouni, Project Manager, Nokia Information Management, Delivery Process Application Services, Demand/Supply Planning Applications, 7.4.2000

Vikman Osmo, Assistant Research Manager, Nokia Research Center, Software Technology Laboratory, 9.6.2000

OTHER

Systems/Requirements Engineering Tool Evaluation Workshops, Nokia Research Center, Helsinki

Analyst Studio & RequisitePro, Rational Software, 2.5.2000

Caliber-RM, Technology Builders, Inc., 5.5.2000

DOORS & DOORSnet, QSS, Inc., 9.5.2000

SLATE & TranSLATE, TD Technologies/SDRC, 11.5.2000

Nokia Requirements Management Forum, Nokia Research Center, Helsinki, 14.6.2000

APPENDIX 1. CR/SIR Fields

The following are the fields in the form and their suggested usage:

Header	Give a short description of the CR or SIR. This field is visible in most of the views so it should be clear enough to give an idea of the issue but short enough to fit in the space available. *(Required Field)*
Category	Change request (CR) or support/investigate request (SIR) **CR** is a change to current functionality of the system (i.e. the system is working as designed but the design needs to be changed or new functionality added) **SIR** is a problem/bug in the functionality of the system (the system does not work according to design) *(Required Field)*
Originator	The name of the originator. Format: Surname First name. *(Required Field)*
BU	The business unit of the CR/SIR originator. Select Nokia IM if you are unsure what BU to use. *(Required Field)*
Priority	This is the priority from the BU point of view. Options are *Critical*, *High*, *Medium* and *Low*. The priority is determined by the originator. *(Required Field)*

Product Configurator	The name of the configurator the SIR or CR is related to, for example, Metrosite. Use Platform for SIRs/CRs related to basic SC functionality that affects all configurators. *(Required Field)*
Test Type	The testing situation where the CR or SIR was found, for example UAT (User Acceptance Test).
Release	The actual release where the SIR or CR was encountered. Give as precise release information as possible (e.g. 1.1.9 instead of 1.1). *(Required Field)*
Software Module	The appropriate SC module the CR/SIR is related to.
Description	As detailed description as possible of the issue. If possible give step-by-step instructions of how to recreate the problem and information on the environment the SIR/need for CR was encountered in. *(Required Field)*
Current Functionality	Detailed description of the current functionality of the feature or process. Please be very detailed and include as much information as possible, for example, screens, error messages, etc. This will prevent unclear situations and confusion regarding the issue.
Desired Functionality	A detailed description of the desired (CR) /expected (SIR) functionality. Please attach files showing possible suggestions, for example, Excel sheets.

Business Reasons	State the business reasons for the suggested CR. Reasons must be tangible and preferably measurable. Without clear business reasons the CR will not be implemented. *(Required Field for CRs)*
Desired Release	Give the major release the CR/SIR should be implemented *(Required Field)*
CR/SIR number	This number will be system-generated.
Status	Status of the SIR/CR defines who is handling it and where in the process the SIR/CR currently is *(Required Field)*.
Currently Handling	The party which is responsible for the CR's or SIR's handling (automatically defined according to Person in charge).
Handling Date	The current handling date of the CR/SIR.
Subject	Subject of the SIR/CR. This field can be used to group multiple SIRs/CRs together. Enter the same subject for all SIRs/CRs you want to link together and use the Subject view to view them.
Nokia IM Comments	The comments entered by the handler (release manager, configurator project manager, tester).
Supplier Ref. Number	The vendor reference number.
Person In Charge	The person who is responsible for the handling of the CRs or SIRs.
Planned Target Release	The release in which this CR **should be** implemented. Decided by SC Steering Group

Delivered in Release	The release in which this CR/SIR **was** actually implemented.
Work Estimate	Estimate of the implementation effort from the vendor.
Planned Delivery	The planned delivery date.
Delivered	The actual delivery date of the SIR/CR.
Supplier Comments	Comments from the vendor.
Specification	The technical specification from the vendor.
Approved by	Release manager's comments and approval after the CR/SIR has been tested.