University of Koblenz Landau
Campus Koblenz
Faculty of Computer Sciences

Institute of Software Engineering
2003

Master Thesis

Maintenance Activities in Software Process Models: Theory and Case Study Practice

Student:	Urs Kuhlmann
Enrolment No.:	201210817
Address:	Karolingerstr. 7
	56068 Koblenz

Supervisors:	Dr. Andreas Winter	Prof. Dr. Klaus G. Troitzsch
	Institute of Software Engineering	Institut für Wirtschafts- und
	University of Koblenz	Verwaltungsinformatik
		University of Koblenz

Declaration

I declare that this report is my own original work and that no part of it has been submitted to any other institute of learning in support of an application for another award. No other sources than those quoted have been used. The opinions expressed in the report are put forward in a personal capacity and do not represent those of the University of Koblenz or any organisation with which the author may be associated.

Signature................................ Date................................

Urs Kuhlmann

Acknowledgements

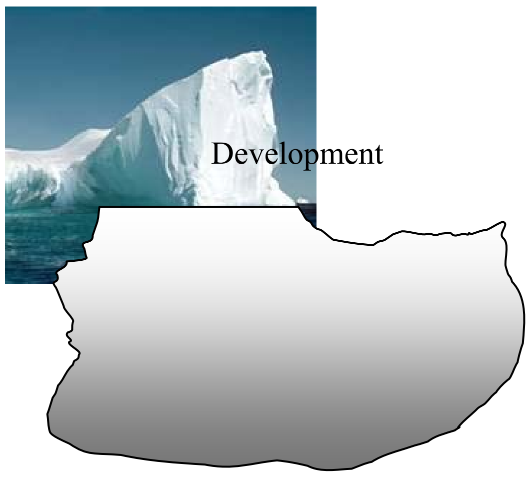
I would like to thank the companies Debeka, IBM, sd&m and Volkswagen AG for the possibility of doing interviews for my thesis.
Furthermore I would like to thank Rainer Gimnich, Eike Escheberg Schröder, Johannes Pellenz, Marion Alfter, Dr. Ulrich Kühler, Buu Tran Van, Franz Klein and Christian Rünz for their time to answer my questions and the information given to me during the interviews.

Abstract

Maintenance cannot be looked at detached from other software development processes as it plays a crucial role in the software life cycle.

In this thesis I have outlined how the perception of maintenance has changed over the years and what is now considered to be important. I have determined maintenance activities in theory and practice in the context of software process models for the use of academics and practitioners. I assumed the role of an interpretive traveller, first looking at different theoretical perspectives, then investigating current practices and finally bringing it together to establish a maintenance workflow within the context of the Rational Unified Process.

Key words: Maintenance, Software Development Process Models, Rational Unified Process



Source: adapted from [Martin and McClure 1983]

Figure 1: The Maintenance Iceberg

Table of Content

List of Figures

1 Introduction

1.1 Motivation

While working for the Institute of Software Engineering, I discussed possible topics for a master thesis with Andreas Winter, my thesis supervisor. As I have a business background, we were looking for a less technical but more organisational topic. After thinking about several alternatives, their core points and possible pitfalls, I decided to write my thesis about Software Maintenance Activities in Software Process Models because I see it as an important area for research for the following reasons.

Ever since mankind starting exploring the oceans, icebergs have been a problem for seamen getting closer to the poles. Most parts of the icebergs are under water and cannot be seen. Software engineers developing software face a similar problem: just as seamen know that the icebergs are there, software engineers know that maintenance will be necessary for the software they develop. However, when approaching maintenance, most parts cannot be “seen” or are not identified while the software is being developed.

Nowadays ship navigators know much more about the activities to perform and the tools to use to avoid collisions. This thesis aims to reduce the risk of being shipwrecked due to maintenance icebergs.

1.2 Research Area Background

Software is going to be changed several times for different reasons while being developed and especially after it has been delivered. Generally the term maintenance is used when referring to those changes made to software products after they have been delivered [Bennett, Cornelius, Munro and Robson 1991]. Depending on the reasons for change and the wider organisational context, various approaches to maintenance such as corrective or adaptive maintenance (cf. 2.3 Categorisation of Maintenance) are or rather should be applied [Pfleeger 1998].

Software maintenance as one aspect of software engineering is addressed in all kinds of general and introductory literature about software engineering [Bennett, Cornelius et al. 1991; Denert 1991; Balzert 2001; Pressman 2001; Sommerville 2001]. Most if not all books about software engineering in general will at least touch on this topic area.

Awareness of the importance of maintenance when developing software has increased over the last decades. Increasingly, software is not developed from scratch but is a further development of existing software [Phillips 2000]. Not only has the definition of maintenance changed (cf. 2.1 Maintenance) but the skills which are seen as essential for software maintainers have also changed.

Several surveys from the 1980s, stating that development and maintenance costs might partly follow the 20/80 rule, indicate that maintaining software might cost two to four times as much as developing it. These surveys are still being quoted in more recent literature such as [Pfleeger 1998; Sousa and Moreira 1998; Balzert 2001; Sommerville 2001].

The IEEE Computer Society established the International Conference on Software Maintenance, which started 1983 and now is held annually. This conference and others, such as the Conference on Software Maintenance & Reengineering and the Working

Conference on Reverse Engineering, are also indicators of the increasing importance of maintenance in the software engineering community. Furthermore, public awareness of maintenance issues is increasing through famous examples such as the 2YK problem (Millennium bug), the Euro conversion [Spinu 2001] and the recent change to IBAN (International Banking Account Number).

The awareness for maintenance as an area within software engineering is increasing. Nevertheless, [Bennett, Cornelius et al. 1991] emphasise that it is important to discuss the activities of maintenance and not the phase itself or just the idea of maintenance. Therefore, this thesis aims to address the maintenance activities which are discussed in literature and applied in practice.

However, the literature is either focused on software development and maintenance on a very generalised and abstract level, or on one category of maintenance and specialised maintenance activities. A detailed but modular description of maintenance activities in the context of software development models such as the Waterfall model or the Rational Unified Process is not available.

With regard to software development models, this thesis focuses on the Rational Unified Process (RUP) as described in [Kruchten 2000], which offers a sophisticated generic software process model. The Rational Unified Process is the direct successor to the Rational Objectory Process which resulted from the integration of the Rational Approach and the Objectory Process. There is no explicit workflow for maintenance but maintenance-related activities are embedded in other workflows such as configuration and change management. Kruchten states that RUP is "*a process framework that can be adapted and extended*" and thus offers the opportunity to include maintenance activities in existing workflows and, if necessary, to define a new workflow for remaining activities.

In the following sections I am going to present the aim and objectives of my research and the methodologies I have applied to achieve these objectives.

1.3 Aim & Objectives

<u>Aim</u>

This thesis aims to investigate maintenance activities in the context of software development process models and their applicability and usage in practice. It will analyse the following research question:

> "Which maintenance activities are applied within different contexts, what is the rationale for practitioners to select these and how are these maintenance activities embedded within the software development process?"

<u>Objectives</u>

My research question consists of three parts, all linked together but all with their own boundaries regarding the overall scope of my thesis. They need to be broken down to specify where the boundaries are to make it manageable.[1]

1. "Which maintenance activities are applied within different contexts?"

<u>Determine maintenance activities through a literature review</u>

Identify and explain what is understood under the term maintenance with regard to software and determine potential categories of maintenance, i.e. different schemata. Based on that, identify and explain what activities, methods and techniques belong to maintenance, their costs and efforts as well as the responsibilities for certain maintenance activities (roles). Tools were added only if they follow specific models or support central activities.

The outer boundary is set by the fact that I am not looking for the one and only truth of what maintenance is or should be but for possible alternatives how maintenance can be approached.

<u>Determine what is practice</u>

Identify and explain through six mini case studies the organisational context and processes in which different maintenance models are applied, which activities and techniques are used and the reasons why those have been chosen by the different organisations.

2. "What is the rationale for practitioners to select these?"

<u>Determine rationale for discrepancies</u>

Identify and explain similarities and differences between (the use of) maintenance activities in theory and practice. Reconcile the different views and try to generalise as much as possible from this small sample.

[1] The three parts do NOT represent the specific order in which the research has been done. This can be found under 1.4 Methodology.

3. “How are these maintenance activities embedded within the software development process?”

Determine how the identified maintenance activities are embedded within software development process models especially the Rational Unified Process

I did not analyse all existing software development process models but examples from general models [Sommerville 1992; Sommerville 2001], also called “software engineering paradigms” [Pressman 2001]. Using the cited sources, I identified the following models which I found to be relevant and sufficient for analysing how maintenance is embedded within different software development process models:

- linear models, such as Waterfall model(s) including the V-model;
- agile models, here: Extreme Programming [Wells 1999];
- incremental models especially the Rational Unified Process.

A description of how the interviewed companies embed their maintenance activities in their software development process is also included.

Establish modular framework of maintenance activities

This is done through conceptual work to establish a modular framework as an artefact as a final outcome. This includes the identification of maintenance activities within existing RUP workflows, the arrangement of those activities and, if required, the set up of an additional workflow.

1.4 Methodology

Research Paradigm

The research question has been used for guidance, not as a formal hypothesis. Thus my research approach is of an exploratory nature.

Stage one – Literature Review

At stage one of my research I started with a secondary data analysis in the form of a literature review. I included general and specialised literature from the field of software engineering. A lot has been written in this field and it was not my objective to cover everything but to identify major maintenance activities and their role in software development process models. Thus my literature review was non-exhaustive.

I excluded detailed estimates about maintenance efforts (broad concepts are included) as well as an evaluation of software tools supporting the maintenance processes.

Success criteria:

I had fulfilled my first objective when most aspects coming up in the literature about software engineering were covered and other ideas from more specialised literature could be inserted in my findings leading to a self-contained pattern. However, I realise that these criteria are rather subjective and different people may come to different conclusions.

Stage two – Interviews

Stage two consisted of qualitative research in the form of six mini case studies. The technique I applied for data gathering is the use of semi-structured interviews.

I interviewed IT staff with similar responsibilities from five companies from different industries about their software maintenance process and activities they apply. This small sample is not representative neither are the results generalise able. The companies were chosen on the basis that they have different industry backgrounds and were found using existing contacts from one of my supervisors and myself.

The purpose of the interviews was to find out which standard software maintenance process companies have and how and why they have developed their process. Furthermore, I asked them to make some comments about an extended IEEE maintenance process and which of the listed maintenance activities from literature they apply, which they do not apply and the reasons for that, if known.

The technique I chose was semi-structured interviews for the following reasons: Some questions needed to be the same for ease of comparison, thus unstructured interviews would not have been useful. However, structured interviews were also not suited as I needed to include open questions because I wanted to find out about their reasons and I did not want to limit them by giving them a catalogue of choices.

Success criteria:

I got an impression that my questions made sense to the interviewees by the quality of the answers I got during the interviews. In addition to that, I had prepared a checklist of aspects I wanted to find out. After the interviews I was able to check that I had covered those points.

Whether the interviews had really been successful could only be determined at a later stage when the data gathered during this stage has been analysed.

Stage three – Comparison

During stage three I assumed the role of an interpretive traveller who tried to describe and interpret the data gathered during the previous stages to come to a richer understanding about the use of maintenance activities in theory and practice [Swatman 2002]. This interpretation consisted of a comparison mainly of maintenance activities identified within the structure of [IEEE 1998] Standard for Software Maintenance and maintenance activities contained in software process models. A further comparison between those theories and the identified practices, being performed after stage four, completed the research.

Success criteria:

Success criteria at this stage are also subjective. Results were measured by: clearly identified and explained differences and similarities and the contribution of previously gathered data to do the comparison. For a validation the interview partners could be asked if they find their processes in the results.

Stage four – Develop framework

The final stage consisted of an inductive data analysis during which I reconciled the different perspectives from the interviews and my literature review. I developed a framework that fit the structure of the RUP, i.e. to existing and, as required, an additional workflow.

Success criteria:

An appropriate way to measure the success of the final stage would be through a validation of the framework by the interviewed or possibly other companies. To limit the scope I only checked if the framework fit my own findings. A thorough validation as well as an extension and improvement of the framework would be an area for further research.

Information Sources

I followed the approach used by Church and te Braake in their article about "The Future of Software Development" [Church and Braake 2002]. They base their forecast of the future on the history of software development. According to them changes in methods and techniques applied indicate a tendency of what that future will be.

In contrast to them I do not try to forecast the future but identify what is currently considered as appropriate with regard to software maintenance.

Sources which were useful for doing that included books about software engineering which have been revised several times over the past few years, indicating a clear development. This helped me to decide which activities are currently perceived to be more important than others.

Seminal literature about software engineering from authors such as Balzert, McDermid and Sommerville were used as well as more specific literature about maintenance, mainly conference proceedings from ICSM (International conference on software maintenance), IWPC (International Workshop on Program Comprehension), WCRE (Working Conference on Reverse Engineering) and CSMR (Conference on Software Maintenance and Reengineering).

Overview

Figure 2 provides on overview of this thesis and the individual chapters. This chapter presented an introduction to this thesis, providing the motivation, aims and objectives and the methodology applied. In the next chapter terms and expressions from the field of maintenance are discussed and then definitions are provided for the context of this thesis. Chapter 3 describes relevant software development process models with special focus on how maintenance is embedded and treated in those models.

In chapter 4 fundamental software maintenance activities are identified and described in detail. References are given to the previous chapter, if and how those activities are embedded in the software development process models. This chapter as well as the two preceding ones are based on literature.

The results of six mini case studies about the maintenance activities of five interviewed companies are provided in chapter 5.

The outcome from the theoretical analysis together with some aspects from the interviews, presented in chapter 6, is a framework for maintenance in the form of an additional workflow for the Rational Unified Process.

This leads to chapter seven which compares the findings from practice with the theoretical approaches identified before. The final chapter offers an outlook about possibilities for further research.

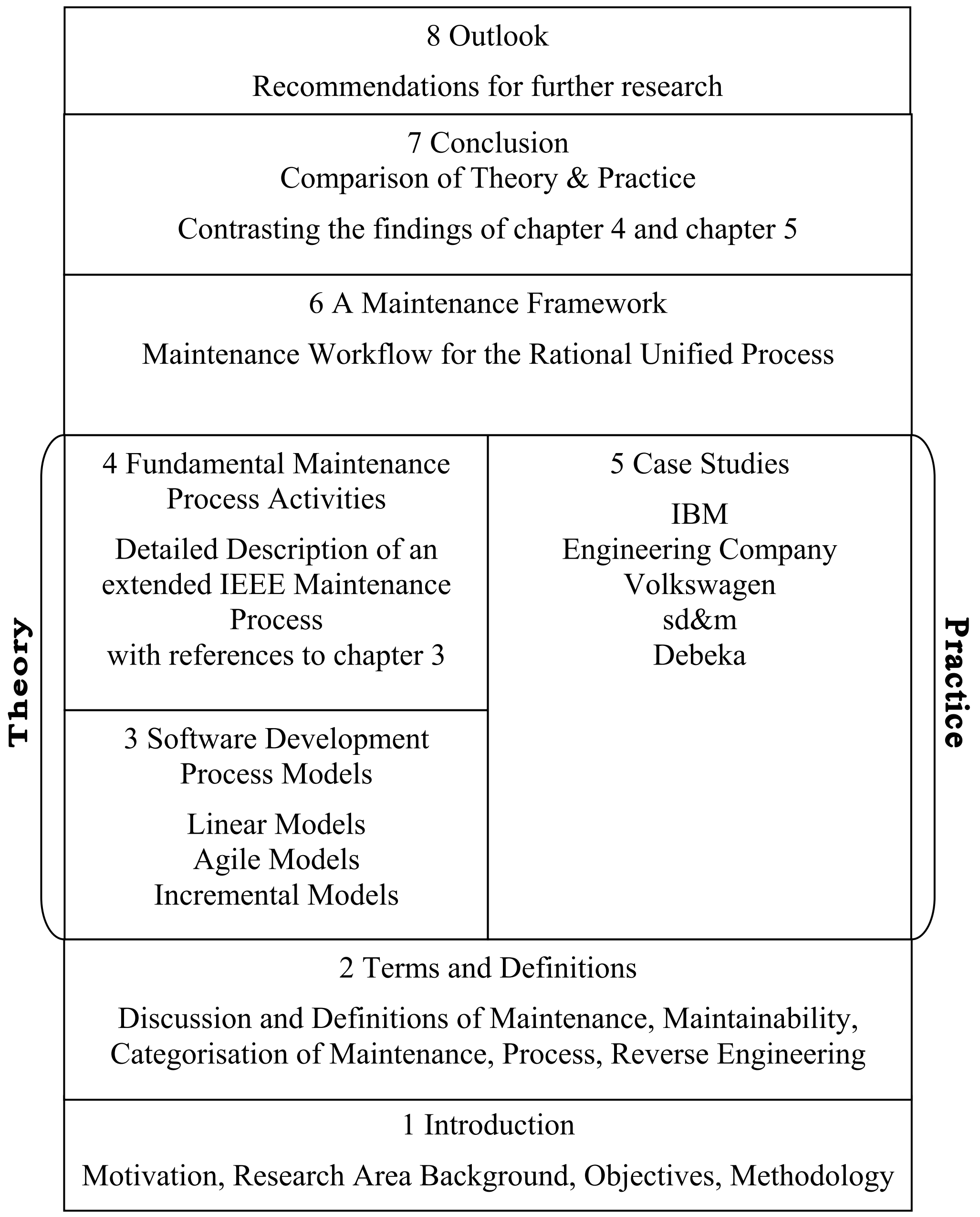

8 Outlook
Recommendations for further research
7 Conclusion
Comparison of Theory & Practice
Contrasting the findings of chapter 4 and chapter 5
6 A Maintenance Framework
Maintenance Workflow for the Rational Unified Process
4 Fundamental Maintenance Process Activities
Detailed Description of an extended IEEE Maintenance Process with references to chapter 3
5 Case Studies
IBM
Engineering Company
Volkswagen
sd&m
Debeka
3 Software Development Process Models
Linear Models
Agile Models
Incremental Models
Theory
Practice
2 Terms and Definitions
Discussion and Definitions of Maintenance, Maintainability, Categorisation of Maintenance, Process, Reverse Engineering
1 Introduction
Motivation, Research Area Background, Objectives, Methodology

Figure 2: Overview of Thesis

2 Terms and Definitions

Although [Phillips 2000] states in his “Software Project Manager’s Handbook” from 2000 that “*little has been written about*” software maintenance, my perception differs from that view. I found quite a lot of articles including academic papers when entering in search engines just keywords in conjunction with maintenance, for example software maintenance, program maintenance, maintenance models or maintenance together with concrete software development process models.

Of course, there are still gaps in research and literature but [Bennett, Cornelius et al. 1991] already stated in 1991 that “*software maintenance has become established as a sub-discipline*” of software engineering, thus leading to the assumption that this acceptance has led to an increase in volume of material being published.

Firstly I present a broad range of perspectives about possible definitions for several terms. However, I have to admit that I did not expect to find so many different views even about the basic terms.

Instead of providing a short overview of terms and definitions with discussions about those terms later on, I will discuss relevant expressions in this chapter leading to definitions which are being used for this thesis. These are not supposed to be generally applicable and true in all cases but valid and appropriate in the context of this thesis.

2.1 Maintenance

Definitions of what I refer to as maintenance can be found under headings such as maintenance [Martin and McClure 1983; Freedman 1995], program maintenance [Illingworth 1983], and software maintenance [Bennett, Cornelius et al. 1991; Sommerville 2001]. [Bennett, Cornelius et al. 1991] also state that the term maintenance per se is not appropriate as it is rather related to hardware than to software but they will use it because this term is “*now well established in the computing profession*”.

According to [IEEE 1998] software maintenance is defined as the “*modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified/changed environment.*”[2]

The emphasis on the after-delivery-phase can be found at most authors but not at all of them. Even though [Illingworth 1983] lists “*after release*” as one aspect of her definition she states that some software process models view that differently. The IEEE definition explicitly states “*after delivery*” but says in the introduction that the standard applies to all maintenance activities with “*maintenance planning ideally beginning during the stage of planning for software development*” [IEEE 1998]. [Kajko-Mattsson 2001] explicitly includes activities of predelivery/prerelease, transition and postdelivery/postrelease stages in her definition of software maintenance. Furthermore, maintenance is not limited to the forward moving process of system development, it certainly contains others methods such as refactoring which do not belong to the category of forward engineering.

[2] For an overview of definitions please cf. [McClure 1992] which includes (early) definitions from Boehm, IEEE, Reutter, Swanson and others.

Also the order in which the aspects of maintenance are stated are relatively consistent throughout most authors with "*correcting errors*" usually coming up first. One exception is [Freedman 1995] who first refers to changing requirements and lists "*fixing bugs*" and "*adapting (...) to new hardware*" as "*also*" belonging to maintenance.

Some authors such as [Martin and McClure 1983; Kusters and Heemstra 2001] compare software maintenance with icebergs. The issue they want to raise is that maintenance costs are quite often hidden "under water", i.e. not properly taken into account when developing software. Similar to icebergs where 90% cannot be seen above the water level, software maintenance costs can make up to 80% of the overall costs throughout the life cycle of a software product. [Sommerville 2001] states that "*it is difficult to find up-to-date figures*" about maintenance efforts spend by large organisations. This explains why even more recent literature such as [Pfleeger 1998; Sousa and Moreira 1998; Balzert 2001; Sommerville 2001] continues to quote rather old surveys from the 1980s which indicate that maintaining software might consume up to 90% of the total expenditure during the life of a software product.

When critically studying those definitions it is difficult to draw a border line between development and maintenance. Some maintenance categories (cf. 2.2 Categorisation of Maintenance) are defined as updates to software "*which blur a clear boundary* (so that) *maintenance is now though of as continued development*" [Phillips 2000]. The issue how maintenance can be differentiated from development and whether this should be done at all is addressed by several authors. [Pfleeger 1998] labels the maintenance stage as "*evolutionary phase*", [Phillips 2000] calls maintenance simply "*continued development.*" According to this perspective, development stops and maintenance starts very early, every time when a software project does not explicitly starts from scratch.

In linear process models, such as Waterfall models, maintenance is clearly seen as a separate phase that follows after development. This differentiation is not that obvious in agile or iterative process models. In those cases maintenance can be seen as an activity that is performed all the time, as a continuous workflow or as a stage/cycle which follows after development. This issue is further discussed in chapter 3.

A comprehensive definition of maintenance need to cover all of the above mentioned areas and has to take into account that maintenance activities are performed before and after the delivery of a software product to the customer.

Definition

Software Maintenance is the planning and execution of activities for the modification of a software product to improve performance or other attributes, to adapt the product to a modified/changed environment, to correct faults or to improve its maintainability during all stages of the software life cycle.

2.2 Maintainability

The key word of "maintainability" appeared in the definition of maintenance. It is also listed as the first key attribute of well designed software by [Sommerville 1992] in the introduction to his book.

[Ramage and Bennett 1998] write that "*maintainability is the ability to be maintained.*" They also quote the IEEE definition of maintainability from 1990: "*The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment.*" This definition again contains all aspects of software maintenance listed above.

[Pfleeger 1998] argues with regard to level of maintainability that it is "*impossible to build systems* (for long-term use) *that do not need any maintenance*" because the environment within which the systems operate are not stable and change and thus the requirements of the systems change.

[Madhav and Sankar 1990] claim that "*formal specifications increase maintainability*" as it makes fault detection easier.

Surprisingly, from all articles listed in my reference list, only [Kajko-Mattsson, Forssander and Olsson 2001] explicitly demand to "*plan for and build in maintainability*" as a "*predelivery phase activity.*" In another paper [Kajko-Mattsson 2001] also calls for "*preserving maintainability*" at a post delivery stage.

In the introduction to his book [Pressman 2001] asserts that computer software succeeds when it is "*easy to use and easy to modify; performs flawlessly over a long period of time.*" All aspects, especially the latter two, can be somehow linked to issues raised before in the context of maintenance.

He continues by saying that "*computer software* (should be build) *like any other successful product by applying a certain process.*" With regard to maintainability I interpret this that sometimes you build a (software) product to be easy to maintain, sometimes you neglect this issue. Certainly software developers have to plan for maintainability because a system "*cannot have maintainability retrofitted to if later*" [Martin and McClure 1983].

Clearly a definition of maintainability needs to be closely link to the term maintenance. The demand to plan for maintainability should be kept in mind but is not part of a definition.

Definition

Maintainability is the ease or simplicity with which a software system can be maintained (using the definition of software maintenance above) and is a key characteristic of software.

2.3 Categorisation of Maintenance

Some authors such as [Phillips 2000; Balzert 2001] explain or define maintenance by listing possible categories of maintenance and defining those. Examples for such maintenance categories are: corrective maintenance, e.g. the removal of errors in the code; adaptive maintenance, e.g. the adjustment of software for a new operating system; and perfective maintenance, e.g. the modification of the code to improve performance.

Although the above mentioned authors agree on the number and names of the categories, they differ with regard to the definition and explanations of some of them. Due to the naming problem outlined below [Sommerville 2001] writes that he "*avoided the use all these terms*" although he used them in a previous edition of his book [Sommerville 1992].

To demonstrate the variety and the related naming problem I will briefly describe the different definitions and explanations. Where a general agreement about certain terms or their core meaning exists, I will continue using them. Otherwise I will state what I mean, thus avoiding the necessity to stick to a certain definition which might lead to some confusion when quoting certain authors.

[Balzert 2001] emphasises the importance to differentiate between the German terms of "Wartung" and "Pflege" which are both translated into English as maintenance. Generally according to the above mentioned source it can be said that Wartung stands for correcting faults, whereas Pflege is concerned with product enhancements. Depending on the point of view the author argues that Wartung, i.e. correcting faults, can be seen as a final activity of development.

This is already one approach of categorising maintenance. Other categories of maintenance listed by [Sommerville 2001] are "*corrective, adaptive and perfective maintenance*". Often these categories are complemented by a fourth category which is called preventive maintenance. Authors using the four categories include [Bennett, Cornelius et al. 1991; Sousa and Moreira 1998; Phillips 2000; Balzert 2001].

The above mentioned authors using the four categories of maintenance all define "*corrective maintenance*" as concerned with "*correcting faults*" mostly after but also before delivery.

[IEEE 1998] also list emergency maintenance as a special form of corrective maintenance as unscheduled action "*to keep a system operational*".

In the context of corrective maintenance [Ebert 2002] stresses the difference between failure/error, which is a deviation from expected system behaviour, and fault/bug, which refers to the reason for occurring errors.

[Bennett, Cornelius et al. 1991; Sousa and Moreira 1998; Sommerville 2001] characterise "adaptive maintenance" as a situation when it is necessary to adapt software to changes in the operating system or hardware. In contrast to that definition are [Phillips 2000] and [IEEE 1998] who also include adaptations to software due to a "*changing situation such as new regulations or business opportunities.*"

Others include that under "perfective maintenance", which is described as "*expanding requirements, improving efficiency and partly increasing maintainability.*" The latter – increasing maintainability – is put under the heading of "preventive maintenance" by

[Bennett, Cornelius et al. 1991]. Also contained under that term are activities to "*prevent malfunctions in the future.*"

Depending on the respective definitions different authors come to various results regarding the distribution of maintenance efforts. This can be seen in figure 3 where e.g. according to one definition adaptive maintenance accounts for 70% and according to a different definition it only accounts for 25%.

[Martin and McClure 1983] also list as an alternative another seven categories which are "*emergency repairs (immediate repairs necessary), corrective coding (correctly reflect the specifications), upgrades (changes in processing requirements), changes in conditions (business conditions), growth (data requirements), enhancements (user request), support.*" All these aspects can be included in one of the above mentioned four categories of maintenance.

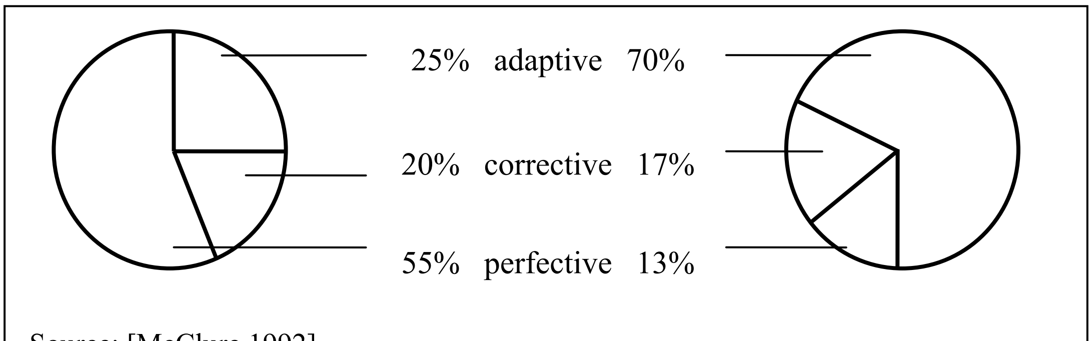
There are probably even more ways how to categorise maintenance depending on the context. For the purpose of this thesis this overview should be sufficient. The following definitions are based on the most common denominator of the above listed explanations. They draw boundaries between different areas of maintenance with the smallest number of useful definitions for this context.

Definition

Corrective Maintenance: Event driven, reactive and partly unscheduled modification of a software product to correct discovered faults to keep a system operational.

Adaptive Maintenance: Event driven modification of the software product due to changed or changing environment or requirements

Perfective & Preventive Maintenance: Quality driven modification of the software product to improve efficiency, performance and maintainability and preventing problems in the future.



Source: [McClure 1992]

Figure 3: Distribution of Maintenance Efforts

2.4 Process

Due to the fact that the maintenance **process** is the centre of the thesis it is important to determine what characteristics such a process posses and which need to be described to differentiate it from others.

[Sommerville 1992] defines a process simply as a set of activities. The definition of [Turk and Vaishnavi 2002] goes further by saying that this set of activities needs to be in a partially-ordered sequence and the activities are carried out in order to accomplish a set of certain goals.

The attributes of a process which are used by the IEEE standards committee are input, process, output and control as well as associated processes [IEEE 1998].

In the Rational Unified Process the activities are organised into workflows which are defined as "*a sequence of activities that produce a result*" [Kruchten 2000]. It is important to note that a workflow is not identical with a process because a process does not necessarily represent a sequence.

I will use the term **workflow** in the context of RUP, **process** for tasks which will be broken down further in contrast to **activities** which denote the lowest level for sufficient representation in the context of my thesis.

Definition

An **activity** is any task which will not be broken down any further.

A **process** is a meaningful and partially-ordered set of activities that is carried out to produce a valuable result.

The following structure/characteristics as illustrated in figure 4 are used for describing processes (and activities) in the next chapters:

- Purpose: why this process is needed;
- Input: artefacts needed for the process;
- Process (set of activities): activities that need to be performed to produce the output;
- Output: valuable result of the activity/activities;
- Role: responsibility for the activities and therefore for the output.

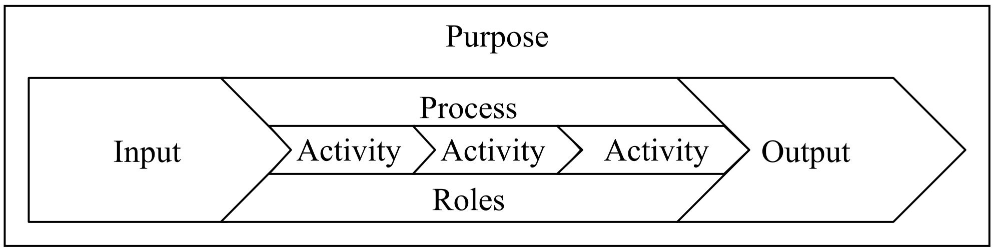


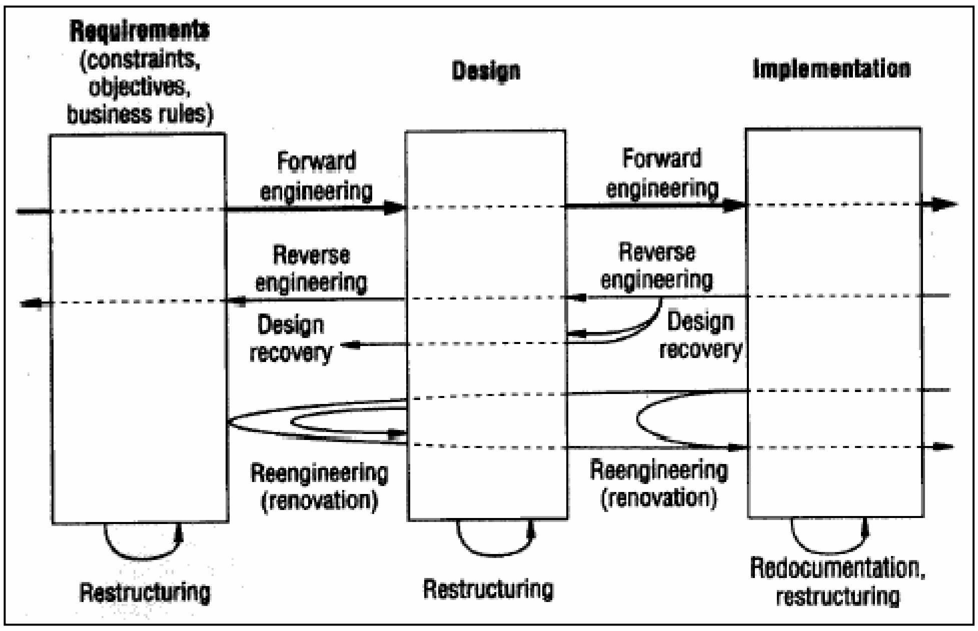
Figure 4: Process

2.5 Reverse Engineering / Reengineering

Maintenance, maintainability and reengineering are often mentioned in a similar or in the same context, for example it is said that reengineering improves maintainability. However, they are not synonymous. Reverse and reengineering activities support and are part of the maintenance process but are more limited regarding its purpose.

[Arnold 1993] defines the purpose of reengineering as the improvement of program understanding, i.e. reverse engineering, and the improvement of the software itself, i.e. forward engineering, for increased maintainability, reusability or evolvability.

Within the [IEEE 1998] Standard of Software Maintenance it is said that reengineering consists of reverse engineering and forward engineering. The relationships between forward engineering, reverse engineering and reengineering as defined by [Chikofsky and Cross 1990] can be seen in figure 5. Forward engineering is the traditional forward moving process of system building, moving e.g. from requirements via design to implementation. Reverse engineering, as the name suggests, is an examination process in the diverse direction to identify and present (parts of) the system. Design recovery is an example where the process starts at the implementation level to recover the design, i.e. moving backwards to the design level. According to their taxonomy of reverse engineering this process can start at any level of abstraction as an existing functional system is not a prerequisite.



Source: [Chikofsky and Cross 1990]

Figure 5: Terms of Reengineering

The [IEEE 1998] definition continues that reverse engineering does not change the system but provides "*an alternate view of the system at a different level of abstraction.*" [McClure 1992] adds that this view is recovered from the system's lower level form, for example redocumenting code as structure charts or flow diagrams. The purpose is to reconstruct a description of its components and their interrelationships as well as other design information to improve program understandability so that the system can be re-implemented or maintained more easily [Bennett, Cornelius et al. 1991; Bennett and Rajlich 2000].

The purpose of improving a system's future maintainability can also be found as a key driver of reengineering. The opinions in literature differ if reengineering only changes the HOW or also the WHAT the system does. [Sommerville 2001] states that no new functionality is added during the reengineering process in contrast to a maintenance process whereas [Chikofsky and Cross 1990] suggest that reengineering examines and alters a system which may include modifications due to new requirements. However, most of the cited authors explicitly say that reengineering is an umbrella technology where several disciplines meet.

The perception of maintenance does not only differ between various authors. Changes in the perception of the importance of reengineering can also be found for example in [Sommerville 1992] and [Sommerville 2001]. In the fourth edition from 1992 reengineering was included in the chapter about maintenance, in the sixth edition from 2001 a dedicated chapter about reengineering is included.

An additional but also important issue is raised by [McClure 1992] who writes that reengineering requires a long term strategy and is not suited for immediate short term changes due to its efforts and costs. A long term strategy, however, involves a rationale, which means that not the particular activities or tools are important to define reengineering but the underlying purpose.

Definition

Reverse Engineering is a process with the purpose to improve the system's comprehensibility. The inputs are existing artefacts which are analysed to recover a description/representation of a system's components and their interrelationships. The results of the process are artefacts at a different level of abstraction than before.

Reengineering is a process with the purpose to implement a system in a new form to improve its future maintainability and extend its life expectancy. The inputs are existing artefacts at any level of abstraction. The process consists of the examination (reverse engineering) and alteration (forward engineering) of a system and the result is a system in a new form where the new form can have new functions or not.

3 Software Development Process Models

[Sommerville 2001] summarise the software process as all activities involved in software development and maintenance. The view of [Turk and Vaishnavi 2002] that processes are a set of activities in a partially-ordered sequence is similar to that of [Bennett, Cornelius et al. 1991] who also state that a software process model "*should represent attributes of a range of particular software processes and be sufficiently specific to allow reasoning about them.*"

This only states the general concept of software process models. With regard to maintenance [Bennett, Cornelius et al. 1991] write that traditionally maintenance in comparison to other stages was regarded as rather detached than belonging to the software life-cycle. The software life-cycle itself is defined by [Illingworth 1983] as "*the complete lifetime of a software system from initial conception through to final obsolescence*" who continues that the traditional stages are "*system requirements, software requirements, overall design, detailed design, component production, component testing, integration and system testing, release, operation and maintenance.*"

The dictionary of computing by [Illingworth 1983] also notes that some models of the software life cycle view "*program maintenance as simply being iteration occurring after rather than before release of the system for operational use.*" This leads to the issue how different models incorporate the aspect of maintenance.

I do not want to cover what software process models exist but the different ways in which maintenance is embedded in those models. Thus I am looking for categories which can be used to demonstrate how maintenance is treated differently by different models.

Although every author seems to have his or her own view on how to group software process models and which are worth mentioning the following general models or engineering paradigms can be identified [Balzert 2001; Pressman 2001; Sommerville 2001] and will be used in the following sections:

- linear models – Waterfall model and V-model;
- agile models – Extreme Programming;
- incremental models – Spiral Model, Evolutionary Model, Rational Unified Process.

In the following I am going to outline the above listed models with a special focus on the maintenance phase or function. As the outcome/framework at the end of this thesis depends on the identification of maintenance activities in the Rational Unified Process, this model will be described in more detail than the others.

3.1 Sequential, linear Models

The software process can be broken down into smaller parts. The smallest units, depending on the perspective, are individual activities. In sequential, linear models these activities are summarised to stages which are carried out in a strong linear sequence. That means that one stage needs to be completely finished before the next stage can start. Two examples of linear models are the Waterfall model and the V-model which are outline in this section.

Waterfall Model

Purpose

In his article about "The Conventional Software Life-cycle Model" [Agresti 1986] writes that the Waterfall model is an early attempt from 1970 to structure software development. Software should be developed by a sequence of general activities following the underlying assumption that successful software was developed by successively achieving subgoals at each phase, the corresponding milestones.

Input

All of the described software development process models have more or less the same input to the process; that is requirements for the software product. The differences regarding input are when what kind of information is fed into the process. This is a process characteristic and is therefore described under process.

Process

In the Waterfall model, sometimes called classic life cycle model, all of the above mentioned traditional stages are processed sequentially, one after the other. Figure 6 shows one possible form of the Waterfall model.

– Analysis: All requirements of the system and the software are gathered – WHAT the software is supposed to do.
– Design: Translation of requirements into a representation of the software – HOW the software will meet those requirements.
– Implementation: The code is developed.
– Testing: Tests are conducted to ensure a system *as required*.
– Integration: System is integrated into is operating environment.
– Operations & Maintenance: System is running and needs to be supported and maintained.

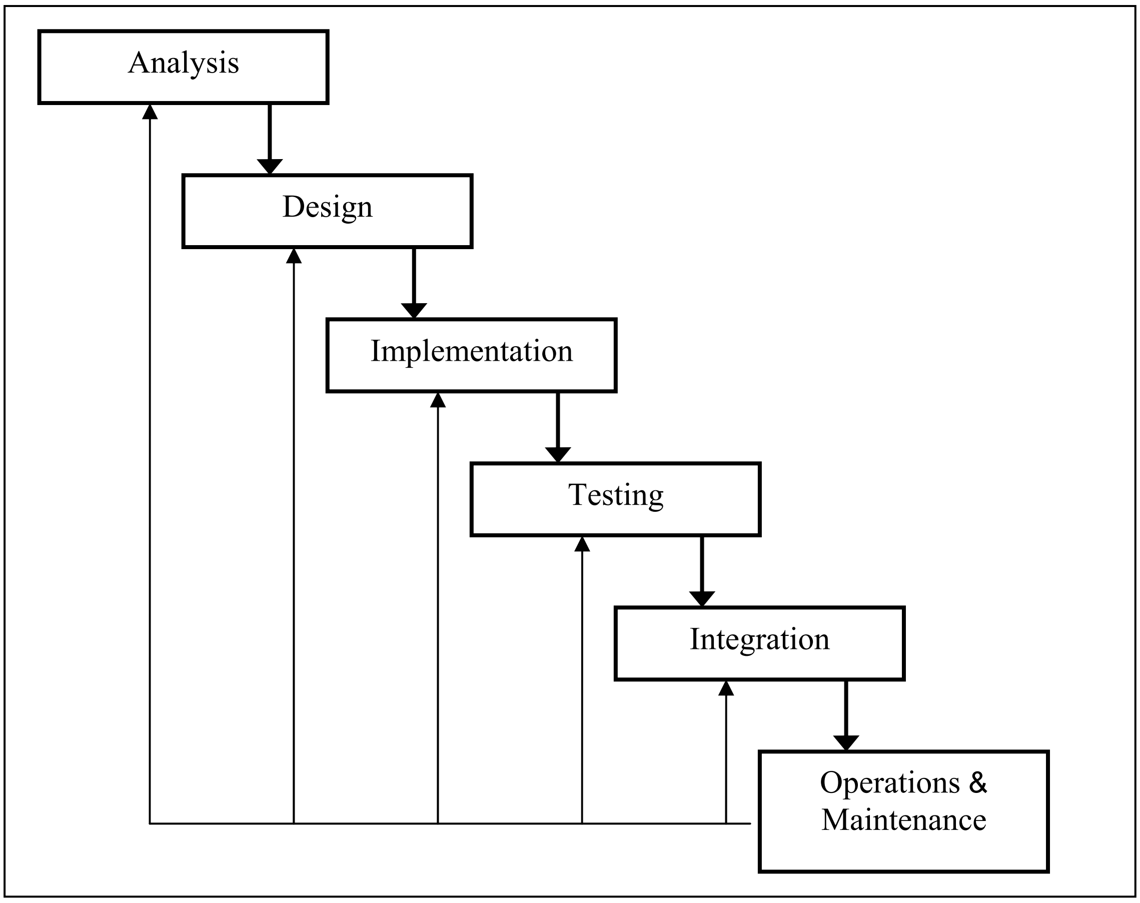


Figure 6: Waterfall Model

This model starts at a system level where relevant domain information is gathered. That means all requirements of the system need to be fixed at an early stage [Pressman 2001]. The output of every phase is the required input for the next phase. A major criticism of this procedure is that all artefacts of one stages need to be fixed before one can proceed to the next. One example for this problem is the separation of "*the WHAT of specification from the HOW of design*" [Agresti 1986], saying that it is difficult or even impossible to prepare detailed specifications without knowing how it can be done of if it can be done at all.

Output

The output of one software development cycle is an operational system with all required functions and complete documentation. The final output, depending on the perspective, is a discarded system which cannot be maintained and/or operated anymore.

Roles

Employees with all the skills needed for a software project play a role in the Waterfall model but not all at the same time. The strong sequence of activities can lead to blocking states, i.e. due to dependencies some team members often have to wait for input from other team members. "*In fact the time spent waiting can exceed the time spent on productive work*" [Pressman 2001].

Maintenance in the Waterfall Model

Maintenance is explicitly mentioned in this model in the form of a maintenance phase. As suggested before the maintenance stage is seen as a stage at the end of the life cycle, often put together to operation, maintenance and support. Nevertheless, all authors admit or add that during this last stage the activities of the preceding phases are reapplied to the software product again. A similar alternative is provided by [Sommerville 2001] who draws a classic software life cycle where feedback is given from the operation and maintenance stage to earlier phases which may lead to going through all phases again.

However, maintenance is not well supported in the sense that it is stated what to do and how to do it; a separation into maintenance categories is not mentioned at all. The Waterfall model simply allows for maintenance-related activities to be added to the last phase, whatever a user of this development paradigm might want to use. This way the model does not take into account the importance of maintenance or its proportion of the life cycle.

Furthermore some authors argue that the strict application of a linear model will reduce the maintainability because requirements and thus specifications are fixed early and often programmers avoid going back to make necessary changes. Afterwards they find ways to work around it during implementation.

V-Model

Purpose

The V-model is a software development process model developed by the German Federal Ministry of Defence in cooperation with the Federal Office for Defence Technology and Procurement in Koblenz since 1986 [BMVg 1997]. It is the official standard for software development in the Ministry of Defence since 1991 and in the Ministry of the Interior since 1992. The current version is from 1997.

The V-model was developed with the focus on the following applications:

- basis for contracts;
- instruction for system development with detailed descriptions of the activities and documents;
- communication basis;
- with the feature of universal validity and project independence.

[Agresti 1986] stresses the emphasis on verification and validation[3] at each phase of the V-model. This is the major difference to the Waterfall model. The V-model can be seen as a bent Waterfall model, enabling feedback to corresponding phases during the process. However, the V-model still belongs to the category of linear sequential models because one phase needs to be completed before the process moves on to the next phase.

Input

In addition to the needs of the organisation and the users, which are transformed into requirements for the software product, a major aspect regarding the input is the detailed specification of the activities.

Process

The well-known V-form of the V-model can be seen in the submodel of system development. The other submodels are Quality Assurance, Configuration Management and Project Management. A detailed description of these submodels can be found at [BMVg 1997].

The nine phases of System Development, as illustrated in figure 7, are:

- SD1: System Requirements Analysis: Requirements are analysed, a description for the system and its environment is established and a risk analysis is realised.
- SD2: System Design: The system is divided into segments.
- SD3: Software/Hardware Analysis: Technical requirements are described in more detail.
- SD4: Software Design (rough): The software architecture is designed.
- SD5: Software Design (detailed): The detailed design is specified.
- SD6: Implementation: Software components are coded.
- SD7: Software Integration: Software components are verified.

[3] [Sommerville 1992] explains those terms as follows: Validation means building the right product, the what; and Verification is building the product right, the how.

- SD8: System Integration: Components are integrated and validated; the components are integrated into segments and the segments are afterwards integrated into the system.
- SD9: Transition to Utilisation: The finished system is installed at the point of operation.

In addition to early fixed requirements and a linear sequence of activities, test cases and scenarios are important inputs for later phases which are used for verification and validation of the implementation and the system requirements in the second half of the model.

Output

The final output is an operational system with all functions and a complete and detailed documentation.

Roles

The main roles in the V-model are project leader, quality assurance manager, configuration management representative and controller. Other roles are the usual candidates like system analysts, programmers, etc. who are in involved in software development.

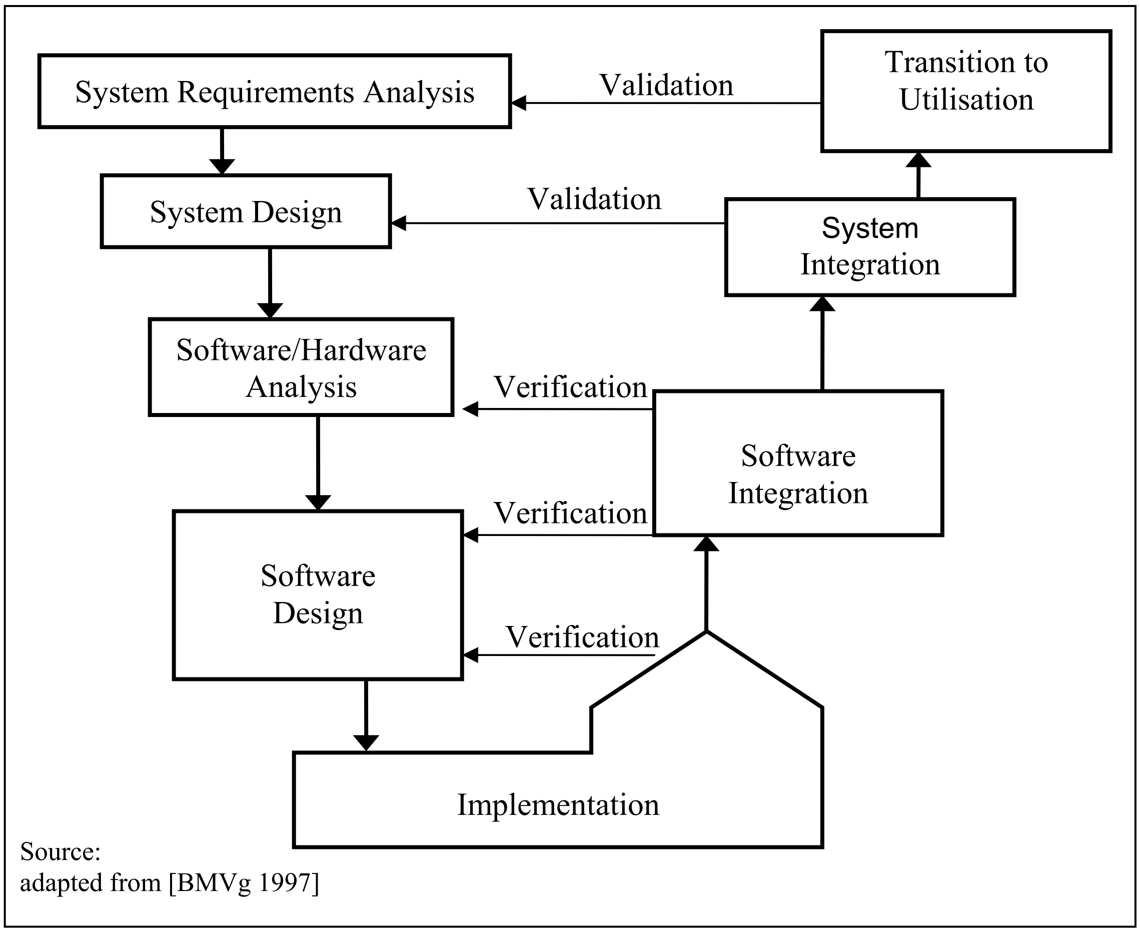


Figure 7: V-Model

Maintenance in the V-Model

[BMVg 1997] names the following advantages of the V-model with regard to maintenance:

– reduction in the number of maintenance cases as a result of improved product quality;
– decrease in maintenance effort via easy comprehensibility and the existence of an adequate software documentation.

It is right to say that the V-model comprises all necessary activities to make use of the above listed advantages. However, it is a complex model and only suited for large projects. Furthermore it is not always possible to transfer the ideas of a perfect world model to the real world.

In contrast to the previously described Waterfall model, maintenance is not explicitly embedded in this model; it only comes implicitly after transition to utilisation. What is said about maintenance is that maintenance itself is reduced through proper development. Activities for maintenance after product delivery are not described. Thus the V-model is only suited to reduce the need for maintenance but not for maintenance projects.

3.2 Agile Models

Extreme Programming

Purpose

Extreme Programming (XP) was developed in the late 1990s as a new style or methodology of software development. It is claimed to be well suited for small to medium sized teams and to be a method that offers simplicity and flexibility, early and constant feedback, low risk but is also predictable and lightweight in a vague or rapidly changing environment.

[Wells 2003] describes Extreme Programming as a new tool that delivers incremental releases. It is listed separately as it has some characteristics that make it different from the classic incremental approaches (cf. 3.3 Incremental Models).

It is not appropriate for projects that need extensive documentation, e.g. for audits or other controlling functions, for very large teams due to communication problems or if the requirements are well understood.

This chapter is based on [Beck 2000] unless stated otherwise.

Input

The input is always a simple plan for a short planning horizon. All requirements for the system are established over time, not at the beginning of development.

Process

The traditional (and proven perspective) is that the cost of change rises exponentially over time. Every incorrect decision will be more costly the later it is discovered in the process. Enormous resources have been spent for better technology, better tools and practices to reduce cost of change. The assumption, as shown in figure 8, is that these investments have paid off and the cost of change rise slowly over time. This leads to completely different behaviour, i.e. an XP process, where requirements are fixed late in process to reduce the need for estimates and therefore to increase the chances to be right.

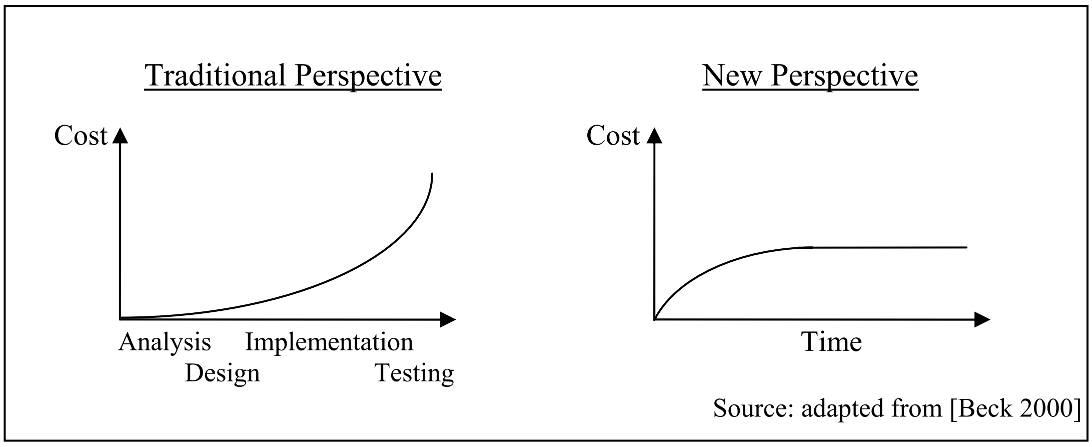


Figure 8: Cost of Change

The main input for XP development projects is established during the so called Planning Game which combines business priorities and technical estimates. User stories, consisting of plain text sentences, are written on story cards which are turned into tasks and are written on task cards during planning.

The input is used for short iterations of a length of one to three weeks. The release planning is user driven, i.e. what the users consider as most important is included in the next iteration. Everything else has to wait.

Every day starts with daily stand-up meetings with representative of all stakeholders (cf. Roles). The daily programming is done via pair programming following certain coding standards.

A key factor is continuous integration, i.e. code is integrated and tested after a few hours. Especially the testing, which provides constant feedback, is important. At the end of the day there is always a defect-free product (at least one that passes all existing tests).

Output

Every iteration provides a new release that can be given to the customer. Only when all user stories are implemented and the customer cannot think of anything else to add to the system is the project over and the final product can be released. Nevertheless, the final output will be the same as the output of most development processes, and that is a system that cannot be developed any further economically.

Role

XP teams consist of two to ten people [Wells 2003]. These team members include a manager who is the team leader, programmers and always a customer representative who needs to be a real customer, i.e. a future user, to answer questions and set priorities. Two programmers always work together on one computer, called pair programming.

Maintenance in Extreme Programming

XP is a rather new approach and not much has been written about it with regard to maintenance. Maintenance does not belong to the practices of XP and is not explicitly embedded in the process. However, it is contained somehow because it is interesting to note that [Beck 2000] says that "*maintenance is really the normal state of an Extreme Programming project*" as maintenance contains naturally many characteristics of Extreme Programming like user driven, quick responses through small increments. Nevertheless, it is not regarded as code and fix but as a proper engineering method.

Furthermore the consulting company ClearStream argues that software developed through XP is easier to maintain as it is "*seen by many eyes, understood by many brains, verified by many tests*" [ClearStream 2001].

One of the practices of XP that supports that statement is testing. This practice ensures that occurring errors, at least most of them, are discovered quickly. Corrective maintenance takes place immediately and continuously to prevent or reduce the need for that category of maintenance after the product has been delivered to the customer.

Another practice, which ensures that one can continue maintaining a software product easily in the future, is refactoring. From a maintenance perspective it belongs to the category of perfective & preventive maintenance as it is described as "*restructuring the system without changing its behaviour*" [Beck 2000] with the objective to *"improve code maintainability and stability"* [Poole and Huisman 2001].

Nevertheless, in the same article [Poole and Huisman 2001] state that if the initial development does not follow XP practices, especially the application of coding standards and simple design, refactoring according to XP cannot be utilised. In those cases a "*wholesale reengineering*" effort is necessary before refactoring and "*extreme maintenance*" can be applied to the software product.

3.3 Incremental Models

Software products often require iterations and cannot be produced in one big step. According to [Pressman 2001] incremental models including the rational unified process are iterative by nature, i.e. the output of each iteration is one increment of the product although not necessarily an operational product. [Phillips 2000] notes that incremental models are “*a repackage of the basic process*” which can be found in all of the following models.

Spiral Model

Purpose

The Spiral Model is a meta-model or process of processes which is well suited for high risk and/or very large projects. It stresses the evaluation of alternatives during every iteration and thus requires good communication to come to an optimal decision.

Input

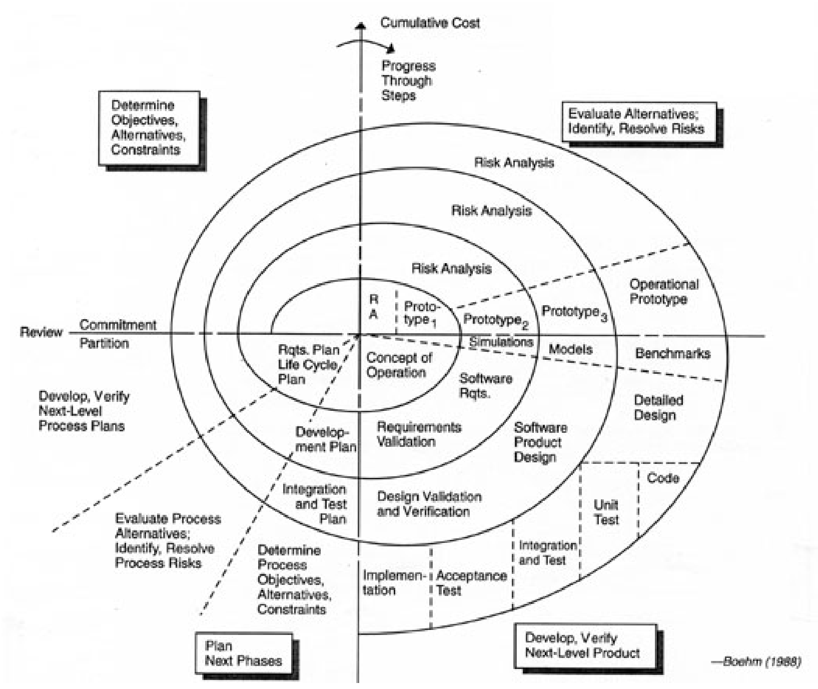
The input for the spiral model is the need of the organisation for the system, transformed into requirements.

Process

The development is done in several cycles which are divided into quadrants as can be seen in figure 9. Every cycle starts with determining objectives and alternatives. Furthermore every cycle, with the exception of the very first one, is supplied with a decision how to proceed from the last quadrant of the previous cycle.

Each cycle consists of four quadrants. At the end of every quadrant a decision is made and/or an artefact is produced. These four quadrants are:

1. Identify objectives, constraints and alternatives
2. Evaluate alternatives
3. Implement chosen alternative
4. Evaluate product and plan the next cycle



Source: [Wideman 2003]

Figure 9: The Spiral Model

The outputs of the respective cycles are similar to those of the classic Waterfall model phases. That means that the first cycle might produce a requirements plan, the second cycle a development plan, then an integration plan until the last cycle produces the real software product.

An alternative view is that the first cycle provides the requirements; afterwards more and more sophisticated versions of the product are developed.

Output

Again two perspectives with regard to output are possible. The first one sees an operational system including the documentation as the output. The alternative applies the spiral model to the life cycle of the product and therefore the final output is a system which is not being used or maintained anymore.

Role

As different activities are performed during various cycles the abilities of the people needed vary from quadrant to quadrant and from cycle to cycle. From the beginning of the project to the release of an operational product all “software development roles” are required.

Maintenance in the Spiral Model

The spiral model can be used for maintenance projects although maintenance is not explicitly contained but it can be inserted into the model.

If the product of the last cycle is an operational product developed through evaluation of alternatives and step-by-step progression through the development process, in that case there is no maintenance in this model. According to this perspective maintenance comes after the last development cycle of this model, i.e. outside this model.

On the other hand it is possible to represent the life cycle of a software product in a spiral model. Every cycle after the first release belongs to maintenance where the development activities are applied again but now as cycles under the heading of maintenance [Sommerville 2001]. In this case maintenance is embedded in the model. Whatever the perspective is, the difference between these two alternatives is a question of a narrow definition.

Evolutionary Model

Purpose

The Evolutionary Model is a process model designed "*to accommodate a product that evolves over time*" [Pressman 2001]. It is needed due to ever more complex systems, existing time pressure when developing products, and constantly changing business requirements. Furthermore it supports development when users do not know what they want right from the start of the project.

Input

[Phillips 2000] suggests two versions of the evolutionary incremental model with distinct inputs being the main difference. The first option, fixed requirements, is that all requirements are gathered and the high level design is fixed at the beginning of the project. Only the implementation and the delivery occur step by step, i.e. in increments.

The alternative, incremental requirements, is that changed and/or additional requirements are collected at the start of every iteration and therefore the high level design is adapted according to the current requirements. Of course, the implementation and delivery has to occur in increments. Obviously these two alternatives regarding input lead to different processes.

Process

The activities of this process are those of a sequential, linear process for every increment. It can be said that the overall process consists of lots of linearly developed increments. Delivering products in increments mean that product functionality is added a little at a time where an operational product is released with every increment.

The above mentioned variations have an impact on the process line when represented in the V-model form (cf. figure 10). The line represents the course of the project with one V illustrating one iteration. In both cases the first iteration follows the linear process where all activities are performed. In case of fixed requirements there is a breakpoint (marked with ◆) where the requirements are fixed. During the next iteration the process line drops straight down after the first delivery. That illustrates that the original two phases requirements and design are not needed again because the requirements and the high level design have been fixed. The process continues at a later phase, shown in the diagram with the continued diagonal line in the V-form.

In the second case of incremental requirements all process activities are performed again and therefore the V-form remains the same for all iterations.

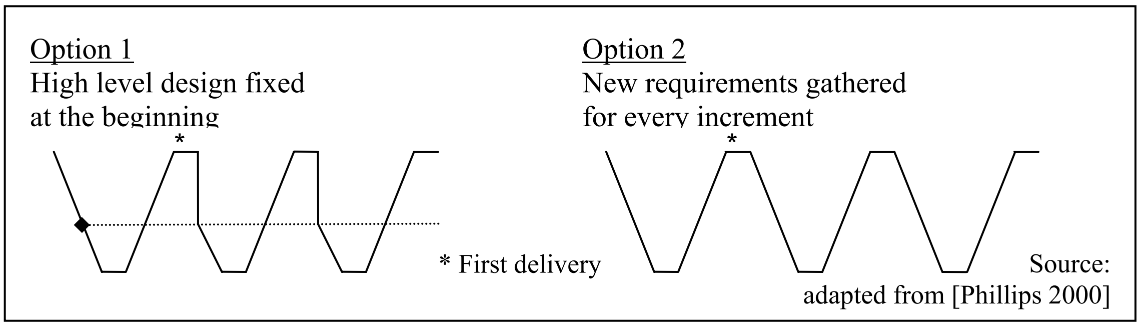

Option 1
High level design fixed at the beginning
*
Option 2
New requirements gathered for every increment
*
* First delivery
Source: adapted from [Phillips 2000]

Figure 10: Variations of the Evolutionary Model

Output

The output of alternative one at the end of the process is an operational product according to the original specifications. Alternative two provides the option that new requirements can contain maintenance requirements until the product cannot be maintained economically anymore. In this case final output is a discarded system.

Role

All roles are needed in every iteration. What is special in this process is the continuous communication. The developers have to listen to the customer, build a product, listen again, build something else, etc. Also the developers have the chance to learn how to do things.

Maintenance in the Evolutionary Model

Maintenance is not explicitly embedded in the evolutionary process model. Depending on the definition, the starting point of the second iteration can be considered as maintenance because a software (core) product exists which is going to be modified. The activities which are performed are still labelled development but need to be related to maintenance, for example activities like impact analysis need to be applied.

Rational Unified Process

Purpose

RUP claims to combine best practice of software development, to offer a well defined process but to be nevertheless flexible and adaptable and therefore suited for projects of all sizes and organisational forms.

[Kruchten 2000], the lead architect of the Rational Unified Process, asserts that its application ensures the production of quality systems in a repeatable and predictable way by assigning tasks and responsibilities, specifying the artefacts to be developed and offering criteria for monitoring progress and performance.

RUP provides a systematic way to produce and validate a system or software architecture around which the design activities are centred. One aspect that supports this is the tight linkage between RUP and the Unified Modelling Language (UML).

This section summarises the key aspects of RUP with the purpose to identify maintenance-related activities.

Input

The major input for RUP is a need of users or an organisation which is transformed into the requirements for the system, expressed through use cases which define the supposed behaviour of the system. These use cases drive the development of the system and serve as the "*foundation for the rest of the development process*" [Kruchten 2000]. They are also the centre view point in the RUP view model of architecture.

Another important input for the development process is detailed specifications for activities which shall or should be performed, depending on the adaptation of RUP to the organisation.

Process

The life of a software product is composed of several cycles during which the product is developed and modified. Every single cycle consists of four phases with varying lengths. The four phases are:

Inception: The objective of this phase is to achieve concurrence among all stakeholders on the life cycle objectives by formulating the scope of the project, planning and preparing the business case. The focus is on understanding the overall requirements. It ends with the Life Cycle Objective milestone[4].

Elaboration: In the elaboration phase the goal is to establish the architecture, specify features and plan necessary activities and resources. The focus is on requirements. It is the most critical of the four phases with the decision to proceed or to abort the project. This phase is concluded by the Life Cycle Architecture milestone.

Construction: The purpose of this phase is to build the product by developing, integrating and testing all components and application features. The focus is on design and implementation. The outcome is a product which can be given to the customer. This leads to the Initial Operational Capability milestone.

[4] Descriptions of the respective milestones can be found in the following paragraphs.

Transition: In the final phase the aim is the roll-out of the final product. The focus is on ensuring that the system meets its objectives. The transition phase is concluded by the Product Release milestone which also concludes the cycle.

Each phase consists of one or several iterations. These iterations follow a similar pattern as the phases of a linear model which are not applied sequentially but in parallel. But from one iteration or one phase to the next the emphasis on the activities, such as analysis and testing, varies.

The emphasis on these activities will vary from project to project but certain milestones at the end of every of the above described RUP phases should be observed. They serve as reference points for controlling the process otherwise cost and schedule estimates are obsolete. They are based on findings that "*success correlates to the degree to which the projects employ equivalents of three critical milestones*" [Boehm 1996].

The following definitions of the milestones are taken from [Boehm 1996]'s article "Anchoring the Software Process" where he suggests three (plus one) milestones as the basis for software development processes. If one of these milestones is not met the project's success is at risk. [Kruchten 2000] uses these milestones for RUP as well to gain control about the process from a managerial perspective.

When reaching the milestones the content/key elements are as follows:

1. Life Cycle Objective (LCO): establishes top level system objectives and provides an operations concept (primary use case), system requirements (stakeholder concurrence on essential features), and a life cycle plan according to the WWWWWHH principle (*Why* is the system being developed? *What* will be done by *when*? *Who* is responsible for a function? *Where* are they organisationally located? *How* will the job be done technically and managerially? *How* much of each resource is needed?). Important to achieve is the feasibility rational which ensures the conceptual integrity and compatibility of all the milestone's components. An additional document supporting the LCO is the initial project glossary.
2. Life Cycle Architecture (LCA): mainly constitutes of elaborations of the LCO elements. Most critical is a definition of the system and software architecture. This is complemented by a documentation of the system's requirements and environment and a use case model which is more than 80% complete.
3. Initial Operational Capability (IOC): has the key elements of software preparation including operational software with documentation, site preparation, and user & maintainer operation selection including team building and preparation.
4. Product Release (PR): contains a complete operational program with documentation for this cycle, an evaluation whether the objectives have been met, an assessment of planned vs. actual expenditure, and a vision for the next cycle. This milestone might lie in the next inception phase or even coincide with the LCO milestone of the next cycle.

Output

The output of the first cycle, the initial development cycle, is an operational software product. Usually the product goes afterwards through several evolution cycles where all phases of RUP are repeated but with different emphasis on the various phases. The output will always be an operational software product until the final version is released.

Process Workflows

In RUP a workflow is defined as "*a sequence of activities that produces a result of observable value*." These workflows are the meaningful representation of activities, artefacts and roles and therefore they are the focal point of my analysis. RUP is composed of nine core workflows which are divided into six engineering and three supporting workflows.

Six engineering workflows:

- Business modelling workflow
- Requirements workflow
- Analysis and design workflow
- Implementation workflow
- Test workflow
- Deployment workflow

Three supporting workflows:

- Project management workflow
- Configuration and change management workflow
- Environment workflow

Figure 11 provides an overview of the general structure of RUP with its phases, iterations and workflows. In the following these workflows are described in more detail to identify maintenance-related activities.

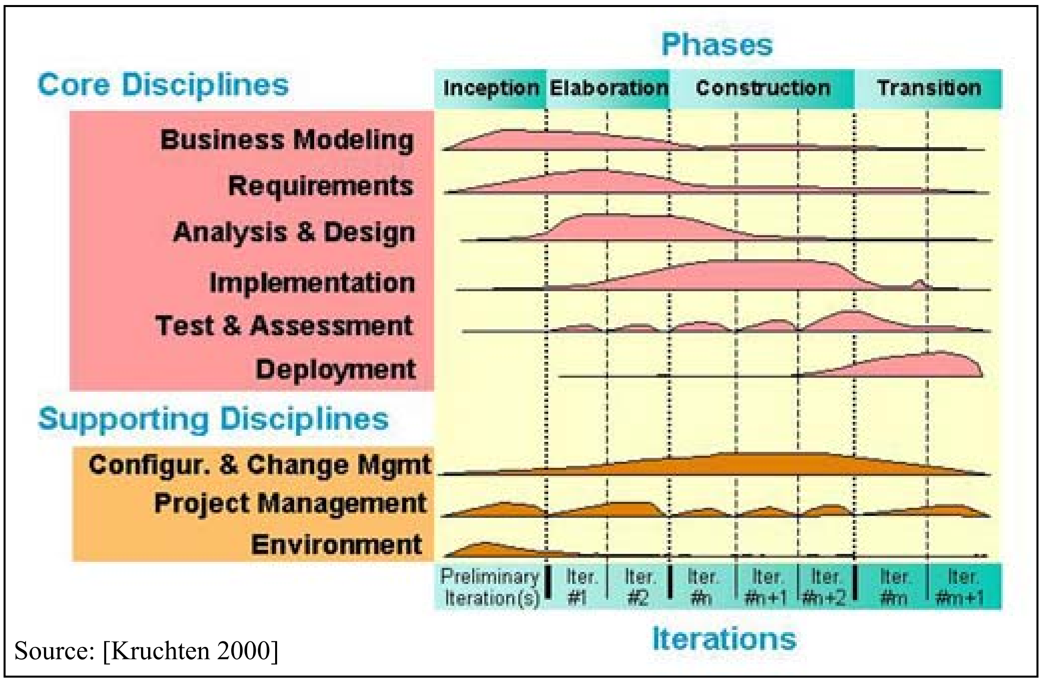


Figure 11: Rational Unified Process - Overview

The workflow descriptions with the exception of the statements about maintenance are based on [Kruchten 2000]. The focus of the descriptions under the heading **Purpose & Process** is on the content of the workflows, only the sections about **Maintenance** focus on and assess whether and how maintenance is covered by the respective workflows. The order of the workflows follows the list above, first covering the engineering workflows, followed by the supporting workflows. The only exception is the project management workflow which comes first because it sets the framework for a software project.

Project management workflow

Purpose & Process

Software-intensive projects need a framework that provides managers with guidelines for planning, staffing, executing and monitoring those tasks. Additionally project related risks like delays and budget overruns need to be managed.

Many factors like the project idea, staffing and other resources and a rough estimate of the total effort influence this management task. In most cases the date of the final release sets an important limit for the project. If a previous version of a product is being altered, artefacts of previous iterations also play a crucial role.

According to [Kruchten 2000] "*the workflow details, i.e. the composition of activities, depend on the fact if the iteration is the initial one or belongs to the subsequent iterations*". During the initial iteration the project as a whole need to be organised and assessed. This includes the identification and assessment of the project scope and risk, the development of a business case and the development of a software development plan.

The last artefact, the software development plan, which contains a risk management plan, a problem resolution and a measurement plan, is developed by the project manager and reviewed by the project reviewer (if this role exists).

During subsequent iterations the planning of those iterations and the monitoring and control of the project are the core activities. Iteration plans (one per iteration) and iteration assessments are produced as artefacts. At the end of every phase the respective lifecycle milestone is being reviewed, at the end of the project the project as a whole is being evaluated.

Maintenance

All of the activities are general project management activities which need to be applied in some form to maintenance projects as well but they are not maintenance specific. The adaptation of certain activities from the first iteration (development) to later iterations (evolution) could be seen as relevant for maintenance because in the latter cases an operational product exists that is being modified.

Business modelling workflow

Purpose & Process

The main goal of business modelling is "*to derive software requirements from business models*" [Kruchten 2000]. To achieve that it is necessary to understand the structure and dynamics of the organisation, to understand current problems and to identify improvement potentials. Stakeholders, here customers, end users and developers, need to have a common understanding of the target organisation so that the system requirements which are derived actually support this target organisation.

First the current business status has to be described and assessed before the business processes including roles and responsibilities can be identified and refined, if intended.

Depending on the context and the intention of the system, information at different levels such as business or domain level are transformed into a business vision document, a business use-case model and a business object model. Supporting artefacts are an assessment of the target organisation and a business glossary.

Stakeholders provide input which is used by the business process analyst to lead and coordinate the business modelling by outlining the organisation being modelled. The input is also used by the business designer who details the specifications of a part of the organisation.

Maintenance

Most activities are relevant for larger modifications of the categories adaptive and perfective & preventive maintenance. Especially the refinement of existing or the definition of new business processes can be a trigger for maintenance projects and thus requires maintenance skills. However, those activities and skills are not maintenance specific.

Requirements workflow

Purpose & Process

Having agreed on what the business does or should do, the customers and other stakeholders need to establish and maintain an agreement on what the system should do and why it should do that. These are the needs which provide the system developers with a better understanding of the system requirements. This helps "*to define the boundaries*" of the system and "*to provide a basis for planning the technical contents of the iterations and for estimating cost and time to develop the system*" [Kruchten 2000]. Another goal of this workflow is to define a user interface for the system focused on the needs of the users.

The first task of the system analyst, who leads and coordinates the requirements elicitation, is to analyse the problem and understand the stakeholders' needs. To achieve that the stakeholders' requests have to be translated into needs and those again into a set of system features to be considered for delivery. The artefact that supports this activity is the use-case model which can serve as a communication medium and as a contract among the stakeholders, also scoping the system.

As the requirements change those changes have to be controlled and the impact has to be assessed. The original requirements as well as the changes need to be documented so that later on others can understand what has been decided and most important why these decisions have been made.

Maintenance

Again, most of these activities are also relevant in the context of maintenance but are not maintenance specific. Requirements workflow activities during evolution cycles can be useful for pure maintenance projects, i.e. without a development cycle right before the evolution cycles. The management of changing requirements is one of the underlying activities of maintenance projects.

Triggers for maintenance can include misunderstood requirements and any errors in the translation process from user request into needs into system feature and finally, in this workflow, into requirements.

Analysis and design workflow

Purpose & Process

To bridge the gap between requirements and implementation, the requirements need to be translated into a specification that describes how to implement the system. The given inputs for this task are, of course, the system requirements identified before but also existing design elements. If no design elements exist, a candidate architecture has to be defined. Otherwise, i.e. in the case of evolution cycles where an operational product already exists, the existing architecture needs to be refined. New design elements are integrated with existing elements so that the consistency and integrity of the architecture is maintained.

In parallel to the refinement of the architecture, the descriptions of use-cases are transformed into a set of elements on which the design can be based. These elements are then completely specified, i.e. how the required behaviour can be implemented.

The output of this workflow is the design model, the major blueprint for the system. The main roles are the architect who establishes the overall structure for each architectural view (in a metaphorical language: the big picture – the forest), and the designer who defines one or several classes (metaphorical speaking the details of the big picture, i.e. the trees within the boundaries of the forest).

Maintenance

A translation of requirements, depending on the scope of the modification, is also required for maintenance. Given that an architecture will exist when maintaining software, the activity *refine the architecture* is especially important with an architecture centred approach like RUP. Although the identification and modification of existing design elements is mentioned, nothing is said about the necessity to understand or recover the given architecture. Also existing documents need to be understood and maintained. This cannot be taken for granted.

A modified design model will also be needed as a specification how to implement the changes. The close link between design and coding is emphasised in RUP.

Any errors in the translation of the requirements leading to wrong specifications will be a trigger for maintenance when these errors are discovered.

Implementation workflow

Purpose & Process

A well-defined organisation of code in terms of implementation subsystems is one of the purposes of this workflow. Other goals include the implementation of classes and objects in terms of components, the testing of these developed components as units and the integration of the results into an executable system by individual implementers or teams.

The implementation model is structured with the goal to ensure "*a near conflict-free process"* [Kruchten 2000]. For each iteration an integration build plan needs to be made that has to state which subsystems should be implemented and their order. Afterwards the plan has to be executed, i.e. the components are implemented and each subsystem is integrated into the system.

Round-trip engineering will allow a close tie between design and implementation also during maintenance. A person can either modify the design model and regenerate the code or modify the code and then alter the design. However, this is only possible with a limited number of programming languages and tools.

Maintenance

The activities *structure implementation model* and *plan the integration* are becoming more important with increasing size and complexity of the system. If previous activities are done according to the book, i.e. exactly follow the rules of RUP, maintenance will be very similar to development because with regard to implementation itself there are no maintenance specific activities which are not also applied for development.

Round-trip engineering helps to avoid the design getting out of sync with the implementation. This is a crucial aspect for later changes as an adequate documentation is named as important not only but especially in the V-model; design can be trusted by implementers.

In case of the modification of existing system, even if these have been developed according to RUP and a complete documentation is available, understanding of the code and the documentation is essential. If design documents are not up to date, reverse engineering for code understanding is required.

Triggers for maintenance at this stage are implementation errors which will lead to corrective maintenance, either immediately after unit testing or hopefully after integration testing, which is part of the test workflow, at the latest.

Test workflow

Purpose & Process

The purpose of the test workflow is to assess the product quality. This involves testing the quality of the architecture trough to the final product. Its objective is to ensure that all defects are addressed before the software is deployed to the customers.

The test workflow contains most elements of the software development process adapted to test development and test implementation. First the purpose and goals of testing have to be established by generating a test plan. Test design transforms use-cases into test cases leading to a test model, a representation of what will be tested and how. Afterwards the tests are implemented and executed. The execution involves integration and system test and finally acceptance tests. The outcome are test results, which show defects which have to be corrected.

An additional but also important part is test evaluation. It is necessary to ensure the quality of the test process.

Maintenance

Testing is *the* trigger for corrective maintenance. Every test case that is not passed leads to maintenance. This requires an appropriate test plan and test design.

Not only before but also after maintenance has occurred testing is crucial. It is important that maintenance does not reduce the overall quality of the product. In this context regression testing becomes a critical activity to ensure that changes made to the code have not introduced new or reintroduced old defects.

An increasing number of tests will require test maintenance as well.

Deployment workflow

Purpose & Process

The purpose of deployment is "*to turn the finished product over to its users*" [Kruchten 2000]. To achieve that, the deployment needs to be planned well. This requires a high degree of customer collaboration and preparation.

The operational product is tested one more time at the development site which approximates the intended target site. At the same time the support material is developed. This includes information that will be required by the end user to install, operate, use and maintain the system.

Afterwards the release is created, that is the operational product with all additional artefacts. This release is then deployed to the customers. The mode of deployment depends on the kind of software. Examples for different modes are direct installation at customer site for custom build system, distribution of shrink-wrapped software and software that is downloadable from the internet.

In the first case, the product is installed by the developing organisation and tested by the customer, in the latter cases the customers install and test the software on their own.

Maintenance

When software has been modified the current releases need to be deployed to the users. Therefore the deployment workflow activities can also be applied to maintenance but are also not maintenance specific.

Depending on the context it needs to be defined how to deploy updates and new releases and how to inform the end users how to maintain the system.

Unexpected installation problems at the customer site can be a trigger for maintenance as well as mistakes in the documentation or installation guidelines.

Configuration and change management workflow

Purpose & Process

Especially in iterative development projects, artefacts are changed again and again. It is the purpose of the configuration and change management workflow to keep track of these changes, to maintain the integrity of evolving project assets, to keep the documentation in sync, and to manage baselines and releases.

This workflow is concerned with all project artefacts (which are under configuration control) and relevant for all roles because everybody can submit a change request and, on the other hand, might be affected by one.

Configuration and change management deals with three interdependent areas, configuration management, change request management, and status and measurement.

Configuration management is related to the product structure, which covers artefact identification, versions, dependencies, and configurations that are consistent. Its purpose is to create a project configuration management environment that should facilitate product development where all people can work on the correct artefacts. The outcome is a configuration management plan that describes policies and practices to be used on the project for configuration management, and is part of the software development plan.

Change request management is related to the process structure, which covers keeping track of change request and analysing the potential impact of change. Its purpose is to ensure that changes are made in a consistent manner and that appropriate stakeholders are informed of the state of the change request and product. Changes request, which may be of a wide variety, are each associated with the originator and root cause; the impact of the change is attached later.

Status and measurement is related to the control structure, which covers the extraction and provision of relevant status information for the project management.

Maintenance

This workflow is concerned with managing modifications for all kinds of reasons, e.g. faults, changing requirements and/or environment, etc. Every modification request leads to maintenance activities, i.e. is a trigger for maintenance, even if it is only the decision to deny the request. Thus all aspects of this workflow are relevant in the context of maintenance.

Maintainers need to be able to look at the history of artefacts to understand the current state of the product; history and reasons for change must be available for them.

Reporting is also important to identify general problem areas and to provide useful information for future projects.

However, although many aspects are covered, it is not a maintenance management workflow because it is not adapted to suit all needs of maintenance such as proactive management of change requests.

Environment workflow

Purpose & Process

The environment workflow lays the foundation for a supportive development environment, i.e. it supports the development organisation with both processes and tools.

First it prepares the environment for a project. That includes the assessment of the current organisation and tool support, and based on that the production of a list of tools and templates for the development project. The artefact of this task is the development case which specifies the tailored process for the individual process. This process will be continuously improved during development.

When the environment is prepared for a project, it needs to be prepared for an iteration. This involves the preparation of tools and templates for the iteration and the training of people how to use and understand the tools. Furthermore guidelines for tools, design, modelling, programming and testing for the iteration have to be established.

During development the developers get support for the tools and the process.

Maintenance

These activities are generally needed to provide adequate support during development and also during maintenance. No specific aspects are mentioned for support tools needed during maintenance which are different from those needed during development, e.g. analysis tools for reverse engineering activities that are not required for the forward engineering process of development.

Maintenance in RUP

The Rational Unified Process consists of a development cycle followed by evolution cycles. During the evolution cycles an existing software product is modified. This can be defined as maintenance.

However, maintenance is not explicitly contained in RUP. Many activities are included within the nine core workflows which can be applied for maintenance tasks but they are not maintenance specific.

In a "perfect world" evolution iterations are similar or equal to maintenance and RUP would nearly include everything that is needed during the life cycle of a software product. If a product is not still under RUP construction or has been developed outside a perfect RUP world, i.e. not following all relevant guidelines of RUP for software development than RUP does not include everything needed for effective maintenance.

Important activities for maintenance, such as the identification of modification requests, the classification of maintenance categories to trigger different process, and the identification of the impact of the modification, are neglected. And even in a "perfect RUP world" analytical activities such as impact analysis should not be omitted.

[Aversano, Canfora and Stefanucci 2001] write that "*a maintenance process follows the same phases of a software development process*" just the outcome is different in the sense that nothing new is created but only modified. As mentioned before it is obviously difficult or even impossible to draw a clear and distinct line between development and maintenance. Although some activities, such as the classification into maintenance categories, are only used in maintenance, many activities will be applied in both cases.

There is constant feedback and thus constant change. Maintenance is not explicitly included in the model but it is contained in many activities in all workflows especially in the supporting workflow *configuration and change management*. Theoretically many of those and other feedback providing activities can be put under the heading of maintenance.

In these cases it is really important to determine how maintenance is perceived in the organisational context. This can be either as an ongoing process until the software product has been discarded, thus maintenance is included in most activities. An alternative is that the transition phase is followed by a specific maintenance phase during which more or less the same will be done as before, that is delivering increments just under a different heading.

A further insight and a summary from a comparison with chapter 4 Fundamental Maintenance Process Activities can be found in chapter 7 Conclusion - Comparison of Theory & Practice.

4 Fundamental Maintenance Process Activities - IEEE Standard for Software Maintenance -

This chapter aims to give a broad overview of maintenance activities and to analyse the maintenance process. A comparison with software process model helps to identify which activities are covered by those models and which are neglected. It does not provide a detailed description for application by practitioners but indicate the main activities which need to be applied within the maintenance process.

This chapter presents fundamental maintenance process activities from literature. The [IEEE 1998] maintenance process serves as a framework. It has been extended and adapted so that all required activities are listed and described within this chapter.

The original IEEE Maintenance Process is sequential and thus contains a repetition of activities in certain stages. The framework presented here is not divided into linear phases but eight blocks of activities belonging together or having a similar purpose. Some activities are applied iteratively throughout the process with different intensity. Especially analysis and design activities are strongly interwoven. This perspective is comparable to RUP.

In the following the process activities are described using the process model presented above (cf. figure 4). Additional comments are added how/where these activities are included in process models, mainly RUP to identify what is contained in RUP and what is missing. Partly more extensive comments are made to emphasise important activities, problems which can arise or to stress the views of certain authors about those activities.

Overview of activities

4.1 Identification & Tracing	▪ Identify problem/modification ▪ Manage modification/maintenance requests ▪ Classify type of maintenance ▪ Schedule implementation of modification requests
4.2 Analysis	▪ Identify impact of the modification ▪ Analyse alternative solutions ▪ Identify safety and security implications ▪ Define firm requirements for the modification ▪ Identify the elements of modification/ Impact analysis ▪ Program understanding/program comprehension ▪ Cost analysis/estimation
4.3 Design	▪ Modify software module documentation ▪ Create test cases for the new design, including safety and security issues ▪ Identify/create regression tests ▪ Identify documentation update requirements
4.4 Implementation	▪ Structure implementation ▪ Coding ▪ Unit testing ▪ Integration

4.5 System Testing	▪ Test-readiness review ▪ Integration test and system test ▪ Regression testing
4.6 Acceptance Testing	▪ Design test ▪ Productive system test ▪ Support material acceptance test
4.7 Delivery	▪ Installation at customer site ▪ Final testing ▪ Notification of users/customers ▪ Deployment ▪ User training
4.8 Maintenance Management	▪ Determine maintenance effort ▪ Manage maintenance process ▪ Manage maintainability ▪ Configuration management ▪ Manage people

4.1 Identification & Tracing[5]

The first stage is concerned with the identification and management of modification requests. Maintainers validate the modification request, classify it according to the categories of maintenance such as corrective or adaptive maintenance (cf. 3.1; which ever categorisation is used by the organisation) and assign a priority. Based on that they can estimate resources required to fulfil the request. The requests are then traced through the process until their fulfilment is confirmed.

Identify problem/modification

Purpose

The objective is to actively identify and understand problems within the area of concern and to anticipate areas that need modification now or in the near future. Problems need to be validated to be acted upon.

Input

The inputs are all sorts of information about the system and its context. This includes error messages, changing requirements in the form of new ideas and needs, business domain information, context and intention of the system, and reports about change-prone applications and software modules.

Process

The process is concerned with proactive management of modification/maintenance requests, which means the active search for areas which need modification. This includes the gathering of modification requests from business ideas, a close cooperation with business process modelling and request for feedback from users.

[5] IEEE standard calls it: Problem/modification identification, classification, and prioritisation

In general terms this means that maintainers acquire modification requests, i.e. a problem statement or a description of a particular need from customers and users.

An initial step after receiving a modification request is to check whether the request is valid or not, i.e. whether some action needs to be taken or not.

Output

The outputs are validated modification requests which need to be acted upon.

Role

Two main roles are needed here. The first one is a receiver, a person that is known and can be reached easily via all means of communication and who has to collect all incoming modification request s. The second role is a validator. This role needs to be occupied by skilled people such as system analysts who can distinguish between real modification requests and other problems such as wrong input or handling mistakes.

A supporting role is a stakeholder who provides input of all kinds.

Tools

All means of communication such as email, conversations, or phone calls, are seen as a tool because only if it is easily possible to submit modification requests the stakeholders especially the user will do so. A specific tool is a web interface which is connected to a database where requests can be stored and additional information can be added until the request's validity has been confirmed. This can be same database as used for modification request management (cf. Manage modification/maintenance requests; next activity) but the status must be clear.

Embedding in Process Models

All linear models start with the realisation of a need to do something, i.e. to start a project. However, neither the identification nor the active search for needs are part of the models. In XP the user stories within the Planning Game are the encouragement to search for new ideas; the validity check is done through discussion.

The business modelling workflow in RUP contains the idea to derive software requirements from business models and the change management includes the anticipation of future needs. A validity check might be contained in change request management but it is not explicitly mentioned in [Kruchten 2000].

Manage modification/maintenance requests

Purpose

The goal is to provide current figures about the status of all modification requests and thus ensure control about the project.

Input

The input to this process is the modification requests and all relevant information related to the modification requests.

Process

The first step in the process is to assign an identification number to every modification request. In the best case this number partly states the problem if it is done together with classification. The numbers are used for all changes associated with the modification request in the documentation or source code.

Constantly information about the status of the modification request is added. This enables the tracking and tracing to monitor if a modification request is completed.

Reporting procedures which provide regular reports for relevant stakeholders are also part of the modification request management process.

Output

The main output is the current status of all modification requests. According to [Bennett, Cornelius et al. 1991] other outputs are statistics, for example about the time to resolve errors, change-prone modules, outstanding requests and resources required for certain types of changes, and reports providing lists sorted by requests or customers.

Role

The modification request control manager is responsible to oversee the process and track delayed and outstanding modification requests. All other roles provide status information about the modification requests they are working on.

Tools

The required tools are a database into which all information can be entered and an application that helps to track the modification requests and generate relevant statistics and reports.

Embedding in Process Models

The monitoring of the status of the project is an implicit or explicit part of all models. Especially in RUP the configuration and change management workflow contains all activities that are necessary to ensure a proper tracking and tracing of all modification requests.

Classify type of maintenance

<u>Purpose</u>

The intention is to put the modification requests into categories that trigger different processes.

<u>Input</u>

The inputs are validated modification requests and, if they already exist, categories of maintenance.

<u>Process</u>

Initially the categories for maintenance need to be determined. One suggestion for categories is corrective with the subcategory emergency repairs, adaptive, and perfective maintenance. Criteria which determine the category need to be set up and reviewed regularly if the modification requests are categorised correctly. Depending on the organisational context and the existing maintenance processes a matrix classification containing several levels can be used.

The activity that is always performed is to classify incoming modification requests. This also determines a first kind of priority since emergency repairs need to be executed immediately.

<u>Output</u>

The output can be seen from two perspectives either as modification requests with a classification or as types of classifications filled with modification requests.

<u>Role</u>

The classifier is someone who knows the processes which are triggered and understands the requirements of the modification requests to classify them accordingly.

<u>Tools</u>

A useful tool is a partly automated database containing classification criteria which generates suggestions for classification based on the information provided.

<u>Embedding in Process Models</u>

Classification in the way it is described here is not mentioned in linear models. The user stories in XP are somehow sorted which determines when they are implemented. The intention is similar to the purpose of the classification activity described here.

The configuration and change management workflow in RUP possibly contains a classification activity but it is not explicitly mentioned

Schedule implementation of modification requests

Purpose

The purpose is to schedule the implementation conflict-free with the available resources.

Input

Available resources, priorities and the modification requests are the inputs for this task.

Process

Based on existing information a priority is given to the request to enable workflow planning and the assigning of responsibilities. Afterwards a modification request is assigned to a block of modifications scheduled for implementation. The schedule is continuously updated, especially after the analysis when the required efforts for the change are much clearer.

Output

The output is an implementation plan/schedule or a project plan if the implementation of the modification requests is a self-contained project.

Role

The planner needs to be a person with the knowledge and decision making power required to assign the priorities.

Tools

An application for this task is a project management tool that supports assigning of resources and continuous re-scheduling.

Embedding in Process Models

The implementation plan in linear models is fixed early in the process. In XP user stories are assigned to iterations for implementation which is similar to the intention here. Scheduling is contained in RUP in the project management workflow but the constant update and change is not intended in RUP in the form described here.

4.2 Analysis

All artefacts serve as input to the analysis, the second stage, during which a feasibility analysis and a detailed analysis are conducted. The effort spend on this stage depends on the quality of the artefacts. [Phillips 2000] calls this the stage were "*all the re-'s are needed*" referring to reverse engineering, re-documentation, restructuring, reengineering and design recovery.

Feasibility analysis

The analysis stage is divided into feasibility and detailed analysis. The purpose of the first part is to analyse the feasibility and scope of the modification. If the result shows that the project or the task is feasible and worthwhile, a detailed analysis is performed to examine all the relevant details which need to be taken into account.

Identify impact of the modification

Purpose

This activity aims to evaluate the scope of the modification and to estimate whether and how to proceed.

Input

The inputs are (in contrast to above) one specific modification request and all relevant artefacts which are available. If the source code is the only available artefact, reengineering during the detailed analysis is required.

Process

To determine the impact of the modification in general, a rough analysis of all existing artefacts such as specifications, documentation and may be the source code itself needs to be performed. The objective is to identify affected products and artefacts and the elements which need to be changed. According to the purpose a rough estimate about the scope is sufficient, a detailed analysis is not intended at this stage.

The impact on business processes, i.e. whether any changes are required, as well as the short-term and long-term costs (outside the pure implementation of the modification request) need to be determined. This has to be contrasted to the value of the benefit of making the modification.

Output

The result is an assessment about the scope and thus the impact of the modification and based on that a decision whether to proceed or not.

Role

The role for this activity is not precisely defined. Whoever is responsible for the area of the request and has the relevant domain knowledge to be able to perform this task has to analyse the problem.

Tools

Tool support at this stage can include visual modelling tools e.g. for use case modelling to demonstrate the scope through people and processes involved or other modelling tools which assist the identification and demonstration of the scope of the task. Support for calculations of the costs and benefits are also important.

Embedding in Process Models

In linear models it can be assumed that this is part of the normal analysis but it is not explicitly mentioned. In XP it needs to be quickly determined what to do next and thus a first analysis is probably very roughly included. RUP contains several analysis tasks but they are not sufficient for a first estimation after which a decision can be made whether to proceed or not.

Analyse alternative solutions

Purpose

The objective is to identify and assess several alternatives before a decision is made how the modification request is being implemented. This activity can be omitted in case of very simple corrections. The complexity can be determined at stage one.

Input

The inputs consist of the modification request, all artefacts which are relevant and support the evaluation process, and information about the feasibility of alternatives (which mainly need to be determined during the process).

Process

As part of the feasibility analysis several alternatives have to be identified and established. An outline of every solution needs to be designed. This can include the use of prototyping to test certain aspects of the alternatives. The different solutions are evaluated and a decision is made which is the best one to be implemented.

Output

The output is a decision and an outline of the solution suggested for implementation.

Role

The system analyst leads and coordinates the requirements elicitation. The architect and the designer perform the design and analysis of alternatives. Depending on the importance the project manager together with the customer approve the recommended solution.

Tools

To generate and analyse several alternatives tools for modelling, visualisation, and simulation, e.g. an environment for prototyping, are required.

Embedding in Process Models

The assessment of alternatives is not mentioned in linear models; however, it is somehow included in the analysis stage. The spiral model explicitly names the evaluation of alternatives as part of the model. Due to the quick implementation in XP, a thorough analysis is not performed. RUP just mentions it as part of the analysis but it can be considered as insufficient.

Identify safety and security implications

Purpose

The purpose is to identify all aspects which can compromise the success of the modification with regard to safety and security.

Input

The modification request and detailed information about the system are required for this activity. Current safety and security issues as well as the data security policy of the company are also aspects which need to be available for analysis.

Process

The first step is to identify critical areas within the system. This is a general step and does not to be performed for every modification. If the critical areas are known, the problem outlined in the modification request needs to be analysed and understood. Information from the activity 'identify impact of the modification' can be used to determine if critical areas are affected by the modification.

Initially for the feasibility analysis only a rough draft of safety and security implications is established. Later on, this draft needs to be specified further.

Output

The output of this activity is obviously a list of aspects to be observed when designing the system. This list needs to be considered during further analysis, design, and also for test design and testing itself whether the aspects were observed.

Role

A system analyst has the task to identify general areas where safety and security implications need to be observed. Later on together with an architect and a designer the implications have to be specified in more detail.

Tools

For a detailed analysis of safety and security implications it is necessary to trace certain functionality to the code, which can be supported through a tool.

Embedding in Process Models

An activity for the identification of those aspects is not covered by any process model.

Detailed analysis

The result of the feasibility analysis can either be a decision to proceed or to abort the project. In the first case a more thorough analysis is required before a design can be established. However, also at the end of the detailed analysis a decision not to proceed any further can be made.

Define firm requirements for the modification

Purpose

The goal is to establish and maintain an agreement of what the system should do after the modification, to define the boundaries of the modification and to provide a basis for design.

Input

The inputs are the modification request, the results from the previous feasibility study and all functional and non-functional requirements identified and established so far.

Process

To understand the task and the requirements for the system the problem needs to be reproduced (in case of corrective maintenance). An agreement has to be established and maintained with customers on what the system should do, i.e. their needs have to be understood. This enables a definition of the system, which is achieved by establishing a set of features to be implemented using actors and use-case models needed for each feature. The requirements are detailed further, only stating what the system is supposed to do and not specifying how.

Output

The output is an analysis model which is a rough outline of the system or its affected parts. It omits details and provides an overview of the system's functionality.

Role

A system analyst leads and coordinates the definition of requirements. A designer develops the analysis model and an architect ensures the integrity of an existing system architecture.

Tools

A requirements management tool captures all relevant attributes and enables their traceability to other items. Modelling tools support the development of relevant views of the analysis model.

Embedding in Process Models

As this is a basic but fundamental task it is included in all models in a different form. Linear models contain it in the analysis phase, XP in the form of a user story and RUP covers the definition of requirements with the activity 'define the system' in the requirements workflow.

Identify the elements of modification/Impact analysis

Purpose

This activity aims to provide a basis for design by assessing the effects of making the modifications to a system

Input

The modification request, the analysis model roughly outlining the system, and other relevant artefacts which are needed by the performing roles serve as an input.

Process

The identification of elements for modification can be done together with the definition of firm requirements. However, it is a self-contained activity that is especially important for maintenance projects in contrast to development because the elements which are going to be modified need to be identified before detailed design activities can start. Therefore the identification of elements together with an impact analysis should be located between analysis and design of the modification [Antoniol, Canfora, Casazza and Lucia 2000].

The requirements specified before are traced to the design and the code. Features which already exist need to be identified in the design documentation and the source code. These affected elements and the potential impact of their modification are then included in the analysis model, which can serve as a template for the design model.

In the context of impact analysis [Bianchi, Fasolino and Visaggio 2000] claim that the success "*depends on granularity of software representation model*". It is a trade off of efficiency versus accuracy. Organisations need to establish objectives whether the efficiency or the accuracy of the modifications is important and chose the granularity accordingly.

Since there are often many ways to implement a change, evaluate the actual impact after the modification has been implemented to gather information for future modifications and analysis.

Output

The result is an improved analysis model that provides an overview of the intended functionality, the elements which need to be modified and the impact of the change.

Role

The roles involved here are the same as in the activity 'define firm requirements for the modification', mainly a system analysis, an architect, and a designer who detect the elements affected by the defined requirements.

Tools

Tools which support the tracing of features and functionality to the documentation and code support this identification or tracing activity.

Embedding in Process Models

This activity is not included in the Waterfall model. Although the V-model covers the segmentation of elements to be designed it does not cover the identification of existing elements which are to be modified. XP does not mention this activity. In RUP the situation is similar to the V-model in the sense that the identification of existing elements to be modified is not included.

Program understanding/program comprehension

Purpose

Often the source code is the only reliable representation of a system that exists. Programmers have to modify a system and for doing so they need to understand the functions and behaviour of the system before someone can modify it. The aim of this activity is to identify procedures on the program level, to understand the functionality of the system and to create models of the program behaviour.

Input

The input for this activity is old code, i.e. code that is unfamiliar to current programmers because it has been programmed before they joined the team. Other characteristics of old code are "*poor design, use of an obsolete programming language and/or missing or inaccurate documentation*" [Corbi 1989].

Process

The process can include an analysis either of the whole program or only of affected elements to minimise the efforts for the current modification. [Corbi 1989] names three possible actions for program understanding: read about it, i.e. read the documentation; read it, i.e. read the source code itself; and run it and watch it, i.e. execute the program and observe its behaviour.

The information obtained need to be transformed into a meaningful documentation. This can range from plain text stating problematic areas like code that cannot be executed to detailed models providing on overview of the structure of the code.

[Bratthall and Wohlin 2000] focus on program understanding and notice that architecture and design models primarily represent functional behaviour of the system. They demand to include non-functional aspects like experiences from previous releases and modifications which are important to understand what is going on, as most modifications happens in the designers' heads.

Output

The ideal output is a detailed documentation so that others can understand the system's functionality and behaviour when designing modifications of the system. However, as mentioned above, program understanding as well as program developing is a process that happens in the designers' heads. Therefore a result can be a programmer who understands the system and can apply the knowledge to the required modification.

Role

This activity can be or need to be performed by the roles which have to understand and modify existing artefacts such as architect, designer, system analyst, and implementer.

Tools

Tools for program understanding can mainly be put into two categories which are static analysers that analyse the system without its execution or dynamic analysers that perform an analysis of the system during its execution.

Embedding in Process Models

Due to the fact that all of the process models described in chapter 3 focus on forward engineering, activities like program understanding are not covered by any model.

Cost Analysis / Estimation

Purpose

The objective is to make a preliminary estimate of the size or magnitude of the modification that is as precise as possible at the current state of the project and can serve as the basis of further planning.

Input

The input consists of information about aspects that influence the required effort for a project or modification request. The information contains data about the software product, personnel, the development environment including tool support, and project characteristics.

Process

Most methods to estimate the efforts of programming tasks focus on development projects. Therefore it is difficult to estimate the effort that is required to modify an existing system. A comparison of similar projects is a method that is often dismissed for development projects because they are so dissimilar to each other. However, it can be used for maintenance projects because certain modifications are by all means comparable.

[Sneed 2001] describes a method to estimate the effort of a maintenance project. First the impact analysis identifies the artefacts and elements affected. A degree of impact needs to be assigned to any artefact; information about function points and objects points contained in the affected elements have to be available. Together with an assessment about the development mode, i.e. organic, semi-organic or embedded, and data about cost drivers the overall effort in person-days can be calculated according to the function point method. The "*COCOMO algorithm can be used to compute the minimum time for the maintenance project*" [Sneed 2001].

Again a decision can be made whether to accept, to reject or to evaluate the modification request further.

Output

Depending on the experience with comparable modification requests the result is a more or less accurate estimate of the total effort of the maintenance task.

Role

The project manager and the system analyst need to estimate the degree of impact of individual artefacts and have to determine the degree of complexity.

Tools

Required is a tool that determines function points etc. per affected element and calculates the overall maintenance effort.

Embedding in Process Models

A cost analysis is not explicitly covered by linear models. Small and simple releases are fundamental practices of XP, therefore cost estimation for individual tasks is not relevant and thus not covered by XP. In RUP only a rough estimate of the total effort based on lines of code is mentioned. COCOMO is suggested for the validation of effort, time and staffing.

Comment

The introduction to this chapter quoted [Phillips 2000] stating that all the re-methods and –techniques are needed at this stage. These re-methods such as re-documentation, re-structuring, design recovery are often subsumed under the heading of re-engineering. The reason why re-engineering (or reengineering) is needed at this stage of the maintenance process is the following:

Often the only reliable representation of a system is the source code. Sometimes also other reliable documentation is available but usually not everything that is needed. Information are often only implicitly expressed, e.g. requirements specifications are implicitly contained in design but do not exist in an explicit form anymore. A representation of information at the required level of abstraction needs to be re-constructed. Reverse engineering extracts the information needed. Depending on the needs a design recovery or a re-specification might be required. [Byrne 1992] writes in this context that reengineers should only "*work at the level of abstraction at which information about that characteristic is explicitly expressed*". Or in other words, one should work only up to the level that will deliver good results or the information required. It is not necessary always to reach the top level or to remain at code level.

However, all this re-techniques are not mentioned in the IEEE maintenance process. The problem is that the granularity of the description is too detailed for those techniques. The activities that are part of those techniques are covered but they are not combined to methods. Therefore it is difficult to identify the re-techniques in the IEEE maintenance process. From this it follows that the IEEE process cannot be employed for the systematic application of dedicated reverse or reengineering techniques.

4.3 Design

During design changes to the design are made. This is similar to the design phase in development projects "*except that maintenance design has far more constraints because it must fit into an existing product*" [Phillips 2000]. This fitting in requires program understanding, an activity carried out under 4.2 Analysis. Furthermore documentation needs to be updated.

Modify software module documentation

Purpose

This activity aims to translate the requirements into a specification that describes how to implement the system and adapt to other constraints such as implementation environment or non-functional requirements.

Input

Functional and non-functional system requirements as well as affected software modules which all have been identified above during analysis are required for this activity.

Process

Since maintenance requires the existence of a software product an architecture will already exist. In some cases this architecture may require some refinements if possible. In all cases the designers transform the behaviour provided by use cases into a set of elements for the design. Afterwards the main step is to design the components themselves. This requires to refine and specify existing components and to specify new components. The design of components consists of class design and subsystem design. The modification of the software module documentation itself comprises e.g. the modification of data and control flow diagrams.

Output

The products of this process are a design model which is a major blueprint for the system and a software architecture document that provides various architectural views of the system.

Role

The roles involved in this activity are an architect who is concerned with the overall structure of the software architecture and a designer. Usually specialised designers for certain areas such as databases or real-time components are needed as well.

Tools

Tools needed at this stage are tools which are able to capture and manage different models and views. For certain environments round-trip engineering tools which keep design and code in sync are a good support.

Embedding in Process Models

This activity is covered by linear models. Since the XP philosophy requires as little documentation as possible this 'documentation activity' is therefore hardly included. RUP contains a detailed software documentation modification in the analysis and design workflow.

Create test cases for the new design, including safety and security issues

Purpose

The purpose of testing itself is described under 'System Testing'. The objective of this activity within the 'Design' stage is to develop the basis for all testing activities by providing a test model that can be used throughout the process to assess product quality at all stages.

Input

For this activity mainly the design information but also the modification request are needed. Additionally the safety and security aspects identified during the analysis are required.

Process

The process starts with the generation of a test plan which contains the requirements for tests and the identification of types of tests needed. Based on that, the tests themselves are designed. The use cases are transformed into test cases for system and acceptance testing. Afterwards these tests cases are implemented, at best in the form of a test script that can be read and executed by a computer.

The execution of tests is carried out during the testing stages. However, the test results are partly fed back to enable an evaluation of the tests in order to improve them for the future.

Output

Required outputs for the testing stages are a test model containing the purpose and goals of testing, the test cases and the test procedures, possibly implemented in the form of test scripts.

Role

The designer provides information about the modifications planned for implementation and affected areas of the system. The test designer plans, designs, implements and evaluates the tests.

Tools

Tool support is especially essential for the execution of the tests but it can also be valuable for test creation as well. Modelling tools assist the transformation from use cases into test cases and certain development tools can transform those into computer-readable test scripts.

Embedding in Process Models

In the Waterfall model testing, and thus test creation, only appears late in the process, which is a major drawback of this model. In contrast the V-model stresses verification at each stage and therefore the identification of test cases starts much earlier. XP contains continuous testing as one its fundamental practices which is applied throughout the process; programmers continually write test cases and test procedures. In RUP test design is covered by the test workflow with test planning and creation being performed during the elaboration phase.

Identify/create regression tests

Purpose

The purpose of regression testing itself, as described later on, is mainly to ensure that changes made to the code have not reintroduced old defects. The test cases have to be created early in the process together with design because at this stage it is easier to identify which areas are affected and thus what kind of tests are required.

Input

Inputs are old tests, previously discovered defects, and design information from the design model.

Process

An important issue is whether automated test tools can be applied for the software that is being modified or not. If automated test tools exist than all previous tests can be carried out again. Otherwise it needs to be identified which elements are to be modified, the impact of the modification and based on that the required regression tests. The process is principally the same as with new tests, i.e. a test plan is established, tests are designed and implemented.

The evaluation is especially important for regression tests because these tests are executed for every modification again. Not only the feedback from the testing stages but also from operations has to be evaluated if old defects have not been discovered by existing regression tests and thus they need to be improved.

Output

The outputs are a test plan for regressions tests, regression test cases and relevant test scripts. As test evaluation is also part of this process, test reports about the validity of regression tests and recommendations for further tests and usage are also produced here.

Role

The roles and their responsibilities are similar to those of ‘create new test cases’ with the difference being the focus on regression tests.

Tools

Tool support for regression tests is even more important than for new tests because regression tests are carried out again and again and thus efficient execution is crucial.

Embedding in Process Models

For linear models and XP the same applies as mentioned under ‘create new test cases’ with the exception that regression testing is not explicitly mentioned. In contrast RUP clearly emphasises the importance of regression testing and integrates this with other testing strategies in the test workflow.

Identify documentation update requirements

Purpose

Not only the software module documentation but also other kinds of documentation need to be updated. The objective of this activity is to identify all documentation update requirements so that those documents are updated during the process by the roles concerned.

Input

All existing documentation and documentation guidelines are a massive but minor input for this activity. Important are information about the required modification, mainly from the analysis stage, which support the identification of affected areas and therefore of potential update requirements.

Process

This activity only identifies the documentation requirements. The updating itself is done by the roles concerned. Just the identification needs to be carried out because the designer has the overview what areas are affected and can therefore trigger and control the updating process.

The identification should be performed in a top-down approach. Firstly, general areas such as system documentation and user manuals are assessed whether they need to be updated. Afterwards the individual areas are looked at in more detail, e.g. which areas of the user manual needs to be modified and who is responsible for this task.

In addition to the identification of update requirements for individual modification requests, new documents that are created need to be gathered and included in a database so that the identification is easier or actually possible for later modifications. Furthermore the necessity or right to exist of documents has to be evaluated regularly to eliminate needless and redundant documents.

Output

The result is not updated documentation but a list of documentation areas which need updates.

Role

The implementing role is the designer who has an overview and the experience to identify all areas which are affected and where documents need to be updated during the maintenance process.

Tools

The required support is database that provides a link between affected areas and related documents which need to be updated.

Embedding in Process Models

The Waterfall model does not mentioned this activity, the V-model due to its document-orientation is very thorough regarding documentation update requirements. XP tries to minimise documentation and thus this activity is neglected. Also RUP does not cover the identification of documentation update requirements as a dedicated activity. However, relevant documents are updated during the process within the context of the different workflow activities.

4.4 Implementation

The fourth stage is implementation. The code is changed according to the design developed at the previous stage. However, many organisations try to include the earlier activities here and start with coding right away which is, of course, not recommended.

Structure implementation

<u>Purpose</u>

To structure the implementation model means to define the organisation of the code with the goal to ensure a near conflict-free process.

<u>Input</u>

The input needed to structure the implementation model is the design model.

<u>Process</u>

The process can start when the activities from the analysis stage have identified components which have to be modified. Then it needs to be fixed what needs to be implemented and when. This means that the organisation of the code is defined in terms of implementation subsystems organised in layers.

Also included in this process is the planning of the integration, i.e. which subsystem should be implemented in which order. A plan has to be developed that considers the implementation as well as the integration of components.

<u>Output</u>

The results are an implementation model and an integration build plan which define the order in which the components and subsystems should be implemented and integrated.

<u>Role</u>

The architect defines the structure of the implementation model and supports the scheduling of the integration plan which is done by the system integrator who is later responsible for the integration itself.

<u>Tools</u>

Needed is a tool that assists the scheduling process by visualising dependencies between individual components and enables the structuring of a near conflict-free implementation process.

<u>Embedding in Process Models</u>

This activity is not covered by linear models. In XP the planning game covers some aspects of it but a proper planning step for implementation of the code is not included. RUP contains the structuring of the implementation model as well as the development of an integration plan.

Coding

Purpose

Using the results of the design phase, this activity has the purpose to implement the designed objects in terms of components.

Input

Required for coding are the results of the design phase, the implementation model, current source code, and existing documentation to support the coding process.

Process

The developers implement classes and objects as specified in the design model. They adapt existing components, write source code and compile the files. Feedback is given to the designers if errors or problems with the design are discovered.

Output

A component is piece of software code that is the result of coding.

Role

The role in coding is performed by the implementer who develops components and related artefacts. This can be done either alone or with a pair programming partner.

Tools

Classic development tools are e.g. editors and compilers. More advanced development environments enable round-trip engineering to either modify the design model and regenerate the code or to modify the code and then alter the design.

Embedding in Process Models

Coding is, of course, contained in all process models. However, coding practices vary. In linear models and RUP nothing is said about coding practices. XP emphasises pair programming.

Unit testing

Purpose

The objective is to perform limited tests of developed components as units before those are integrated into the system.

Input

The inputs are units, which are the smallest testable elements. General test principles can also serve as an important input but detailed test strategies are not required at this stage.

Process

The developers test the developed components as units directly with or after coding. This is done to verify the changes which are implemented. Code defects are corrected immediately. A code review can follow to ensure that programming guidelines have been observed.

Output

At the end of this activity tested units are ready for integration.

Role

The implementer performs the unit testing. This activity should be done by the same person as the coding itself. A code reviewer reviews the code if required.

Tools

A classic and simple tool is a debugger that supports the tracking and tracing of defects to the source code. Recommended are more complex development environments that support round-trip engineering, providing a close tie between design and implementation. This makes it easier to ensure that the software is defect-free.

Embedding in Process Models

Linear models contain unit testing as part of the implementation stage. XP incorporates unit testing as an important practice as part of their continuous testing paradigm. Unit testing is also completely contained in RUP.

Integration

Purpose

Integration aims to integrate the results from coding from several programmers into an executable system.

Input

The integration plan, existing code or the existing system, and the tested units form the input of this activity.

Process

This process integrates the results produced by individual programmers or teams into a combined system. This should be done at several levels. First the components of each subsystem need to be integrated before all subsystems are integrated in the system as a whole. Close cooperation with testing is required. To support the cooperation, incremental integration is recommended. That means that the components and subsystems are added one by one (or in very small groups of components) so that the detection of an error can be easily traced back to the last components or subsystems that have been implemented.

Output

The output of integration is an operational version of the system or part of the system that is either ready for system testing or, if integration has been done simultaneously with system testing, ready for acceptance testing.

Role

The role in integration is that of a system integrator who constructs a build which is an operational version of a part of the system.

Tools

An integration area or workspace for system integration is required here. It should also support collision detection to identify any element that does not fit in with the existing code.

Embedding in Process Models

Integration is a separate stage in linear models. XP contains continuous integration as a basic principle. RUP describes integration in several detailed steps.

4.5 System Testing

Testing is not only the responsibility of the testers involved in the testing stages. Every role has to ensure that the activities are performed properly. Even if it is not explicitly mentioned like 'unit testing' during implementation, results have to be check throughout the whole process. Moreover the importance of testing is emphasised by the fact that dedicated testing activities are split up into two stages, system testing and acceptance testing.

System tests which are conducted during the fifth stage comprise of new and old tests. These tests should be done preferably by people who were not involved in analysis, design and implementation.

Test-readiness review

Purpose

The purpose of the test-readiness review is to assess the preparedness of components and systems for system testing and acceptance testing.

Input

To perform this activity the test model and the artefacts ready for testing are needed.

Process

To receive some initial feedback a formal test-readiness review is not necessary. However, before the final testing of artefacts, the review needs to be performed to demonstrate that an internal milestone has been reached. Here the artefacts are handed over from designers and implementers to the testers. This step should avoid a code-and-fix mentality through a see-saw between implementation and testing by officially handing over completed artefacts.

The implementers have to examine that their artefacts are complete and ready for testing, a test designer checks whether this is the case. Later on the tester confirms that the artefacts have undergone a thorough test process and are ready for the next stage. The project manager approves acceptance testing because during acceptance testing customers are involved.

Output

The output is the approval for testing and a system ready for testing at the next level.

Role

The implementer confirms readiness for system testing, the test designer acknowledges it. A tester and the project manager approve the readiness for acceptance testing.

Tools

As this is mainly a check for completeness, a useful tool can speed up this process such as a database where all artefacts are listed and only need to be confirmed as present.

Embedding in Process Models

The test-readiness review is not covered by any model

Integration test and system test

Purpose

Testing is done to assess the product quality and to ensure that all defects are addressed before the software is deployed. At this stage, testing is done at the development site.

Input

Inputs are the test model, which is a representation of what will be tested and how, and, if implemented, test scripts for automated test tools. Additionally the documentation about the modification is need as well as the components and subsystems which are to be tested.

Process

A code review can be a fist step or nearly sufficient in case of minor changes of the code. Otherwise or afterwards other tests need to be executed. The first tests are integration tests and should be performed iteratively with the integration of components, i.e. integrate units, test, further integrate, test, and so forth. When all components are integrated the system is tested as a whole through the system test. This test is repeated until the whole system functions as required.

Obviously a close cooperation with integration and coding as well as with design is required if any problems due to implementation or design errors occur.

Output

The results are defects in the code listed in the test report and a supposedly defect-free system ready for regression testing.

Role

A code reviewer looks literally at the code, a tester executes the tests. The tester should not be identical with the designer or implementer.

Tools

Tool support is essential for testing because manual testing cannot be as productive as automated tests. Especially since many tests throughout the whole process increase the chances to discover defects, this can hardly be done manually due to high costs. Efficient testing tools execute the tests without manual interference and report any discovered defects. They still offer the option to interrupt the test at any stage to assess the code.

Embedding in Process Models

Linear models contain a dedicated testing stage, the V-model also with intensive feedback to previous stages. XP contains continuous testing following the principle of several iterations of integration and testing. The test workflow in RUP contains thorough testing procedures for many stages and types of tests.

Regression testing

Purpose

The objectives of regression testing are to ensure that no old defects are re-introduced or new defect are introduced and that the defects identified earlier on have been addressed and removed.

Input

Although units, components and subsystem can be subject to regression tests, the main input is a completely integrated system including documentation and relevant test cases.

Process

Relevant tests are executed, partly in close cooperation with other tests if the regression tests are performed during the integration of components. In case any errors are detected those need to be corrected by the developers and the tests are performed again until the software is defect-free. Feedback is given to the design stage where regression tests are identified and created.

Output

The test results are presented in test reports, feedback is provided for the creation of regression tests, and the software is ready for final acceptance testing.

Role

Regression tests should be conducted by an independent test function, at least by a tester not being identical with a developer who designed or implemented the modification.

Tools

What has been stated for integration and system tests also applies here. Actually efficient tool support is even more important for regression tests because they try to detect known old defects and those can be numerous.

Embedding in Process Models

Although testing is quite important in linear models as well as in XP, only the V-model mentions regression testing explicitly. RUP covers regression testing with the test workflow as a special type of tests.

4.6 Acceptance Testing

Additionally to system tests the acceptance tests need to be conducted. In contrast to the previous stage, where mainly IT people were responsible for the testing, this stage is for user testing. Although the [IEEE 1998] standard states that *"acceptance tests shall be conducted on a fully integrated system"*, it is recommended that acceptance tests start much earlier, possibly during design. Otherwise the usual disadvantage of a linear process arises that problems with e.g. design are detected by far too late in the process.

Design test

Purpose

The purpose of acceptance testing is to assess the product quality through acceptance of the users from the architecture through to the final product. Therefore these tests need to start as early as possible.

Input

The customer or user needs something that is understandable. This has to be the input to the process or needs to be generated during the process. Clearly visualised concepts or prototypes could be potential inputs for design acceptance tests.

Process

In order to assess the design the customer has to understand it. Existing documentation from the design stage needs to be evaluated and explained if possible. Otherwise the design documents have to be transformed into a form the user understands. However, changes have to be as little as possible not to influence the results. An option is to build prototypes for demonstration which is especially useful for user-interface design acceptance tests.

Output

The outputs are customer approved design concepts which can be specified further for implementation. Additionally test cases for final acceptance testing are provided.

Role

The roles involved are a test designer and a designer who plan, design, implement, and evaluate the tests. The designer also has to illustrate and clarify questions of the user. The role of the user is to assess whether the suggested design is understandable and fits to the required solution.

Tools

Useful are tools which support the presentation and visualisation of design concepts so that the user understands what can be expected.

Embedding in Process Models

Early acceptance tests in the form of design tests are not covered by linear models. Although XP projects have a customer representative on-site early design acceptance tests are not mentioned. RUP states that acceptance tests should be performed with complete applications or systems but mentions that also conceptual prototypes can be subject to acceptance tests.

Productive system test

Purpose

The purpose is to ensure that the operational system is acceptable for the end-users so that they are willing and able to use the system for their work.

Input

The input is a system (nearly) ready for operational use. Relevant documentation can support the acceptance test. Test cases identified before also serve as an input.

Process

Real user, i.e. people who have to use the system later on, test different aspects of the system if they are able to use it and if the system performs as expected and required. This is done at the development site during and close to the end of the development process. At the end these tests can be performed at the customer site to check if the behaviour of the system in the real operational environment is also acceptable. This can be especially important for performance testing. If some tests are performed at the customer site, a close cooperation with activities from the delivery stage is required.

Experience with user acceptance is fed back to the designers for later changes and for future modifications.

Output

The output is a productive system that is approved by users, that means it is ready for delivery.

Role

A test designer plans, designs, implements, executes and evaluates the tests. The main role is the user who assesses the system if it is acceptable for the tasks it is required to perform.

Tools

To evaluate the users' behaviour log tools are helpful which identify areas where users have difficulties finding their way through the program.

Embedding in Process Models

Depending on the form of the linear model, productive system acceptance tests are either a separate phase or included in the normal test phase. The on-site customer in XP projects has mainly to answers questions but also looks at the system in a kind of acceptance test. RUP states that acceptance tests are usually performed to assess the system's readiness for deployment but adds that testing in the test workflow occurs throughout the development process, which includes acceptance tests by the end users.

Support material acceptance test

Purpose

In addition to the system itself also the support material such as user manuals and training material need to be assessed whether they are appropriate or not.

Input

The main inputs are support materials of all kinds such as installation guide, user manuals, and training material. In addition to that a related version of the system needs to be available for being able to test the material.

Process

Depending on the situation the material needs to be distributed to the testers. They have to try, i.e. to use the material, or in case this is not possible they have to assess it otherwise such as reading it. This might be necessary for e.g. installation support material if an installation is not possible yet.

The test designer has to receive feedback and revise the material if required. A maximum number of iterations should be set at the beginning. At the end the support material has to be approved for usage and then it is ready for deployment.

Output

Acceptable and appropriate support material that covers installation, operation, and maintenance as well as training material are the outputs of this activity.

Role

Responsible for this activity is a test designer and/or a test supervisor who can evaluate if the testers do really understand the material and use it the way it is intended. Other roles are taken in by users from different levels such as technical personnel, training instructors, ‘normal’ users, and – if the responsibility for maintenance is split or handed over between different groups – maintainers.

Tools

A way to efficiently distribute the material for testing and to receive feedback is required, i.e. appropriate means of communication. Online and e-learning tools and a system to use and try the material are also needed. In case of e-learning material a ‘trace or log tool’ for the e-learning software is useful because the existence of many errors in certain training sessions easily indicate a problem with the material itself.

Embedding in Process Models

Acceptance testing of support material is neither mentioned in linear models nor in XP. Although RUP mentions the development of support material as an artefact, nothing is said about acceptance testing of it.

4.7 Delivery

The seventh and final stage is the delivery. Maintainers need to train the users, the package/release is prepared and delivered to the customers which then use the software. Internally the database needs to be updated. And the cycle starts again.

Installation at customer site

Purpose

The purpose of this activity is to install the software in the productive environment.

Input

Required is the ready-to-be-installed product as well as information about the productive environment.

Process

Before the product can be installed a release is created. That includes the operational product with all additional artefacts. The product is installed at the customer site and configured according to the requirements of the users. A back-up of the previous version should be kept in case undiscovered faults are detected. If necessary, data from a previous version of the system is migrated to the new one. If the data migration is more extensive, this can be done as an individual maintenance project.

Output

The result is that the release is completely installed in the environment where it is supposed to be used.

Role

The deployment manager plans and organises the deployment and is responsible for feedback that might arise from the installation in the productive environment. The implementer creates the installation scripts and the required related artefacts for the installation. The installation itself can be performed by the implementer, an installation or productive setting expert or even the end users themselves, depending on the product.

Tools

Back-up tools are required which support the creation and re-installation of a back-up version of the system.

Embedding in Process Models

In the Waterfall model the installation is part of the integration stage, in the V-model it belongs to the last step called ‘transition to utilisation’. XP, of course, also contains the installation at the customer site which happens quite often following the ‘small releases’ principle. In RUP ‘installation at customer site’ is part of the activity ‘test product at installation site’ but data migration is not included.

Final testing

Purpose

Final testing ensures that the software product runs at the customer site and that the customer accepts and approves it.

Input

Inputs which are required are the software product itself, a test model and test cases for final testing at the customer site. A workload model supports performance testing and the contract is used to compare whether the system fulfils the agreed criteria.

Process

After the activity ‘installation at customer site’ the developers themselves test the system if it shows the expected and required behaviour. Afterwards the customer tests the system to check if the contract has been fulfilled. The final testing can be identical with acceptance productive system test. That depends in what form and in which environment the acceptance tests have been performed.

Part of the final testing process is the approval of the customer that confirms the fulfilment of the contract. This is especially important for legal reasons.

Output

Test results and a running system are the main outputs of this activity. From a business perspective the approval from the customer is also very important.

Role

Roles involved here are the test designer, who designs and oversees the tests, and the tester, who is a user running through the acceptance tests. Also important is the deployment manager or project manager who oversees all activities at the customer site and receives the approval of the customer.

Tools

Depending on the scope of the tests similar tools as mentioned above under testing can be used.

Embedding in Process Models

Linear models include testing in general, final testing is not explicitly addressed. In XP there is constant testing also through the customer. Thus it can be said that final testing is somehow integrated in the process. RUP contains an activity that is comparable to final testing, although Kruchten only calls it a ‘formality’.

Notification of users/customers

Purpose

If the changes have an effect on users, they need to be informed about the modifications. Furthermore in some cases the awareness for updates has to be raised and customers need to be reached.

Input

The required input consists of information about the ready-to-be-deployed modification request and information about affected customers.

Process

Having received the approval to proceed, it needs to be determined whether some users need to be informed or not. In case of some internal modifications with no effect on users, such as modifications to improve the maintainability, no further action needs to be taken. Otherwise the users who need to be informed have to be identified. These users do not always comprise all end users, only those were the changes are relevant.

The users are informed about the changes using the appropriate means of communication.

Output

One output is the notification of users, a result of the process are notified users/customers.

Role

The role can be named notifier. This role can be taken by any contact person of the customers and the users who can reach them easily. Often this is done by the project manager.

Tools

A tool to support the first part of the process is a database to discover which users use what part of the system and thus will be affected. Depending on the contact to the users, different means of communication such as phone calls, emails, or intranet or internet news board can be used.

Embedding in Process Models

The Waterfall model does not mention any notification of users, the V-model indirectly includes it because the installation at the point of operation as the last stage also implies a notification. In XP it is also implied through the on-site customer who is aware of finished modifications. However, there is no comprehensive notification. In RUP it is partly included in “make product available” but this is also no comprehensive notification of affected users.

Deployment

Purpose

The purpose of deployment is to turn the software product over to its users.

Input

Apart from all installation artefacts similar information about the customers/user as in the previous activity are needed.

Process

To plan the deployment requires a high degree of customer collaboration and preparation. It can be decided that the deployment is identical with the installation at the customer site. Especially with a central system that is only used by one customer it should be one step. Furthermore a decision needs to be made when to deliver new versions to the customer, as quick fix or emergency repairs or later bundled with other changes to a new release.

Having decided that (at least some) customers need an update or a new version of their system, a deployment plan is set up how to get these systems updated. The software can be send via mail, it can be made available to download on a website or the system can be updated via a direct link to the customers' system. In case the customers are known the software developer's internal database needs to be updated as well, regarding which customer has received which version of the system. This requires a close link to configuration management.

Output

One output that is produced and also used during the process is a deployment plan. The final results of this process are updated systems of the customers.

Role

The project manager or deployment manager approves the deployment based on test results and customer's acceptance of delivery. Depending on the way of deployment a customer support manager, technical staff or a member of the dispatch department 'deliver' the software.

Tools

The deployment process can be supported through an application for online updates of customers' systems and a database that contains customer information.

Embedding in Process Models

Deployment is not contained in linear models. XP mentions related activities in the planning game and through the existence of an on-site customer. However, these are not proper deployment activities. With the exception of updating the internal database to store information about delivered versions of the system, RUP contains all relevant activities in the deployment workflow.

User training

Purpose

The objective is that the customers or rather the end users are able to do what they are supposed to do with the software. That is to install, operate, maintain and use the software, depending on their roles.

Input

Important input for user training is a system that can be used. This can be either a real or a test system. The first one has the advantage that all data and the performance are realistic, the second one that nothing can be damaged during training. Additionally information about the system, modified areas and affected processes are required.

Process

The information about the system and affected processes are used to develop the support material. This covers the information that will be required by the end user to perform their respective tasks such as installation, operation, maintenance and usage of the system. A close coordination with the activity 'support material acceptance test' is required.

Although the development of the training material is crucial, the emphasis of this activity lays on the user training itself. Groups of users where training is required need to be identified. Usually the users have to be split up according to their roles for the training to be effective and efficient. Follow up sessions or a close link to user support round up this activity.

Output

The result of this activity and thus the outputs are trained users who can use the system properly.

Role

Primarily the training material needs to be prepared. This is done by the course developer who plans and produces the training material and by the technical writer who is responsible for technical end user support material. Additionally the trainer provides the course itself.

Tools

User training can be supported through e-learning tools, online manuals, and other class room training material.

Embedding in Process Models

The linear models do not contain any training, neither does XP. RUP only contains the provision of training material but not the training itself as part of the deployment workflow.

4.8 Maintenance Management

Maintenance management is attached to the standard as an appendix. It includes, amongst others, management activities which need to be performed to organise the process. This belongs to the informative part of the standard.

Determine maintenance effort

Comment

In a recent paper [Sneed 2003] repeats his opinion that the cost of maintenance is better to estimate than the cost of new development. Although the likely return of reengineered software is smaller than of newly developed products, so is the risk. The return on investment can be much better calculated as there is less uncertainty, which is a huge problem of new developments.

However, this is only valid for certain reengineering projects. With maintenance in the long run it is hardly possible to know what errors will occur or how the environment changes. Nevertheless, it seems that many authors believe that the estimation of costs with a horizon of one year is possible.

Purpose

The aim is to determine the cost for maintenance for a time horizon of at least one year, i.e. the next budget round.

Input

Required are figures about completed maintenance projects. This includes not only financial data but also data about elements that needed modification as well as the source and reasons for modification requests.

Process

Using figures from the identification and analysis stages, especially from the ‘cost analysis/estimation’, and data about actual costs, evaluate the effectiveness of preliminary cost estimation. Extrapolation of current maintenance efforts assists the planning of future maintenance needs which can be used for the budget.

Additionally it should be determined what the costs would be if no maintenance is done, i.e. identify the benefit or value of maintenance.

Output

The results are current and future efforts/needs for maintenance and a budget draft.

Role

Roles involved are a project reviewer to evaluate the project data and an accountant to analyse the financial data.

Tools

Supporting tools are a database analysis tool to extract and analyse data about the system and a calculation application for financial planning.

Embedding in Process Models

Linear models and XP do not cover this aspect. RUP contains the evaluation of projects with the aim to plan for the future but not financial planning.

Manage maintenance process

Purpose

The purpose is to establish a (set of) maintenance process(es) that fit the organisation's and system's resources and requirements and support the maintenance team with adequate processes and tools.

Input

All kind of information about the organisation, its capabilities and processes, the systems to be maintained and experiences with current processes are required to manage the maintenance process.

Process

The starting point is to assess the current processes and tool support and identify the strong and weak aspects of these processes. Performance tracking supports the assessment and confirms results. Based on these results, developments in the market, and future needs of the organisation, the maintenance processes need to be adapted. Ideas of employees need to be taken into account for the adaptation as well as quality assurance aspects. Quality control/assurance for maintenance is not different than for development or other business processes. Every company should have those procedures and apply them accordingly to the maintenance process.

Different processes for corrective and other maintenance categories are likely to emerge because the triggers for the processes are different. Emergency maintenance can be treated as corrective maintenance, applying the same process more quickly.

Output

The output is a framework of adapted maintenance processes that suits the organisation and is appropriate for the modification of the systems to be maintained.

Role

A business process analyst or engineer adapts, determines and continuously improves the process and describes or models the process to be applied. A quality reviewer ensures that quality assurance aspects are observed. A project manager advises on the applicability of the suggested process and offers feedback for improvement. A tool specialist provides adequate tool support for the activities of the process.

Tools

The tool support for other activities is identified here and thus other tools have to be analysed. A tool really applied here is a business process modelling tool.

Embedding in Process Models

In linear models the management of the process is not part of the model. According to [Beck 2000] adopting XP means to adapt to XP practices. The systematic management of the process is not included in XP practices. In RUP the environment workflow has a similar purpose as this activity. Although the workflow has a strong project focus it also supports the organisation in adopting RUP for development and maintenance.

Manage maintainability

Purpose

The goal of this activity is to advocate, promote and ensure a maintainability of a system during new development, and to advocate, promote and preserve the maintainability during evolution and maintenance.

Input

Inputs for this process are programming guidelines, development and maintenance processes, and information about elements of systems where the maintainability needed to be increased afterwards.

Process

It is essential to create awareness for the importance of maintainability. Developers and maintainers need to understand that the efforts spent to build in and increase the maintainability of software as early as possible will reduce the required efforts in the future. Guidelines and standards have to be developed and they need to be part of the company's policies. Only then it is possible to control and enforce the standards. These standards need to contain criteria for extensibility and maintainability of software that can be checked.

Activities to prepare software for future maintenance need to be part of the company's policies as well as a concept for the transition from development to maintenance and from a development organisation to the customer organisation [Kajko-Mattsson 2001].

Output

Direct outputs are guidelines and standards for development and maintenance projects. Another result but not directly an output of this activity is the effect that systems being developed and maintained have a high degree of maintainability. Additional requirements and specifications for modifications can also be an output provided for other activities.

Role

A business process engineer is responsible for building in the adherence to maintainability in the development and maintenance processes. The system analyst and designer ensure that a high degree of maintainability is built in into the systems and the project manager promotes maintainability amongst the team members.

Tools

A tool that detects indicators for a decreasing maintainability through the assessment of e.g. code entropy or degraded integrity can be used to calculate a maintainability index.

Embedding in Process Models

The management of maintainability is not mentioned by linear models. Maintainability can be considered as one aspect of the XP practice 'refactoring' because one purpose is to increase the maintainability by reducing a system's complexity. The aspect of maintainability is only covered by RUP in the sense that extensibility should be observed when defining the system's architecture.

Configuration management

Purpose

Another issue that is relevant in the context of maintenance is that of configuration and change management [Bennett, Cornelius et al. 1991; Phillips 2000; Sommerville 2001]. The purpose is to maintain the integrity of the software especially when several people working on changes to the same software product.

Input

All project artefacts which are under configuration control as well as the modification requests are a part of this activity.

Process

To enable a thorough and efficient configuration management it needs to be ensured that all components are available. The procedures which need to be applied including naming conventions have to be precisely described. In addition to that it needs to be decided what versions will be maintained and supported for what time, indicating a close link to sales and marketing.

The configuration management process for individual projects consists of many parallel activities. All roles continuously modify artefacts and while doing so have to observe the procedures for configuration management. A role such as an integrator integrates artefacts and creates baselines which are descriptions of all artefacts that make up the system at any given time. Reporting and tracking of the overall project progress provides data for the current customer and for future planning decisions.

Output

The results are a configuration management plan that describes the policies and procedures to be applied during the project, a status reports that provides information about the project's progress, and reports on defects that can be used to determine future maintenance efforts.

Role

The configuration manager is responsible for the overall configuration management plan and oversees all activities concerned with the modification and integration of artefacts. The project manager evaluates information about the project's progress. All other roles have to apply the procedures specified in the configuration management plan.

Tools

Required is a configuration management tool that can handle the complex task of managing many artefacts which are modified by several users partly at the same time.

Embedding in Process Models

Configuration management is not contained in linear models although it is part of the overall development process specified by [BMVg 1997]. XP also does not contain this aspect. RUP covers configuration management within the configuration and change management workflow.

Manage people

Purpose

The human factor has been identified as the outstanding element contributing to the success of software projects including maintenance tasks [Bennett, Cornelius et al. 1991; Phillips 2000]. It is therefore important to decide who is responsible for maintenance activities especially when considering the fact that the status of maintenance is generally perceived as low [Kajko-Mattsson, Forssander et al. 2001].

Input

Needed are information about the projects' requirements and existing staff.

Process

To manage people involves a couple of aspects that need to be taken into account. The appropriate staffing of teams is the first aspect to be considered. It needs to be determined what skills are needed and which level of knowledge is required. [Bennett, Cornelius et al. 1991] write that a mix of experienced and inexperienced staff is an ideal team. However, maintenance is not a training ground with promotion to development for good performers because this implies that maintenance is a punishment for bad performers. Therefore a rotation between maintenance and development and between projects is required. While the composition of a team is important, it should not remain the same for too long. Job rotation offers the chance for developers to work on different projects.

If the development team is identical with the maintenance team they will make the software maintainable for their own good and less documentation is required. The advantage of different teams is that the development team can move on with their expertise to new development projects. This is a trade-off and at least some people from the development team should be a member of the maintenance team if possible.

[Kajko-Mattsson, Forssander et al. 2001] stress the people factor that it is crucial to educate and train the maintainers even better than developers (if these functions are separated.) It is important to train them in the technology they need, not the latest fashions if they do not need it.

Output

The direct output is a human resource plan that covers staffing, training, job rotation, and team leadership. Indirect but also desired results are a low labour turnover, motivated employees and skilled people who do their job properly.

Role

A human resource manager and the project managers concerned plan the allocation of staff and identify their training needs whereas all other roles involved provide feedback.

Tools

To manage people only planning tools for resource allocation are required.

Embedding in Process Models

Linear models do not cover the aspect of human resource management. XP mentions pair programming and regular rotation but no other aspects of this area. RUP contains guidelines for staffing but explicitly excludes areas such as training and coaching.

5 Case Studies

This chapter contains the results of six interviews which I carried out myself[6]. An interview check list can be found in appendix II. It contains an approximate order in which the questions were asked during the interviews. The results are presented as follows:

Under the heading **Background,** general information are provided about the respective company and a description of the department or project team from which the interview partners come from. To determine the context of the maintenance process some statements about the importance of software for the company and their software development process in general are added.

The interview partners' definition of maintenance is the first point under the heading **Maintenance**. Afterwards their comments about my definition of maintenance (cf. chapter 2.1) can be found. An outline of the organisation of the maintenance function comes next, followed by a description how maintenance is embedded in the company's processes. The main point in those sections is a description of the companies' maintenance process. This description is derived from the interviewees' example process and the additional comments made about the extended IEEE process. To ease readability subheadings referring to stages of the maintenance process are inserted in the text. Further comments about the maintenance process complete this section.

The heading **Assessment & Outlook** first contains an assessment of the interview partners of their companies' maintenance function and activities. This complemented by the critical factors/activities for maintenance from their point of view. The final point of all case studies is an outlook about desired and expected changes in their area of work.

5.1 Case 1 – IBM

Interview Partner	Rainer Gimnich
Company/Department	IBM Global Services
Category	International service provider
Place of interview	Koblenz
Date of interview	15.10.2003
Duration of interview	125 minutes

<u>Background</u>

Company Information

Rainer Gimnich works for IBM Global Services in the area of Business Consulting Services. He describes the maintenance process applied by the department Application Management Services (AMS). This department delivers long term solutions for the whole life cycle of applications in the areas of organisation and processes, development, maintenance and support, and provides application development and management, and application life cycle management services.

[6] Additional information not presented here can be found in appendix III.

They apply the same global methods all over the world for their customers from all areas such as industry, trade, finance, administration, and from small companies to international corporations.

Importance of Software

As the name suggests, applications are the core of their business. They provide maintenance and maintenance consulting including the design of maintenance processes. AMS maintains IBM software products and software products from other suppliers, standard software and individual software solutions. Their service is provided for internal and external customers.

Development and Maintenance

Software development and extensive further developments are separate areas. These activities are not performed by AMS. It is interesting to note that development is mainly concerned with further development and reuse of existing system. Really new developments rarely occur. This is only the case with 'new ideas' such as Enterprise Resource Planning (ERP) systems.

The interface between development and maintenance is clearly defined. During development, the developers at IBM create an artefact called Application Management Turnover Package (AMTP) for AMS. This AMTP contains the source code, documentation, test cases and an impact analysis (impact of requirements on current release).

The AMTP is being used and further developed by AMS. As further development will usually continue with multi-release projects, AMS will hand over the modified AMTP to the developers before the next release is finalised and deployed. Both areas are kept up-to-date using this package.

Maintenance

Definitions

AMS uses the following categories of maintenance:

- "Maintenance" which includes corrective and emergency maintenance. All activities of a size up to 15 man days or 100 hours without changes of functionality or modifications of the architecture belong to this category.
- "Enhancements" are further developments without major modifications (outside the 15 man/100 hours limits).
- "Major enhancements" are bigger than "Enhancements" but still without major modifications to the software architecture.
- "Perfective maintenance" is primarily of a technical nature, including tuning and the improvement of maintenance on code level.

Rainer Gimnich did not provide his own definition but agrees with my definition (cf. chapter 2.1), however, he interprets it differently in the sense that he would exclude the addition of new functionalities. Otherwise he thinks that my definition corresponds to the categories of AMS.

The Maintenance Function

As mentioned above, maintenance is separated from development, but all employees have similar background knowledge. However, the AMS staff do not need to have the same business skills or domain skills because they do not change (in the case of maintenance) or only slightly change (enhancements) existing functionalities.

The development/evolution of most software products is quite stable within a foreseeable timeframe of approximately three to five years for development and further development, and five to ten years for maintenance without bigger modifications.

The experiences from many projects have been collected and condensed over years and a clearly defined method for maintenance has been established. This method is an asset for IBM and is neither published nor for sale. It is applicable for all AMS employees around the world, although it can be adapted to take national peculiarities into account. Sometimes special methods are developed for bigger projects, such as the introduction of the Euro. A description of this example can be found in the appendix.

The Maintenance Process

The general maintenance process for all maintenance categories at AMS is as follows:

Identification

The process starts with the identification of a modification request. Reasons for these requests can be for example legal changes, business needs or other customer requirements. The requests can be transmitted to IBM via all means of communication such as telephone, email or internet. Decisions and instructions from top management can influence the following process but usually the requests remain on the level where they have been received.

The above mentioned categories are used for classification. No customer requests are rejected, although the actual implementation depends on contracts and prices.

The initial assessment about the magnitude of the modification is very detailed, which is also important for contracts. Equally important for a decision how to proceed are potential costs and benefits which are evaluated early in the process. Different severity levels are used for prioritisation. Based on all the information the implementation schedule is continuously adapted.

Analysis & Design

During the next stage the requirements are identified and specified. Functional details are determined. For special areas, some tasks are delegated to experts outside the project team at the project leader’s discretion.

A problem at this stage is that often exact legal specifications are changed or fixed late in the process, e.g. the decision if ISIN is numeric or alpha-numeric or when the Euro conversion will take place. In those cases it is necessary to find ways to work around it.

The technical migration is planned which includes the evaluation of alternatives, the determination of tools and responsibilities. Furthermore the impact of the requirements is updated in the AMTP.

Implementation

The implementation phase starts with an analysis concerning what is affected. This analysis is very detailed and looks at aspects such as individual fields and variables.

The analysis of IBM products is supported by the AMTP which contains the artefacts required for the analysis. In the case of third party products most often only the source code plus some unstructured documents are available, rarely everything. Sometimes not even the complete source code exists. In those cases the analysis starts at the source code level because trust in third party documents is not good enough for maintenance. Nevertheless, the additional documents are also evaluated for future use.

Other standard activities of the analysis phase which are applied in parallel to other activities are program understanding, detailed impact analysis and cost estimation. Efficient tools support this process, especially for third party software in the area of effort estimation, e.g. how much logic is contained in the code. For the implementation on code level a source change assistant is used, which is a tool for standard software.

Testing

The planning of test cases as well as the identification of documentation changes and the updating itself is done as they go along.

Unit testing is kept at a minimum; it should not be done for corrective maintenance because it is quite time consuming relative to fifteen man days projects. Maintenance should correct errors in units without major modification and therefore unit testing in order to look for other errors should not be necessary. If unit testing is required it is done by the programmers who implemented the changes.

The responsibility for testing depends on the project size. In smaller projects testing is done by the programmers themselves, whereas bigger projects have dedicated test teams. In some cases, such as the Euro introduction, the program is transferred to the original developers of the system who do the testing.

Testing consists mainly of integration and system tests. Test tools are important and are being used extensively. Approximately one third of the tools are IBM tools, the others are third party tools. After all tests have been performed successfully the program is approved for productive use.

Final tests are done as acceptance tests, which include integration, operability and acceptance tests. Some of these are partly performed at the customer site. However, this is not valid for maintenance tasks. If a reported error has been removed the program can be tested by the developers. Only in case of major enhancements where the menu has changed acceptance tests are done by potential or real users.

Delivery

As the customers are known they are easily informed about updates during deployment. User training is usually not required for maintenance; in case of (major) enhancement it is part of the method.

Maintenance Management

Due to the size of most applications configuration management is an essential process component. It is an established process based on version control of the affected components such as software modules, test cases, and documentation.

In case of modifications those components, with use of versioned architecture descriptions, are combined in a new version of the system. This is propagated through several steps until an error free and complete version of the system is ready for operational use.

Additional Comments

This process is mostly automated. To support this process IBM uses a Source Configuration and Library Manager. If maintenance is done at customer sites, the maintainers use existing products, often Endevor from Computer Associates.

The final step in the maintenance process is feedback about the method which is part of the method. This compulsory feedback is called 'harvest' and requires that the experience with the method is communicated to a central method-team in the USA. The feedback can range from a short note to detailed descriptions.

In the USA the method is adapted according to the feedback. The method team consists of developers with practical experience who do this job for some years before they go back into practice. Many of them return to the method-team after a while. Most of those who develop the methods are computer scientists or at least are experienced in the IT area. Academic knowledge is also included in the methods.

An additional comment was that software systems which do not require maintenance do not exist provided that enhancements and perfective maintenance are considered as modifications which are defined as maintenance. Maintenance will come for every product and thus the AMTP is expected by AMS. Issues of maintainability such as extensibility are also taken into account during development.

Reengineering does not exist at IBM as a separate activity/phase/method but is included under other headings. Sometimes projects have the unofficial title reengineering so that people know what happens.

Assessment & Outlook

Assessment

Using the increasing number of customers as an indicator, the quality of their work must be good. Most important for customers, as Rainer Gimnich sees it, is the cost-benefit ratio. In this area IBM is able to make good offers. This has several reasons:

IBM has well trained employees and specialists for many activities. Those can work on several projects in parallel as a compensation of peaks in certain projects. Another reason for their efficiency is the high data processing capacity which he considers still as an obstacle in many companies.

Also essential is the high degree of automation. Methods and tools are available. Thus maintenance can be done quicker by IBM than by the customer itself even if the customer knows the software better but does not have the tools. Especially important for

IBM’s efficiency are analysis and test tools. Only in the area of reverse engineering is still some room for improvement.

Outlook

Rainer Gimnich would like to get more requests and ideas for modifications from customers and users. Feedback with the exception of error reports is on a very low level. He supposes that customers often do not know what is possible and take the product as it is.

The department AMS is in a state of organic change and grows with changing requirements. An expected change is that reverse engineering will be more important in the near future and will be embedded in IBM’s processes.

After the acquisition of Rational through IBM in 2003 the methods contained in RUP and the methods of IBM will be combined in the near future to make use of the experience and tools of the other party. The goal is to have homogeneous and the best methods for all IBM employees.

5.2 Case 2 – Engineering Company

Interview Partner	--
Company/Department	Engineering Company[7]
Category	Embedded systems
Place of interview	---
Date of interview	October 2003
Duration of interview	45 minutes

<u>Background</u>

Company Information

The interviewed person is the software development team leader of a German Mittelstand company (comparable to SME – Small and Medium sized Enterprises). This company is a machine manufacturer, still owned by the founder family in the sixth generation. Its customers are industrial companies from Europe, America and Asia.

The software development team consists of six people; all of them are either mechanical or electrical engineers. In a way their customers are the hardware, i.e. the machine developers, whereas the users of the software are the company's customers.

Importance of Software

The machines, including its control software, are developed completely in-house as a whole system, which means that no compromise has to be made to make parts fit to the equipment of other suppliers. The software is an inseparable part of the machines and belongs to the category of embedded systems. Software and hardware are equally important and thus high quality software is crucial for the success of the business.

Development and Maintenance

The original software product was developed around ten years ago. Since then it has been continuously modified in variations with regard to functionality and to fit to different models of the machine.

Thus their software development process is also their software evolution or software maintenance process. It is an incremental process which was not formally set up but has been established internally over time.

<u>Maintenance</u>

Definitions

The interview partner defines maintenance as the correction of errors, improvement of efficiency of the system and the extension of existing functionality. In contrast to that is the adding of new functionality for an existing product.

My definition is too extensive for him because according to this definition they only do maintenance and no development. He does not like the term maintenance at all and he also finds it difficult to distinguish between maintenance and further development. But

[7] The contact to the company was established using personal contacts.

the definition does not matter at all as his team does everything related to programming anyway.

The Maintenance Function

Every team member has his area of expertise for certain functionalities and is responsible for all changes within his area of concern. There is a constant dialog between all six programmers, the hardware developers and management.

The original product was developed with support of computer scientists. A major requirement was the extensibility of the product and therefore the architecture was especially set up to support that. Now there is no development of a new product anymore and they do their job without any computer scientist in their team.

The Maintenance Process

He describes their maintenance or further development process as follows:

Identification

The first phase is called project initialisation. All kinds of ideas, i.e. customer requests and complaints, own ideas, and hardware developments, are collected informally through all channels available. Together with the company's management the broad direction is agreed upon. The details of the priorities are fixed in discussion with the development team.

The first judgement is usually to accept or reject a request. Valid, i.e. accepted requests, are put in categories, which are: urgent repairs, other errors, extension of existing functionalities, new functionalities. This leads automatically to an allocation of tasks to team members according to their expertise.

The efforts of the individual tasks are estimated and prioritised. The output is not a detailed plan but a list when what request is going to be implemented and by whom. The time horizon ranges usually from now to half a year. Requests with the time slot 'sometime' have bad chances of being implemented at all.

Most changes belong to the category of new functionality, hardly any errors are reported. Perfective maintenance is done in parallel to the normal requests. If somebody works on a functionality where the code needs perfective maintenance it is done if time permits.

Implementation

Most activities of the process are done during the second phase called implementation. Here every developer considers alternatives for his tasks for himself. The process is rather unstructured and only a few guidelines exist. As only one person is working on every request more structure would slow down the process. All activities are performed manually without any tool support.

A feasibility analysis is not necessary because the decision about the costs, benefits and realisation has been made before. A detailed impact analysis is also not considered as necessary because it is assumed that every developer knows his area by heart.

Analysis, design and implementation activities are more or less done in parallel. The developer implements requests according to the priority list, the programming language used is C++. Test cases including regression tests are also decided upon and developed in parallel to design and implementation.

After successful implementation the developer integrates his components into the system and does the testing for his area. They have the knowledge for their area to do that and it also increases the motivation as it gives everyone more responsibility.

The interview partner said that all relevant artefacts exist. In their case this means mainly the source code, the knowledge about the machines and little documentation. With regard to the last aspect, i.e. documentation, this is done only in very general terms so that the developers know what they have done and why. Other team members will be able to understand that with little effort, external people do not have a change to understand it at all. The reasons are the low fluctuation in the team and the desire to keep overheads, which includes time for documentation, at a minimum. There are no tools for the documentation that is done; everybody can make notes in the form he considers as appropriate.

Testing

Testing is the third phase. The programmers themselves perform only a few tests so that the program can be transferred to the machines. This is also a transfer to practical people, i.e. the hardware developers, which do testing with the machines in real situations to check the most critical functions which could damage the machines if malfunctioning. Testing is done internally only; there are no external acceptance tests. As 100% testing is not possible, sometimes customers find errors but only rarely.

Delivery

Their fourth phase is deployment, which happens continuously. As a rule it can be said that updates for new machines are released every two months, updates for machines at customers sites are released yearly. Those updates can be bought separately or are included in maintenance and support contracts.

As the customers are all known to the company, information about important updates are sent to affected companies because not all updates are relevant for all customers. Corrections of critical errors are deployed immediately, smaller errors with the next update usually without any comments, and new functionalities with the next release together with relevant documentation. General training is provided if required.

Additional Comments

In his opinion the extended IEEE process is very (too much) structured. They do it much more flexible. As they are a small team every team member has much responsibility how they do their job, i.e. output oriented instead of process oriented.

With regard to maintainability he admitted that the code degenerated over time but they try to improve the quality now.

Assessment & Outlook

Assessment

The assessment of the interview partner of their process is as follows: Based on customer reports about errors, which rarely occur, he judges their quality of work as good. As most requests are implemented quickly he also considers their efficiency as good. The overall mark he would give to their work is ‘very good’.

The basis for this positive achievement is the *“good fit of everything”* and the clear process. Their budget is also sufficient for their task because the management is aware of the importance of their work. Other reasons are the small overheads and highly skilled employees which familiarised themselves with the system. Their responsibility for their product also increases their motivation, leading to positive results.

The central activity is the prioritisation of customer requests. Due to the fact that the company owner and manager developed the first machine and that all software developers are engineers themselves the evaluation of customers’ needs is very effective. Also named as crucial is the testing of the machines. The software and hardware developers know the machines by heart and can do the testing effectively and efficiently and are aware of the critical functions.

Outlook

Since the team works very well the interviewed person does not expect any major changes in the near future. The only area that might change is the use of tools. There might be more tools and more structure in their future job but for applying those, a computer scientist would be needed. And so far they can do without one.

5.3 Case 3 – VW

Interview Partner	Eike Escheberg Schröder
Company/Department	VW AG, IS Sales
Category	Management
Place of interview	Wolfsburg
Date of interview	21.10.2003
Duration of interview	90 minutes

Background

Company Information

Volkswagen (VW) is a car manufacturer with its headquarters in Wolfsburg. The interview person works for an IT service department called 'IS sales' which provides IT services for marketing & sales (for a more detailed description of the structure cf. appendix III). This department works on the basis of contracts to provide a service to VW marketing & sales and partly to the marketing & sales departments of other affiliated companies.

'IS sales' provides consulting and acts as a service provider for the sales division of VW. They have the organisational status of a profit centre but their target is not profit maximisation but to cover their costs. They work on the basis of contracts in the form of service provision agreements. Their service is provided with project character, i.e. limited with regard to time and budget which are fixed in the contracts.

Importance of Software

The applications, for which they are responsible, in this case sales applications only, are considered as crucial for VW's business, e.g. direct sales online to large dealers, online links to dealers and importers.

Software or rather sales applications are the focus of their activities. However, they do not do any development, they only provide management and consulting services. Their customers come from the sales division; the users of their applications are VW's customers, in this case dealers, large buyers and importers.

Development and Maintenance

An important element for their work is the VW standard SEP – System Development Process. SEP conformity is required for all projects; otherwise projects do not get the approval from 'Controlling'.

The five phases of this process are: inception, conception, system design, system implementation, system deployment. This process is required for all IS projects including service projects, that means also for all activities of 'IS sales'.

Maintenance

Definitions

His definition of maintenance is short: modification of an existing system or in other words the path from actual status to desired status using the formula ‘who has to do do what till when with what means’ (which can be translated to: role, activity/output, deadline, methods/process). He admits that this is a very general and extensive definition but it is necessary to cover all related activities. An additional reason is based on company politics. Maintenance/service projects are considered as necessary and belong to the last projects were the budget is being reduced massively or which are being cancelled.

My definition is definitely more detailed than his. He did not think about changes in the environment. He agrees with my definition and finds it suitable for their maintenance projects.

The Maintenance Function

The structure of the IS departments and their relationships to their suppliers and customers is complex. Technical development is done externally supervised by ‘IS sales’. After the technical development the software is transferred to VW; to ‘IS sales’ for general support and for operations to ‘IS technology’ which are responsible for the hardware side. (A more detailed description of the relationship between developers and ‘IS technology’ to ‘IS sales’ can be found in appendix III.)

From the life cycle perspective of a software product it is important to note that after the transition from development to operations all products further developed and maintained. After a while, sometimes months, years or even decades, the products are only maintained to keep them operational. There are no additional investments in further development. This happens usually in case of a migration to a new system or when the system will cease to exist soon for other reasons.

‘IS sales’ has an application oriented personal strategy. That means that one team is responsible for an application or a set of applications for development and maintenance. The team members have a diverse background such as computer scientists, electrical engineers and business people. They mainly need to have technical knowledge and only little expertise in the application domain, i.e. the sales area.

The Maintenance Process

Eike Escheberg Schröder describes the maintenance process from his department’s perspective as follows:

Identification

Modification requests are received via email, telephone or through personal discussions mainly from employees from VW sales. A web interface for recording those requests supports this activity. The tool is a development of ‘IS sales’ and its usage is currently limited to their department.

Modification requests are often based on a strategy and lead to necessary activities. An example is the strategy to increase direct sales to large buyers. A way for achieving that is to improve the online connection, which means improving existing applications.

Other requests come from ‘IS sales’ staff. Those requests are often concerned with the improvement of maintainability. This is worthwhile if fixed service contracts exist. In those cases an improved maintainability reduces the maintenance efforts and increases the profit. If fixed contracts do not exist the improvement of maintainability is suggested to the client who has to decide if they want to pay for it. Generally the issue of maintainability is planned for during development and is taken care of during operations.

On the division level four categories for IS projects are used. These are related to the budget priorities, which are in descending order: legal necessity, service, strategic, and further development.

‘IS sales’ needs only two categories for change requests. The first one is error correction for tasks that consume less than thirty man days and are paid from the service budget. The second category is for further developments/new releases. All tasks which will consume more than thirty man days are put in here. These projects are paid for from the development budget.

Analysis & Design

The next step in the maintenance process is the analysis. Based on source code, user manuals and experience the staff from ‘IS sales’ establish a list of necessary steps to achieve the given objective. They estimate the time it takes to implement it and plan how this can be done. In case of error messages no cost estimate is provided for the sales division because error correction is included in the service agreement.

The first assessment provides a recommendation for implementation including a risk analysis and related follow up activities. The decision to proceed or to abort the project is made by the sales division; ‘IS sales’ cannot decided.

If it is decided to proceed the task is included in the change plan and an agreement is fixed about the time horizon with ‘IS technology’, external developers and their client.

Implementation

Implementation can be programming, the migration to a new platform or only setting up of a configuration. This is done by ‘IS sales’ together with external development partner. All programming tasks are performed by an external party, supervised by ‘IS sales’.

Testing

The first technical tests are done by the developers. When a test version is ready this is transferred to the client for testing. The client is taking parting in testing as they have the required domain knowledge which does not exist at the external development partner or ‘IS sales’. Acceptance tests are executed by the sales division and the users of the system if those are not the same, e.g. dealers and importers. All testing is done manually with no tool support.

The client has to give consent and then the application is installed in a quality assurance environment, which infrastructure is identical to the productive environment. Here the application is tested and approved by ‘IS sales’ and the client.

Delivery

The release is then set for fixed day and the software is transferred to 'IS technology'. Most applications are web based solutions because a world-wide client/server roll-out is too extensive. The tool used for version control is MS Visual Source Safe.

If they provide any training at all, they train the trainer and do not offer any direct user training.

Additional Comments

As a final comment it can be said that they follow the extended IEEE process in an abstract way. The process cannot be applied precisely because of the unique structure of the IS division of VW.

Assessment & Outlook

Assessment

In school marks the interview partner would give his department '2-3' for quality because he is not aware of any bigger problems or complaints. Much worse, from his point of view, is his department's efficiency for which he would give the mark '4-5'. The reason for this is the split responsibility; it is unknown who is concerned with applications through all layers. For example 'IS technology' does not know what applications are running on its hardware and 'IS sales' usually only sees the top level, the application layer.

The factors with the highest impact on his assessment are the following: the estimation of total efforts and the size of tasks/projects during the planning phase are good, based on experience and thorough analysis. Another positive factor is the existence of designated contact staff within 'IS sales' for customers from the sales division.

On the other hand the organisation from the application level to the hardware level is moderate to poor. Unknown contacts resulting in communication problems lead to the above mentioned inefficiencies. Also a negative impact has the little documentation. Especially reasons for changes are not sufficiently documented. Partly only the latest version is kept and for this reason the change history is lost.

Outlook

A desired change for his area is more and better tool support. However, he does not expect that the desired tool support will come in the foreseeable future because the current cost-benefit ration is neither acceptable nor transparent enough for the management.

One change that will be implemented is better documentation of changes. This has been identified and realised as crucial for future maintenance and will be supported by MS Office applications.

5.4 Case 4 – sd&m

Two interviews were conducted at different branches of sd&m. The results are presented as two cases due to the major differences between the two interviews.

Interview Partner	Marion Alfter, Johannes Pellenz
Company/Department	sd&m
Category	Custom-made software solutions
Place of interview	Troisdorf
Date of interview	22.10.2003
Duration of interview	120 minutes

Background

Company Information

The fourth company, where I did two interviews, was software design & management, called sd&m. It employs some 900 staff in Germany and Switzerland and plans to expand to Poland and is part of Gap Gemini Ernst & Young.

The branch in Troisdorf is responsible for the region Cologne/Bonn. sd&m has regional centres which are usually specialised in certain areas; e.g. Frankfurt/Main for finance, Munich for automobile and insurance, Cologne/Bonn for mobile communication and health services.

sd&m develops custom software for business information systems and technical applications and provides consulting for issues related to information technology. Their development and consulting services range from problem analysis and specifications via construction, implementation and testing through to maintenance. These are also the phases of their development phase model which is in contrast to an evolutionary (in their view: prototyping) model.

Importance of Software

Software itself and software related services such as maintenance are the products and services which are sold by sd&m and hence it can be said that the software is the centre of the company's business activities. Their customers are predominantly external; development for internal usage is minimal.

Not always are all tasks done by sd&m. Sometimes they do the specifications only or do the realisation based on third part party or their own specifications. Nevertheless, their favourite order is to do the complete development.

Development and Maintenance

Development and maintenance activities are carried out in the form of projects and every branch runs several projects in parallel. The team size differs depending on the projects.

sd&m has a handbook containing software engineering methods which are applied companywide. However, the methods described are handled flexibly in case the customer has its own specifications how to proceed.

According to the interview partners in Troisdorf, the method handbook also contains general instructions for maintenance and restoration projects. These are based on the development process but not part of it.

Maintenance

Definitions

They distinguish between development and maintenance where maintenance includes error correction, modification of existing functions and enhancements, all based on change request. Reengineering and further development are usually contained in maintenance with the exception of some separate reengineering projects to make products maintainable. In case of bigger modifications where all phases of product development are applied again this is considered as new development and is handled accordingly.

My definition seems to them rather ‘academic’ and very detailed but still applicable/adequate.

The Maintenance Function

Since they distinguish between development and maintenance, the development process is separated from maintenance what they perceive as positive. At sd&m in Troisdorf it is common practice that during the initial development of a product upcoming change requests are blocked or stored. Afterward during operation the change requests are implemented step by step.

Maintenance is event driven in the sense that something, i.e. an event, needs to happen for maintenance to be performed. Their rough estimate is that of their maintenance tasks 25% are concerned with error correction, 25% with enhancements and the remaining 50% with support. In their context support includes configuration, advice and error tracking but excludes all changes to the product. This is considered as 2nd and 3rd level support where the customer’s help desk cannot help their users and asks sd&m for support.

The personnel involved in maintenance projects are the same from the product development but less people. Employees doing maintenance have to know less or in same cases even nothing about the customer’s business processes and the system architecture. However, maintainers need the ability to suffer and their job is much less exciting than development.

The Maintenance Process

The maintenance process as it is applied at sd&m in Troisdorf is as follows:

__Identification__

The users report occurring problems to a central authority in their company which in turn enters the reports as problem tickets (PT) in a database. At this step there is no filtering, i.e. all problems are registered.

A PT contains information about the development team, the version and application, the priority of the problem from a user perspective, and whether the problem occurred in a test or productive system. An automatic notification about a new PT is send via email to the concerned development team, which is stored in the system.

The triggered process depends on priority and source of the problem. The priority is evaluated again, this time from a developer perspective. The classification and prioritisation is 2-dimensional. First the PTs are classified with the classifications of error, support and modification and afterwards a priority is assigned. The priorities are one, two or three whereas three is so unimportant that it will not be implemented at the current budget situation.

At this stage it is also evaluated if the PT is really based on an error or not. If it is not an error and requires certain effort to implement it is considered as change request. Here the customer has to decide about the implementation. This is done by a 'Fachteam', an 'application team' of the customer, which also prioritises the PTs quickly.

The next step is release management which is concerned with the issue what should and what can be implemented until when. The release planning includes all products and differentiates between two kinds of releases. The first category is an error correction release and deals with the question which errors are corrected within which release. In contrast to that is the non-error correction release. Even if errors are know, they are not corrected. This happens in cases such as the implementation of a new Oracle release. No previously known errors are corrected and therefore new errors must be due to the new part of the system. The status immediate action means that the error is corrected and the software is delivered immediately.

As mentioned before the source of the error is also one aspect that relates to release planning. If the error occurs in the quality assurance system the error is corrected within that release before it is productive. If an error occurs in the productive system the correction takes place within the next release where one is allowed to take part. The tracking of the status of problem tickets and change requests is supported by a tool.

Part of the planning process is the estimation of the efforts and what can be done before the next release. Based on that, the project plan is established or updated. It contains the following: In case of change request specifications are determined. Problem tickets are revised and information about the time and responsibility of realisation, testing, and the transfer to the customer's quality assurance are added. The tool being used is MS Project. Status information set or changed in the project plan are also updated in the PT database so that the customer is aware of the current status.

Analysis & Design

When the developer is determined he can start to work on the maintenance task. The first step is to describe the error, the reasons for it and the measures how to correct it. The project leader only checks the description if the problem is not trivial. Representatives of the customer are sometimes consulted especially if an agreement is required. This is usually necessary in case of change requests.

For analysis Johannes Pellenz uses the following artefacts in this order: technical data, sd&m's error database to search for similar errors, and the source code. The impact of modifications is not identified with regard to errors. In case of change request it is identified what the system can (not) do afterwards. Part of the analysis is the evaluation of alternatives. In 5% of all cases a recommendation containing pros and cons is provided. If the analysis is concerned with an error the developer decides for himself. In case of change request at first a rough estimate about expected efforts is established, followed by a decision of the customer. After a detailed analysis the customer has to make a final decision whether to proceed or not.

Safety aspects are only considered when a change request requires changes to the underlying concept. However, more important than safety are performance aspects.

The analysis of short and long term costs is rather limited. The customer should know and state what they expect. sd&m provides an offer and consults on the alternatives. With regard to benefits the situation is similar. The application team of the customer has to make a decision if the implementation is worthwhile. Nevertheless, one customer allows everybody to hand in change requests which are evaluated on the basis of a point system before a decision is made.

During the detailed analysis the requirements are determined and elements to be changed are identified. A general test strategy as suggested by the IEEE process is not developed because sd&m has one test strategy per project. Activities like program understanding and impact analysis are relevant for several stages; their execution depends on the PT and the developer.

In the case of one customer some aspects of analysis and design are mixed and sd&m adapts to the customer's process. Otherwise the contents of documents concluding the different phases are separate at sd&m.

During design the specification of the elements to be changed are adapted based on the results of the analysis. Documentation changes are partly identified and partly updated immediately. The user manual is going to be changed at a later stage but the necessary changes are identified here. Often the user manual is only valid with the change request documentation as a supplement.

Implementation and Testing

The implementation including unit testing is done by the developer who also did the last steps. The time of integration depends on the priority. Documentation in the PT database includes what has been done and the reasons why.

The code review is done by someone else than the developer; possibly the one who also did the analysis but different from the one who did design and implementation. The same applies for testing which is not done by the developer who implemented the changes. If the problem is not trivial test cases are used, otherwise the PT description is sufficient. Regression tests are also performed, supported through a database but still executed manually. They have tried different tools at some stage but the cost-benefit ratio was not convincing because the manual programming effort for new test cases and new/other environments was too big.

When the necessary tests have been passed the software is delivered to the quality assurance of the customer. Pilot tests are run at the customer site, i.e. the system runs in a limited environment for testing. Furthermore acceptance tests are performed.

The quality assurance of the customer must accept the release, sets the status in the PT database to okay and writes a report for approval.

Delivery

Deployment happens through a dedicated deployment team. Orders for deployment are based on change requests and PTs and are checked if the status is correct. After a final approval the software is release to the productive system of the customers. Possibly required user training is done by the customers themselves, partly with support from sd&m.

Maintenance Management

Criteria which are included in the development plan are extensibility, performance, ergonomics, and maintainability. These criteria are supported through the use of components. Optimisation at a later stage is discussed with the customer, the implementation is release driven. In the case of one customer, who has many self-developed products, restoration projects are required to make them maintainable.

Configuration control is always an important issued which is supported by Visual Source Safe and ClearCase.

Additional Comments

An important basis for their work is the error database which contains the PTs. The customers have read access. sd&m in Troisdorf also has an additional database for internal PTs which the external customers cannot access.

Also important are reverse engineering tools. One example given was the function to convert an Oracle program into an HTML description. Code analysis tools are not used because the development environments are very heterogeneous. An additional comment stated that some employees use their own tools, especially for testing, which are partly shareware tools.

The final comment about the extended IEEE maintenance process was that is academic but not bad.

Assessment & Outlook

Assessment

Based on the results of the last customer survey they judge the quality of their work as very good. Although they spend much effort on quality assurance they also consider their efficiency as good. It should be mentioned that their overall performance was improved in recent times through a clear test concept.

The main impact on the maintenance process has the system itself which is to be maintained. For example a grown system is often hard to maintain and consolidation/reengineering is not possible for budget reasons.

Although they do not consider it as relevant in particular for maintenance personnel they both value the good training in new technologies that is provided for all employees. Members of staff usually belong to a project team for two years before they are moved to other projects. This increases the motivation.

Outlook

Changes which they would like to see in the near future but which they consider as unrealistic are to have only clean and clear code, to have proper documentation for all programs, and to have automated test tools for source code which require less effort than manual testing and where the configuration is so easy that it is quicker than manual testing. Furthermore they would like to have an integrated system tracking and tracing of change request and release planning.

In contrast to their wishes both of them expect that none of the issues mentioned before will be fulfilled in the near future with the exception of improvements regarding tracking and tracing within one system. The reasons given for the difference between desired and expected changes are that no super tool exists that can cope with such heterogeneous environments and that all customers are different. Legacy system will be in place for a long time because there are no budgets to replace them.

5.5 Case 5 – sd&m

Interview Partner	Buu Tran Van, Dr. Ulrich Kühler
Company/Department	sd&m
Category	Custom-made software solutions
Place of interview	Ratingen
Date of interview	28.10.2003
Duration of interview	110 minutes

<u>Background</u>

Company Information

A general company description can be found above. The branch in Ratingen is responsible for the area around Düsseldorf.

Importance of Software

The first statement they made was that they do not do any maintenance, only software projects. At least maintenance is not their core business. However, no manager would say that they do not do maintenance. The centre of their business is clearly development; the preference is consulting instead of maintenance. Maintenance projects often result from development projects where they keep at it.

Development and Maintenance

Four years ago there was no maintenance, now there is some. How this exactly looks like in the company handbook is not know by the interview partners. What is important for their job is the quality assurance system which is represented in the handbook. It is based on years of experience and contains methods which include roles, processes and results. However, the prescribed methods are handled flexibly and are adapted to the customers' needs.

<u>Maintenance</u>

Definitions

The two interviewees did not want to give a definition of what maintenance is but rather what maintenance is not: maintenance tasks which take than two weeks and require extensive modification of an existing system are more than maintenance; those are projects with conception, design, implementation and test. Individual maintenance tasks are not performed but every individual project contains many different tasks. sd&m

provides the maintenance team which carries out the necessary activities based on a contract for the whole project.

Although they did not provide their own definition they said that my definition of maintenance is quite comprehensive, they have nothing to add and they would like it.

The Maintenance Function

Maintenance is not popular with the employees. However, it is considered as an ideal starting point for beginners, i.e. for new and inexperienced employees to get insight into practice. They do maintenance tasks with a limited scope, get to know the system and grow with it, and have their first customer contacts.

In spite of the points mentioned before, some maintenance projects need better qualified staff than development projects. One reason is the constant customer contact during all phases and not only during conception. This is at least true for the chief designer.

The typical project team at sd&m in Ratingen consists of a project leader, a non-technical (i.e. domain expert) designer and a technical designer, developers, and a quality assurance agent. The interviewed team is relative stable regarding personnel and consists of four to seven persons. The stability is partly due to economic reasons. In (economically) good times new employees move quicker to new development projects and thus there is more rotation of staff.

The systems sd&m maintains can be divided into two categories. The first category comprises the programs which are developed sd&m itself. Those have an extensive documentation and thus require little effort for program understanding and additional documentation. This is the case because development follows the rules set in the software engineering book from Ernst Denert. The second kind of system is developed by the customers. Little or even no documentation exists and during the ‘take over’ of a project extensive analysis is needed to prepare the required documentation.

Considerably large projects are ‘taken over’ bit by bit from the customer. This includes a review of the legacy system and a description of all components. An additional benefit is that often the customer also gains a first insight into the system. The description of components covers functional and, if possible, non-functional aspects. During take over no modifications are made, only the documentation is prepared: what needs to be documented so that at a later stage less analysis is required. This documentation serves as a starting point of maintenance projects so that developers have a basis to work on.

The Maintenance Process

Identification

The standard maintenance process starts with an error report from the customer via several means such as project leader, other contact persons, or sometimes the users call the developers directly. The problems are not reported to a customer’s help desk but to an external service provider, here sd&m. Then the contact person at sd&m, and not the customer, decides whether the requested measure makes sense, i.e. whether it will be implemented or not.

Since sd&m has the required domain knowledge they evaluate the benefits and also the short term costs. If any task takes longer than five days this would mean that an additional budget is needed because it is not covered by the standard maintenance contract. In those cases the approval from the customer in the person of the customer's IT manager is required.

The error report is entered into a system called SiteDB that is a tool for tracking of maintenance measures. The contact person at sd&m decides who from the team has to do what until when, taking into account the constantly changing requests of the customer. It is difficult to make promises regarding deadlines but the customer does not expect it anyway. More important is that critical problems are addressed and fixed as soon as possible.

Critical problems are given the priority A which means that the error prevents production and a fix need to be delivered within one working day. Priority B are problems that hinder production and priority C is for errors that are neither A nor B. Those are rarely implemented for budget reasons. In addition to a priority the errors are given a classification which are error correction, modification request, and take over analysis.

An evaluation also leads to a further differentiation in specification errors, i.e. a problem with requirements, and implementation errors which are the faults of developers, i.e. of sd&m. According to the interviewees specification errors occur more often than implementation errors.

The delivery of all fixed errors does not happen in releases. Every completed measure is delivered immediately or sometimes as package when a developer has finished several tasks at the same time. This is possible because of the software architecture used. Since the take over of the product in summer 2001 more than 1.000 individual measures have been delivered. This means that on certain days 2-3 deliveries per day have happened.

Although the quick implementation is intended a side effect is that some new measures reverse or neutralise the impact of previous measures. The overview of all tasks has been partly lost and not all measures are well thought-out.

Every team member is responsible for certain sub-systems. They receive message if a new task is entered into SiteDB, where they can also see information about existing tasks.

Analysis & Design

During analysis the responsibility for a task is transferred from the contact person to the developer. The exact point in time depends on the knowledge and experience of the developer. Any results of a preceding analysis are handed over as well as sometimes even ideas for a possible solution.

When receiving a task, the developer in charge of the maintenance request often calls the customer to get an example to reproduce the error in the test system and searches for similar error in the documentation in SiteDB. Sometimes it is not possible to reproduce the error due to the different (more limited) test environment. The main reasons are the different performance and less data in the test system.

The different system they use are a test system for the reproduction of error and mainly for development, an integration or parallel system for integration and approval testing, and a productive system which is the operational system with complete master data and transaction data.

When the fault is discovered the developer defines the requirements and identifies the elements to be changed. Depending on the fault the impact of the expected modifications on processes and neighbouring systems are also identified. If required activities belonging to program understanding are carried out in parallel.

Tools for the analysis of legacy systems are at an experimental stage at sd&m. Otherwise there is only little tool support. The reasons are they do not have the critical mass that justifies investing in tools. In different environments or with many more or more heterogeneous customers this is more likely or more appropriate.

Implementation

The implementation of the modification is left to the developer. As analysis, design and implementation are within the responsibility of one person these activities are not always carried out sequentially but together.

Testing

Testing is done manually, either by the developer who performed the changes or another team member. The test procedure follows the quality management plan. If the criticality is not low, the four-eye principal (or dual control) is mandatory. Criticality, which is defined as the possible impact, is important not the complexity of the implementation. The reason is to keep both risk and cost minimal.

Testing through other team members often happens in the form of a code review. Also if the test system is not suited for testing for reasons mentioned above only code review is performed if this is considered to be sufficient. Sometimes testing is done in the operational system. For example the customers is called and asked to try if the removal of the reported error has been successful.

Delivery

The program is then transferred to the integration system where the customer is supposed to test the system. They assume that the customer is doing that most of time. However, the customer has to approve the changes for legal reasons. Acceptance tests are performed on several levels and afterwards the program is finally transferred to the productive system. Sometimes this is done by the developers themselves, although it is supposed to be done by designated specialists for this task.

Maintenance Management

Aspects that are considered continuously are the planning of maintainability and extensibility of the system. This is done for the customer but also for the benefit of sd&m's employees when doing their job. For this purpose a list is gone through to check if certain activities that are concerned with maintenance are possible.

The configuration and version control is done very thoroughly, much better than by the customers themselves. The configuration management tool used is CVS. Tracking of change requests happens through the project leader and the contact person. Additionally regular meetings with customer representatives are held where important issues are discussed in detail.

Additional Comments

Comments about the extended IEEE process are already included above. They did not comment specifically on the IEEE process.

Assessment & Outlook

Assessment

The two interviewees in Ratingen also base their assessment of their work on a customer survey, which is done regularly. According to them the quality is very good or good and the efficiency of their work is also good.

The main factor contributing to this positive result is the replacement of parts of the customer's IT department. The contact for the departments which use the system is directly sd&m (however, they do not serve as a help desk for users). Important in this context is that sd&m has the technical and domain knowledge to provide the required efficient support. Other contributing factors are the configuration management and the methods they apply, both of them better than their customer, and the assessment of existing (legacy) systems as an important basis for their work.

Aspects which may be considered as negative from the customers' perspective are that sd&m does not do any mass data changes and does not operate the system for the customer. sd&m does not have the required knowledge for mass data changes and therefore does not take the responsibility for doing that. However, unofficially they do it but the responsibility remains with the customer. With regard to operations sd&m argues that the first contact for user support always has to be within the customers' company.

Interestingly both interviewees did not name any weaknesses at the end of the interview although they identified some earlier on such as the partly lost overview/control so that some measure neutralise each other or reverse previous changes.

Outlook

Changes in their work which they would desire are to see legacy systems die, to work in modern development environments and acquire new development projects through maintenance projects by becoming indispensable for new developments based on their technical and especially their domain knowledge.

In contrast to their wished they expect, due to increasing cost pressure, that the maintenance budgets will decrease through decisions from top management, not the IT management. One example is that from one year to the next the budget for a project was reduced by the approximately 30%. The consequence is that customers have to wait longer for implementation, which endangers a good project atmosphere.

5.6 Case 6 – Debeka

Interview Partner	Franz Klein, Christian Rünz
Company/Department	Debeka, IT development
Category	Internal IT development dept.
Place of interview	Koblenz
Date of interview	30.10.2003
Duration of interview	120 minutes

Background

Company Information

The Debeka insurance group was founded in 1905. Companies of the group comprise the following: health insurance, life assurance, building society, pension fund, and general insurances. Apart from the health and life insurance business, which have the form of mutual companies, the different branches are public limited companies. Debeka belongs to the top ten of the insurance businesses in Germany and is represented at more than 1,200 locations throughout Germany.

Around 200 people work in the area where the interview partners come from. Both of them work for a department called information technology development responsible for the development of applications for the general insurance business, which is to be differentiated from information technology system that is responsible for hardware.

Importance of Software

Although software and software related services are not provided for external customers the applications developed and used are essential for the business. For example at the turn of the year many contracts have to be adapted. Due to the dimensions of the number of contracts efficient handling is imperative. Any software related service is only assigned to external providers to import special knowledge that does not exist within the Debeka group.

Most applications are Debeka's own developments which have been developed over a period of twenty years. They mainly run on a central mainframe computer.

The users and thus the customers of the developed applications are only internal users; there are no developments for external customers. However, the programs also comprise internet applications but those offer only a limited functionality. The main functions are reserved for professional sales staff for their consulting and sales activities.

Development and Maintenance

In principle there is no separation between development and maintenance. The separation of activities to implement different tasks is based on the situation. The implementation of measures has always a reference to projects. The reasons are that this enables open communication and offers the option to concentrate required personnel in one project team.

Debeka has a process model for software development that contains guidelines for the whole process. Not every new trend in software engineering is followed because this would require too much effort. But the process model is adapted if appropriate.

Maintenance

Definitions

The interviewees say that they do not use the term maintenance explicitly. The realisation of measures, which could be referred to as maintenance, happens with reference to a certain task.

Thus my definition cannot be applied to the situation of Debeka. However, the interviewees said that my definition is adequate, quite true and that they had nothing to add.

The Maintenance Function

A hierarchy in their department exists but this is structured according to general knowledge and not that a certain group has the power. Persons with diverse backgrounds such as computer scientists, business people and insurance salespeople are employed in the IT area. The only prerequisite is to have a certain background knowledge about IT.

A given task is the focal point, guiding principal or driving force for maintenance activities. For example the change of an insurance tariff can lead to the development of new systems and the modification of existing once. All necessary measures are implemented within the fulfilment of the task. New developments occur regularly and depend on the area/division. In insurance areas newer to the company more new developments are required than in older business areas.

From the point of view of the planning process, maintenance is a kind of buffer. This is used for measures that have not been planned in and can be per division e.g. one man year. With cost reference it is interesting to note that the employees have salaried positions and no separate budget for maintenance or further development is set up.

The Maintenance Process

Identification

The interview partners used the adaptation of insurance tariffs as an example to illustrate the process. The impulse for that and similar projects often come from market observation but can also come from many other sides. The setting of the priorities is a decision of the board of directors.

The department information technology development (i.e. the department of the interview partners) first analyses the task and writes a proposition how this can be realised. Top management needs to approve it. As the importance of tasks has already been established by the board, the setting of priorities is only concerned with the time of implementation. The estimation of efforts has be as detailed as possible before it is integrated in the change plan.

Required classification of tasks happens on several levels, which are errors, further developments and the importance of each of those. Errors are usually corrected immediately. In case of critical errors transaction can even be blocked if required.

Analysis & Design

A very extensive stage is the analysis stage. First a list is compiled of what needs to be done. They try to gather all facts which are relevant for the implementation. Besides from business information the only reliable source is the source code. Documentation and code are mostly not in sync; therefore the documentation only supports the analysis by providing information why something has been done.

During the last years the documentation was getting better and more reliable. However, they do not have fixed norms how to do the documentation. This is down to the developers who can e.g. make their comments on the paper form of a change request or add them directly in the source code. More experienced developers will do their documentation throughout the whole process, other at the end of process. As long as the documentation is available at the end it is accepted. Increasingly the documentation is stored in a repository but everything that exists in paper remains in paper from. Changes are linked via numbers to source code and paper forms.

Another part of the analysis stage is to check input masks, databases and forms to find out what needs to be changed. Here the link to the market is very close in order to do what is really needed and expected. Also the coordination of and agreement on priorities with the sales division is crucial, e.g. should something be ready before a certain deadline or is the content more important.

The analysis also includes an estimation of the scope of the task. The question is whether this can be managed the way it is proposed or not. After an evaluation and an agreement about the times the programmers give a guarantee to keep the deadline. So far their estimations were never wrong, i.e. the results were not much different from the estimation. In the given example the lead time plus implementation was approximately half a year. However, a task can still be rejected at this stage if the analysis shows higher cost or efforts than estimated which is considered as too much.

The analysis stage is support by some tools for code analysis and evaluation like CASE tools, a repository which contains all data fields and the programs they belong to, and workbenches to understand/comprehend branches in programs. The latter ones have been in use for five years.

When the deadline and content are fixed, the detailed specifications are established. Again, a close coordination with the divisions is required. The specifications are determined by the developer who is responsible for the implementation. The elements to be changed are identified and sometimes, but not very often, tests are created.

Implementation

Specification and implementation are partly done in parallel. This is down to the developer. Often it is necessary to coordinate the implementation with other areas for capacity planning. One example for capacity problems is the usage of printers at the end of the year to send out all the insurance policies which have been changed.

With regard to implementation every team should in principle be able to do everything. Of course, some are betters at certain tasks than others and therefore do the programming the way they can do it. The programming guidelines are in fact quite detailed but offer some freedom to the developers, e.g. they can decide to enter data in an additional table or directly in the code.

Testing

In some cases initial acceptance assessments with the involvement of users are carried out at this stage. Also the data processing auditing department checks from time to time if certain standards are followed.

In addition to some official audits, testing of the programs by the department itself is important. One third of the tests include white box tests, done by the program developer. The other two third are black box tests. Here the four eye principal or dual control is exercised. One person programs the software, another person checks whether it is okay. Depending on the situation this happens in the form of a code review and/or others tests. The guiding principal is to do it as easy as possible.

The second person, the tester, usually comes from the same team. In more complex cases the application/modification is given to a division where the end users do the testing.

The usual tool support at this stage is a line debugger. They also have a test system which is smaller than the productive system but the environment is identical. Here test cases are run through to make sure the system operates as required. Although the test cases do not cover everything, new cases are not created that often. The majority of cases from real life situations are covered.

Another tool of what they refer to as a test tool is a self made recorder. It compares old and new files and records the difference on microfilm Changes can be reconstructed and understood later on.

With regard to tool support they added that many things are done manually because tools are not available at justifiable conditions. E.g. for the Year-2000 and Euro conversion tasks many tools were tested but they were not reliable. Therefore the changes were done manually.

Acceptance tests are done at an early stage, often simultaneously with design and implementation or mainly system tests. Sometimes drafts of the design and/or prototypes are given to the divisions for testing to receive an early feedback. Also the user training is done quite early. After the delivery of the software it would be too late. The users have to know the system when it is online as they have to work with it. Work processes can change, e.g. new input masks through the substitution of paper through online applications. In case training is required their area is involved but not responsible. Individual training for sales representatives is not provided.

Delivery

Since Debeka has a central system the handover of an application to the productive system is not a complicated task with regard to roll-out. In principal a daily transfer is possible with small modifications occurring constantly. In the day time the operational system is used in dialogue mode, at night for batch processing. The productive setting for dialogue process mode transactions happens at night. The changes are effective after the next boot up in the morning of the next day. Affected users are informed about the time of the change. The only part of setting a system productive that requires more effort is when new data has to be entered. This is the case if e.g. new motor vehicle insurance classes are valid.

Maintenance Management

Planning for extensibility and maintainability of software is done more thoroughly than some years ago. Experience leads to some adaptations but often this is not possible due to a negative cost benefit ratio.

As most applications run on a central mainframe computer different versions of the systems are not used. Sales representatives with “independent” notebooks will try to get the latest version as soon as possible. A field informs the users about the version number so it is easy to recognise if one has the current version or not.

Old versions of the software are not stored only the difference from an old to a new system are saved on microfilm

Additional Comments

Most comments about the extended IEEE process are integrated in the description above. The final comment about the IEEE maintenance process was that it was interesting and comprehensive.

Assessment & Outlook

Assessment

The first comment with regard to an assessment was the statement that their team has little labour turnover. This can be seen as an indicator for a stable team that works together quite well.

They have very few errors in their applications. The reasons are the continuous adaptation and that the software is well known. Also their dinosaurs/legacy systems are maintainable without bigger problems. An important factor for their success is the fact that in contrast to external software developers they develop the applications for their company. According to them they know their company better, are more analytical and are really interested in the success of the company.

The freedom with regard to programming described above is positive for the motivation of the team members. The downside is that the software might not be optimal with regard to software engineering aspects.

Another aspect which has a negative impact on their overall performance is that the modularisation has not always been enforced and thus they have redundancies. They have been trying to improve it for a while and it is better in more recent applications. However, it is difficult to improve older applications afterwards.

Outlook

Changes they would like to see are to have all systems and applications within one system, to further develop existing applications to improve their processes, to match processes to or within the system, and to adapt and implement new aspects quicker. Furthermore they wish for tools for their software or rather to use software where tools exist.

Some bigger changes will come in the near future but they did not specify those changes any further. The adaptation of applications to include new technologies, more user friendly, i.e. graphical user interfaces, and an increasing modularisation are already being implemented or will come soon.

6 Maintenance Framework for RUP

This chapter aims to present where maintenance-related activities need to be added to existing workflows of the Rational Unified Process and to establish an additional workflow for maintenance for RUP.

As identified in chapter three, RUP contains several activities which are relevant in the context of maintenance, however, most of these activities are not maintenance specific. The analysis of the extended IEEE maintenance process demonstrated that other maintenance specific activities are not contained in RUP. This leads to the finding that maintenance activities need to be added to the process model of RUP so that it can be applied for development *and* for maintenance.

The first option is to add maintenance-related activities to existing workflows. The detailed descriptions of the workflows can be extended and the basic structure can remain as it is.

The other alternative is to establish a new workflow. At first view this seems to require more effort and changes than the first option. However, since a workflow summarises activities belonging together, the logical grouping of maintenance-related activities within a single workflow seems to follow the principle of separation of concern and therefore is a better way of adding maintenance to RUP. As maintenance cannot be added completely to one of the existing workflows, a new and dedicated workflow for maintenance needs to be established. Another big advantage is that an additional workflow raises the awareness of the importance of maintenance and makes maintenance more explicit.

This workflow will be called "Maintenance Workflow". Maintenance may not be the best term to use but other terms such as evolution, modification or reengineering are also related to certain areas of software engineering and not a better choice. Therefore it will be labelled "Maintenance", related to the definition of maintenance established in chapter two.

The following section points out where maintenance-related activities need to be added to existing workflows. This is followed by a description of the maintenance workflow. The last section in this chapter highlights some additional aspects which need to be observed in the context of maintenance in RUP.

Adaptation of existing Workflows

Project Management

The project management workflow can nearly be applied for maintenance projects as it is. The project manager only needs to take into account to keep some experienced members of the development team for later evolution, i.e. pure maintenance cycles. Besides this staffing aspect no other modifications are required.

Business Modelling

The definition of new business processes will often lead to changes in the organisation's software systems. From this it follows that a strong link to business process engineering is required to anticipate future modifications. The roles involved in business modelling have to use their contacts to receive relevant information as early as possible. They also have to assess the impact of changing business process on the system. This activity remains within this workflow and is not moved to the maintenance workflow because the roles involved in this workflow, i.e. the business-process analyst and the business designer, have the required domain knowledge to provide adequate assessments.

Of course, existing business vision documents and other business models need to be maintained, i.e. kept up to date, as all other artefacts.

Requirements

The revision of requirements needs to be actively followed up. It is important that this is done regularly and thoroughly to fully understand the needs of the users and anticipate future modification requests. The activities "analyse the problem" and "understand stakeholder needs" need to be extended to include an understanding of the requirements contained in modification requests.

The management of changing requirements should be linked to the understanding of stakeholder needs because the changed requirements are based on users' needs and are a trigger for maintenance.

Analysis and Design

So far only the refinement of the architecture is covered. Activities for the redesign of other components and the update of the design documentation need to be added to the workflow.

The results of reverse engineering activities such as program understanding and impact analysis are required for this workflow. However, the activities themselves will be embedded in the maintenance workflow because the analysis and design workflow only contains forward engineering activities so far and based on the principle of separation of concerns these other activities are grouped together within a different workflow.

Implementation

The forward engineering activities of the implementation of modification requests are nearly identical with those of development projects. Based on the design specified before, the code needs to be programmed and tested. The maintenance-related activity of code understanding, that is required before the implementation can start, is embedded in the maintenance workflow.

Test

The test workflow needs to be restructured and some activities need to be extended. The analysis of the IEEE process and processes in industry demonstrated that tests including acceptance tests need to start right from the beginning. Thus also design acceptance tests should be contained in the workflow.

Additionally tests need to be maintained and kept up-to-date. Therefore the planning and designing of tests should be extended to contain test maintenance as well. Otherwise the workflow covers all activities that are required for maintenance projects.

Deployment

The deployment workflow turns the finished or modified product over to its users. So far it does not contain any activity to inform the users about finished products or implemented modifications. The activity “notify users/customers” needs to be added. This notification also includes informing users about errors that have been identified and which are going to be removed with the next update.

In the current model of RUP, deployment starts in the middle of the elaboration phase or even later. In certain cases such as critical errors it can be required to inform users about the discovered fault directly after it has been discovered. Thus the start of the workflow, i.e. the activities “plan deployment” and “notify user” have potentially to start in the inception phase. Furthermore two different ways to distribute updates and new releases should be covered because the deployment of updates to remove critical errors will be different from normal releases.

Configuration and Change Management

The evaluation whether a modification or change request is valid or not will be moved to the maintenance workflow because the roles there have the required knowledge to make this decision. The configuration and change management workflow is more administrative and does not get involved with analytical activities like the evaluation of the validity of modification requests.

Otherwise this workflow supports maintenance projects through all activities contained.

Environment

The environment workflow provides the foundation for a supportive environment. This has to include processes and tools for maintenance. Especially for different categories of maintenance different processes are required because corrective maintenance often requires a much quicker process than e.g. adaptive maintenance.

Furthermore, the environment workflow should also cover the evaluation of projects to learn for future projects and adapt the processes accordingly.

The Maintenance Workflow

Purpose

The purpose of this workflow is the planning and execution of activities for the modification of a software product to improve performance or other attributes, to adapt the product to a modified/changed environment, to correct faults or to improve its maintainability during all stages of the software life cycle.

The workflow aims to promote, ensure and preserve maintainability to decrease the maintenance efforts after delivery. It provides methods and tools needed for maintenance and clearly assigns the responsibility for maintenance-related aspects to certain roles.

Input

The input depends very much on the cycle, i.e. whether it is a development cycle or an early or late evolution cycle. For development cycles just the needs of the organisation and existing guidelines serve as an input for the process.

For evolution cycles all artefacts of the system being modified including their history if available are required. Additionally information about the context of the system, (business) ideas and data about change prone modules and previous defects and modifications are needed.

To schedule the implementation of modification requests, available resources and priorities (of the customer) have to be available as well.

<u>Process</u>

Maintenance planning ideally begins during the planning stage of software. The expected need for extensibility and adaptability of the software product has to be taken into account and planned for accordingly. Guidelines for forward engineering activities which are especially related with improving maintainability have to be created and later on promoted and enforced. Tests and audits will ensure that these guidelines are observed. These tests are not part of the test workflow because it is more an internal control than a test of features which are required by the customer.

The maintenance workflow supports the business modelling and requirements workflow in their active search for areas which need modification or might need maintenance in the near future. Data is received from configuration and change management to identify and evaluate areas which have been modified more often than others. This can be used to determine likely future maintenance efforts and plan the resources accordingly.

The activities determined so far are general activities which need to be applied throughout the life cycle of a software product. A concrete maintenance process is covered and supported by the maintenance workflow as follows:

Modification requests are received and entered into a database. Future tracking is done through the configuration and change management workflow; nevertheless the roles of the maintenance workflow are responsible for continuous updating of the entered information.

The incoming modification requests are evaluated whether they are valid or not. The classification triggers different maintenance processes with scheduling and prioritising influencing the point in time of implementation. A first analysis is also performed that provides a rough estimate about costs, benefits and the impact of the modification before the modification request is handed over to the roles of the requirements workflow who continue processing it.

At this stage or a little bit later in the process often the customer needs to get involved to make a decision whether to approve the change, i.e. to proceed with the modification or to abort the process. This decision making process is supported by the contact person that belongs to the maintenance workflow. Alternatively the project manager can perform this task.

If required reverse engineering activities for analysis are carried out within the maintenance workflow. Elements which are affected and the impact of the changes are identified. Together with program understanding, this provides essential information which is especially relevant for roles in the design and implementation workflows.

An additional task that is performed within the maintenance workflow is to create a maintenance package as suggested by [Kajko-Mattsson 2001]. This is done in close cooperation with deployment and is partly intended for external customers if some or even all maintenance tasks are performed by the customers themselves. Of course, the information contained in the maintenance package is also relevant for internal modifications or for future projects.

Output

The artefacts created by the maintenance workflow are firstly guidelines for development and maintenance projects which ensure and enforce a high degree of maintainability.

Furthermore updated documentation and mainly supporting information for other workflows such as the impact of changes are created within this workflow.

Roles

Every role in a project is responsible for high quality software and thus for software that is easy to maintain. This aspect of the responsibility needs to be understood by everybody involved. Of course, some roles have a greater responsibility for maintenance-related aspects and these roles are as follows:

The maintenance coordinator oversees all activities in the context of maintenance. This role is responsible for the promotion of maintainability and the communication flow between the roles in different workflows which need information from the respective counterpart.

Furthermore the maintenance coordinator together with the project manager, a customer representative, and potentially a planner sets priorities and schedules the modification requests for implementation.

The receiver is someone who is always available to receive all kinds of modification requests and can provide information about the status of requests being implemented.

The validator and classifier has the required domain and system knowledge to assess and classify incoming modification requests.

The reverse engineer plays a crucial role by providing information that is essential for roles in other workflows such as designers and implementers. Besides performing a thorough analysis of existing system, this role also supports others who need to get an insight into old code or documentation.

Of course, every stakeholder can submit modification requests and thus plays a role within the maintenance workflow.

Tools

To receive modification requests all kinds of communication tools such as emails, fax, and web interfaces need to be available to the executing roles as well as to other stakeholders who want to submit a request.

Databases with several kinds of information are required. The first one needs to contain data about modification requests with all relevant information about those, data about earlier modification requests and error reports as well as classification information. This can support the validation and classification process and provides a basis for tracking of modification requests throughout the process. Another database can support the analysis by providing data about affected areas and links between components.

A scheduling tool that can manage resources, estimates and priorities can support the overall planning process.

Analysis tools are required to trace features to the code and the documentation and to support the program understanding process by analysis static and dynamic aspects of the system.

Finally, modelling tools help to develop and modify all relevant views such as use case models and architectural views which are required to provide relevant information for other workflows.

Additional Aspects

Milestones

Some of the milestones need to be extended to take the added activities into account. The Life Cycle Objective milestone can nearly remain the same; just the focus of some of the elements needs to be broadened to include maintenance-related aspects such as the importance of maintainability in elements like the life cycle plan. The principle need for extensibility and adaptability should also be kept in mind, i.e. included in some documents.

The Life Cycle Architecture milestone certainly has to contain the expected need for extensibility and adaptability in the definition of a system and software architecture. In evolution or maintenance cycles, the system and modification requirements need to be understood, program understanding needs to be completed and an impact analysis should have been performed at the end of the elaboration phase, i.e. at the Life Cycle Architecture milestone.

Engineering vs. Supporting Workflow

Although it is not an essential aspect, for the principle of completeness it needs to be determined if the maintenance workflow belongs to the engineering or the supporting workflows.

An argument for an engineering workflow is that it contains engineering activities where the product itself is modified, i.e. engineered. It would be possible to divide the engineering workflows into forward and reverse engineering workflows. Some parts of the maintenance workflow would belong to a dedicated reverse engineering workflow containing activities from the field of reverse engineering such as program understanding and design recovery. The forward moving aspects of the maintenance workflow such as to increase maintainability during development would be part of a forward engineering “maintainability workflow”. On the other hand, if the maintenance workflow is split up into two workflows, the advantage of coordinated maintenance management is partly lost.

Furthermore, since the maintenance workflow mainly supports the engineering workflows by providing input that is required for the engineering activities to produce a software product, it is rather a supporting workflow than an engineering workflow.

7 Conclusion – Comparison of Theory & Practice –

First insights into the topic and hence the basis for a conclusion could already be gained during the discussion of terms and definitions in chapter two. The description and analysis of software development process models in chapter three and the evaluation of the extended IEEE maintenance process in chapter four broadened the theoretical background on which the conclusion is based. The final step to establish the following conclusion was a comparison of the different theories outlined within the chapters three and four and a further comparison of those theoretical approaches and the practices identified during the interviews as described in chapter five.

Terminology

The terminology in the field of software maintenance is not precisely defined.

The boundaries between new development and maintenance are blurred. When looking at literature it is not clear where the first one stops and the other starts. Even some categories of maintenance which are often assumed to be precisely defined such as corrective maintenance have different meanings by different authors.

The discussion whether reengineering or maintenance is the more abstract or generic term continues throughout the literature: does the field of reengineering include maintenance, does maintenance include reengineering, are the areas identical, totally disjoint or do they intersect.

The terms are also not clear in industry and therefore the software development and maintenance processes are not absolutely transparent. At the beginning of a number of interviews some interview partners asked what area I wanted to talk about because maintenance activities could be everything. Some interview partners even contradicted themselves during the interviews regarding their definitions.

Thus, one cannot assume that the terminology in the field of software maintenance is understood in the same way by everyone. It needs to be discussed and defined when talking or writing about this topic in the academic world as well as in industry.

Statistical Data

Current figures about the importance of maintenance are not available.

Surveys from Lientz and Swanson conducted in the 1970s and early 1980s are still quoted in more recent books as for example [Balzert 2001] and [Sommerville 2001]. Current figures are not available. [Lehner 1999] writes that it is often assumed how much of the software budget is used for maintenance. And it still needs to be proven that the percentage of maintenance of the budget increases as suggested by other authors.

Admittedly the people interviewed are not responsible for the software budget. However, the statements they made demonstrate or indicate that at most companies, there is no separation between development and maintenance in their budgets. Most companies can only assume how much is spent on maintenance if they are interested in this at all. This leads to the next aspect.

Thus, the importance of maintenance nowadays is known neither in industry nor in literature. Thorough and extensive empirical research is required to provide up-to-date figures to determine the current importance of maintenance.

Awareness of Maintenance

Maintenance is not sufficiently observed or regarded as important in industry.

It is surprising that many companies do not have a dedicated position (or cost centre) for maintenance within otherwise detailed budgets. Furthermore, only one of the companies interviewed has a clear definition of what maintenance is but the interviewee did not know the exact definition. Considering these facts, it is not surprising that many companies do not have a dedicated process for maintenance. At the first branch of one company they presented an abstract from the company's handbook showing a maintenance process description, while at another branch they said that maintenance was not included in that handbook. Statements like "we do not do any maintenance" round off the picture.

In the literature, maintenance is said to be so important that it is not suitable as a training ground. Some companies go to the other extreme by explicitly using maintenance tasks as a training ground for new staff to become acquainted with the software. Depending on the tasks this may be useful, but it is the most obvious difference between theory and practice.

Another issue is whether maintenance is considered as an unpopular area to work in. Here academics and practitioners agree, partly saying that long maintenance projects are a kind of punishment, but not much is being done to improve this image.

Thus, it is necessary to promote the importance of maintenance in industry and within organisations. Companies as well as their employees need to be aware of the significance of maintenance and have to understand and believe that working in the field of maintenance can be as challenging as working in development.

Embedding of Maintenance in Software Process Models

Software process models do not sufficiently represent the whole software life cycle.

The life cycle of a software product starts with its initial development, followed by an evolution phase where the software is further developed. This evolution stops at a certain point in time where the software is not further developed. Usually maintenance will continue for some more months or even years until the product is no longer used. During this last phase of the life cycle software is only being maintained to keep it operational without any further enhancements being implemented.

Maintenance is definitely part of the software life cycle. Taking the view that maintenance will also take place during the evolution phase or even during development, it is usually the part with the longest duration. Although maintenance is obviously important, it only plays a minor role in software process models.

Maintenance is only explicitly included in the Waterfall model, where it is the last stage that comes at the very end together with operations. The V-model and, depending on the view, the spiral model do not include maintenance at all. Most models include maintenance implicitly. In Extreme Programming it is said that maintenance is the normal state of an XP project, but the suitability of XP for maintenance is limited. In the spiral model maintenance can be included as one ore more separate iterations, similar to the evolutionary model. The Rational Unified Process includes maintenance throughout several maintenance-related activities that are contained in existing workflows.

Although all interviewed companies have dedicated process models for software development, only IBM has a clearly defined and detailed process for maintenance as well.

All these examples demonstrate that software process models focus on the initial development of software and neglect the longest part of the life cycle: the maintenance phase.

Tool support for Maintenance

There is a gap between the tool support suggested in literature and the actual tool usage in practice.

According to software engineering literature, general software tool support is very important and can be the crucial factor for success in maintenance projects. Many tasks such as refactoring can be partly automated using adequate tools. Good tool support is also mentioned as one reason why reengineering is getting better and more efficient. One definition of reengineering even included automation through tools as one aspect.

In contrast to that view are the results from the interviews. Only one company, IBM, named tool support as an important factor influencing the quality and efficiency of its process. This company provides services for many customers. The use of tools is efficient because IBM has the critical mass to justify the investments. The others said that most of them tried tools at some stage but decided not to use them. The reasons are that too much effort is required to adapt (to) the tools, the investments are simply to high, and that no tools for their software exist. Only for simple tasks are tools such as error report databases and line debuggers used.

Organisational Context

The organisational context and the nature of the maintenance task are important to determine the appropriate process.

Most authors say that the industry as the external environment is less relevant than the companies' objectives and policies regarding software development and maintenance.

During all interviews the interviewees mentioned that the context of their company determines their software development and maintenance process. At Debeka they said that they develop software for their own company and thus are highly motivated to deliver quality software. In contrast, although IS sales belong to the VW group, they are set up as a profit centre and have different objectives than the purely internal developers at Debeka. This is also reflected in their process where budget constraints play a more important role than at Debeka. sd&m partly adapts to their customers' processes showing the close and individual cooperation between the software company and the customer.

One aspect mentioned before which influences the maintenance process is the existing tool support. IBM as the biggest company uses tools all along the way and many steps are automated. The other companies use less tools and therefore much more manual work is required. The sizes of the companies and their possibility to use tools efficiently have been identified as the aspects which mainly determine the usage of tools. Also partly determined by the size is the issue whether development and maintenance are separate or not. IBM and Volkswagen clearly separate development and maintenance. sd&m distinguishes between maintenance and development projects but mixes the staff and the process. At Debeka and the engineering company there is no separation at all.

The engineering company is also an example for an embedded system. The close cooperation between software and hardware developers was not found at the other companies in that form. Moreover due to the small team size, they have hardly any documentation.

A statement at an interview at sd&m was that the highest impact on the maintenance process has the system itself which is to be maintained. Additionally the maintenance categories lead to different processes. This is mainly corrective vs. other maintenance tasks. Error correction is usually done immediately, applying a short process. Other tasks follow more sophisticated processes which are partly similar to the IEEE maintenance process. Nevertheless IBM is the only company which has distinct processes for more than two maintenance categories.

Applicability of IEEE Maintenance Process

The applicability of the IEEE maintenance process is limited.

All interviewees were asked what they think of the extended IEEE process. The most common answer was that it is comprehensive and somehow appropriate but too complex. The main criticism was that it is only one big process. Although everything seems to be included it is difficult to see what is really needed for certain tasks because conditions are missing when to use what. It is too complicated to be of use for a simple error correction process.

Another issue that came up during the interviews which is not covered by the IEEE process is the question of control. The IEEE process assumes that the provider of the maintenance service is also in complete control of the process and can make any decision needed. The customer focus is missing and thus certain activities asking for decisions by the customers are also missing.

An obstacle that was identified during the interviews and the theoretical analysis of this thesis is the problem of granularity. Some fine-grained activities need to be summarised to methods and techniques to be useful. With the current IEEE process it is not possible and thus the interviewees did not recognise techniques like refactoring. Although these are somehow included in the process they cannot be identified because the description of activities happens on a very detailed level.

Thus, the IEEE process needs to be adapted to fit to specific situations. On the one hand, it needs to be broken down to reduce the complexity and make it fit to different categories such as corrective or adaptive maintenance. On the other hand, certain activities need to be summarised to well-known methods such as refactoring so that the IEEE maintenance process can be easily applied for e.g. dedicated refactoring projects as well.

8 Recommendations for further Research

Although chapter six presented specific recommendations about an additional maintenance workflow for the Rational Unified Process, this thesis leaves some aspects for further research.

An obvious step is a validation as well as a possible extension and improvement of the suggested workflow. The ideal way for a validation is to find a company that uses RUP and that is willing to integrate the maintenance workflow in their processes. The application of the workflow would show whether it is a valid way to do maintenance. This should identify areas within the workflow which can be improved. However, it is not realistic to find a company that would do that. A valid alternative is to ask the interviewed and other companies for comments about the workflow and its applicability in practice.

There are also some other areas which can be investigated in more detail. Many books, even from recent years, often quote the surveys of Lientz and Swanson from the 1980s instead of recent research results. Current figures which can demonstrate the importance of maintenance nowadays are hardly available. Therefore primary research in the form of empirical work to fill that gap is an interesting field to be looked at in more detail.

The interviews in this thesis only covered the personal views about maintenance of one to two employees of six companies. More interviews asking for more information about certain parts of the maintenance process, such as analysis-related tasks, can lead to a different emphasis of suggested processes.

Even a survey of companies and an exhaustive literature review about the terms “maintenance” and “reengineering” themselves can be the basis for extensive discussions.

An alternative to a specialisation on individual activities is to look at whole maintenance projects in more detail. Especially a differentiation of processes according to categories of maintenance will broaden the view on a life cycle model for software. A desired result would be a reference process model that covers the whole software life cycle from inception to close down of the system.

Most of the suggestions made above can be executed as stand-alone projects or can be combined to a bigger research project in the form of a doctoral thesis. In the latter case it is important to specify the objects of the research because “maintenance” is a broad field and can be looked at from many perspectives, such as management, technical perspectives or process oriented views.

9 References

Agresti, W. W. (1986). "The Convential Software Life-cycle Model: Its Evolution and Assumptions." New Paradigms for Software Development: 2-5.

Antoniol, G., G. Canfora, G. Casazza and A. D. Lucia (2000). Identifying the Starting Impact Set of a Maintenance Request: A Case Study. 4th European Conference on Software Maintenance and Reengineering, Zürich, IEEE Computer Society: 227-230

Arnold, R. S. (1993). A Road Map Guide to Software Reengineering Technology. In R. S. Arnold Software Reengineering. Los Alamitos, CA, IEEE Computer Society Press: 3-22.

Aversano, L., G. Canfora and S. Stefanucci (2001). Understanding and Improving the Maintenance Process: A Method and Two Case Studies. 9th International Workshop on Program Comprehension, Toronto, Canada, IEEE Computer Society: 199-208

Balzert, H. (2001). Lehrbuch der Software-Technik, Band 1: Software-Entwicklung. 2nd ed. Heidelberg, Spektrum.

Beck, K. (2000). Extreme programming explained: embrace change Upper Saddle River, Addison-Wesley.

Bennett, K., B. Cornelius, M. Munro and D. Robson (1991). Software maintenance. In J. A. McDermid Software Engineer's Reference Book. Oxford, U.K., Butterworth-Heinemann: 20.0-20.18.

Bennett, K. and V. Rajlich (2000). Software Maintenance and Evolution: a Roadmap. 22nd International Conference on Software Engineering, Limerick, Ireland, ACM: 73-87

Bianchi, A., A. R. Fasolino and G. Visaggio (2000). An Exploratory Case Study of the Maintenance Effectiveness of Traceability Models. 8th International Workshop on Program Comprehension, Limerick, Ireland, IEEE Computer Society: 149-158

BMVg (1997). Entwicklungsstandard für IT-Systeme des Bundes (EStdIT)

Boehm, B. (1996). "Anchoring the Software Process." IEEE Software **13**(7): 73-82.

Bratthall, L. and C. Wohlin (2000). Understanding Some Software Quality Aspects from Architecture and Design Models. 8th International Workshop on Program Comprehension, Limerick, Ireland, IEEE Computer Society: 27-34

Byrne, E. J. (1992). A Conceptual Foundation for Software Re-engineering. Conference on Software Maintenance, Orlando, Florida, IEEE Computer Society: 226-235

Chikofsky, E. J. and J. H. Cross (1990). "Reverse Engineering and Design Recovery: A Taxonomy." IEEE Software **7**(1): 13-17.

Church, K. and G. t. Braake (2002). The Future of Software Development. In S. Valenti Successful Software Reengineering. London, IRM Press: 99-110.

ClearStream (2001). eXtreme Programming.
http://www.clrstream.com/Extreme%20Programming.htm

Corbi, T. A. (1989). Program understanding: Challenge for the 1990s. In R. S. Arnold Software Reengineering. Los Alamitos, California, IEEE Computer Society Press: 596-608.

Denert, E. (1991). Software-Engineering Berlin, Springer.

Ebert, J. (2002). Softwaretechnik II. Vorlesung SS02. Koblenz.

Freedman, A. (1995). The computer glossary: the complete illustrated desk reference. 7th ed. New York, The Computer Language Company.

IEEE (1998). IEEE Standard for Software Maintenance. IEEE Std 1219-1998. New York, The Institute of Electrical and Electronics Engineers.

Illingworth, V., (Ed.) (1983). Dictionary of Computing. Oxford, Oxford University Press.

Kajko-Mattsson, M. (2001). Towards a Business Maintenance Model. IEEE International Conference on Software Maintenance, Florence, IEEE Computer Society: 500-509

Kajko-Mattsson, M., S. Forssander and U. Olsson (2001). Corrective Maintenance Maturity Model: Maintainer's Education and Training. 23rd International Conference on Software Engineering, Toronto, IEEE Computer Society: 610-619

Kruchten, P. (2000). The Rational Unified Process - An Introduction. 2nd ed. Upper Saddle river, Addison-Wesley.

Kusters, R. J. and F. J. Heemstra (2001). Software maintenance: an approach towards control. IEEE International Conference on Software Maintenance, Florence, IEEE Computer Society: 667-670

Lehner, F. (1999). Software Reengineering vs. Business Reengineering. In Workshop Software Reengineering. Bad Honnef, Universität Koblenz.

Madhav, N. and S. Sankar (1990). Application of Formal Specification to Software Maintenance. IEEE International Conference on Software Maintenance, San Diego, California, IEEE Computer Society: 230-241

Martin, J. and C. McClure (1983). Software Maintenance: The Problem and Its Solutions Englewood Cliffs, New Jersey, Prentice Hall.

McClure, C. (1992). The three Rs of software automation: re-engineering, repository, reuseability New Jersey, Prentice-Hall.

Pfleeger, S. L. (1998). Software Engineering: theory and practice Upper Saddle River, Prentice Hall.

Phillips, D. (2000). The Software Project Manager's Handbook - Principles that work at Work. 1st ed. Los Alamitos, California, IEEE Computer Society.

Poole, C. and J. W. Huisman (2001). "Using Extreme Programming in a Maintenance Environment." IEEE Software **18**(6): 42-50.

Pressman, R. S. (2001). Software engineering: a practitioner's approach. 5th ed. New York, McGraw-Hill.

Ramage, M. and K. Bennett (1998). Maintaining Maintainability. IEEE International Conference on Software Maintenance, Bethesda, Maryland, IEEE Computer Society: 275-281

Sneed, H. M. (2001). Impact Analysis of Maintenance Tasks for a Distributed Object-oriented System. International Conference on Software Maintenance, Florence, Italy, IEEE Computer Society: 180-189

Sneed, H. M. (2003). Aufwandsschätzung von Software-Reengineering-Projekten. Regensburg, Institut für Wirtschaftsinformatik, Universität Regensburg.

Sommerville, I. (1992). Software Engineering. 4th ed. New York, Addison-Wesley.

Sommerville, I. (2001). Software engineering. 6th ed. Harlow, Pearson Education Ltd.

Sousa, M. J. C. and H. M. Moreira (1998). A Survey on the Software Maintenance Process. IEEE International Conference on Software Maintenance, Bethesda, Maryland, IEEE Computer Society: 265-274

Spinu, M. (2001). About the conference. http://www.dsi.unifi.it/icsm2001/icsm_about.html

Swatman, P. (2002). Research Methods in Information Systems. Developing and managing a research proposal. Koblenz.

Turk, D. and V. Vaishnavi (2002). Process Models are Software Too: A Domain Class Model for Software Process Models. In S. Valenti Successful Software Reengineering. London, IRM Press: 284-292.

Wells, D. (1999). The XP Philosophy. http://www.extremeprogramming.org/Kent.html

Wells, D. (2003). Extreme Programming: A gentle introduction. http://www.extremeprogramming.org/

Wideman, M. (2003). Software Development and Linearity, Part 1. http://www.maxwideman.com/papers/linearity/spiral.htm

Appendices

Appendix I: Letter – Request for Company Interviews

Ich studiere Informationsmanagement im Masterstudiengang an der Universität in Koblenz und schreibe gerade an meiner Abschlussarbeit zum Thema Softwarewartung. Betreut wird meine Arbeit von Herrn Prof. Troitzsch und Herrn Dr. Winter.
In meiner Abschlussarbeit beschäftige ich mich mit Aktivitäten im Bereich der Softwarewartung. Dieser Bereich hat durch Ereignisse wie z.B. die Jahrzweitausend-Problematik, die Euro-Umstellung und in diesem Jahr die Umstellung der Wertpapierkennnummern (WKN) auf International Security Identification Numbers (ISIN) an Beachtung gewonnen. Die Anpassung von Software als Reaktion auf solche Ereignisse, wie auch die Behebung von Fehlern und die Weiterentwicklung von Softwareware gehören zum Gebiet der Softwarewartung dazu.
In meiner Arbeit möchte ich dabei die Vorgehensweise bei der Softwarewartung der Praxis berücksichtigen, um zu sehen, ob und wie Ansätze der Literatur von Firmen angewendet und bewertet werden. Interessant sind vor allem Begründungen für die Anwendung und insbesondere für die Nichtanwendung verschiedener Wartungsaktivitäten in der Praxis.
Das abschließende Ergebnis der Arbeit soll eine Einbettung der Wartungsaktivitäten in ein Softwareentwicklungsprozessmodell sein.

Für die Identifizierung der Wartungsaktivitäten der Praxis benötige ich Ihre Mitarbeit. Ich möchte drei bis vier Firmen befragen, wie sie einen bestimmten Prozess für die Wartung einsetzen. Im Falle der Umsetzung eines Prozesses wären die genauen Aktivitäten (z.B. Programmverstehen, Aufwandsschätzung, etc.) zu benennen.
Die Firmen, die ich befragen möchte, sollen dabei aus verschiedenen Bereichen kommen. Ich bin also sowohl an großen Konzernen und spezialisierte Softwarefirmen als auch an Firmen interessiert, für die die Softwareentwicklung zwar wichtig ist, aber nicht das Hauptprodukt darstellt. *(individuelle Begründung für jede Firma)*

Meine Bitte:
Ich möchte mich gerne mit Ihnen und/oder einem Ihrer Mitarbeiter Anfang/Mitte Oktober für ein bis zwei Stunden über Softwarewartung unterhalten. Die befragten Personen sollen in Wartungs- bzw. Weiterentwicklungsprojekten arbeiten und einen Überblick über die Aktivitäten haben. Wenn möglich möchte ich zwei Personen gleichzeitig zu diesem Thema befragen.

Den Ablauf der Interviews stelle ich mir folgendermaßen vor. Ich möchte zunächst einleitend meine Arbeit vorstellen und anschließend etwas über den Softwareentwicklungsprozess der Firma erfahren. Der Schwerpunkt der Interviews liegt dann auf der Identifizierung der Wartungsaktivitäten. Das soll anhand eines von Ihrer Abteilung durchgeführten konkreten Projektes geschehen, das Ihnen als geeignet erscheint. Weiterhin erhoffe ich mir eine Beurteilung/Bewertung der Wartungsfunktion sowie der einzelnen Aktivitäten. Ein Ausblick auf erwünschte und erwartete Änderungen in dem Bereich der Softwareentwicklung und Softwarewartung soll am Ende der Gespräche stehen.

Falls Sie noch Fragen zu meiner Arbeit haben, können Sie mich per Email (kuhlurs@uni-koblenz.de) oder telefonisch (0172 - 51 67 67 9) erreichen.

Über eine baldige Antwort und einen ungefähren, gerne auch kurzfristigen Terminvorschlag Ihrerseits würde ich mich sehr freuen.
Mit freundlichen Grüßen
Urs Kuhlmann

Appendix II: Interview Questions Check List

Focus: WHY those maintenance activities – rational for practitioners to select these

required: Reference maintenance process against which the investigation can be conducted

0. Introduction

– Objectives of master thesis
– Purpose and structure of interview; embedding of interviews into thesis

1. Background and structure of organisation

General description

– Company
    international
    size
    age, ownership
– Business areas of company
    products
    customers
– Organisational chart (at least of business unit); chart or description

Importance of IT/software for organisation

– Software as core product or complementary product
– Importance of software for operations
– % of software spending of turnover

„warm up“
have information ready, let interviewee confirm them and add something;
used to classify company, can be helpful to explain certain company specific activities

Organisation of IT function → projects

– Overall size (personnel) and products/services
– Customers: internal & external
– Software development in-house, external or both
– Software development process → identify maintenance; link to 2.

2. Maintenance activities (descriptive)

Structure of maintenance function (or the responsible within IT function)

– what belongs to maintenance – their definition:

– Organisation of maintenance:
    separate from development, embedded in development process?
    personnel – capabilities, number
    structure/communication channels of maintenance staff
– Efforts of maintenance: time, budget (in comparison to new development, may be other IT functions)

Identification of maintenance activities with concrete project(s)

- Maintenance process model
  → if yes: how and why this model was developed
- let them describe concrete project
- Maintenance activities in this project

Review activities in detail

- what is contained in these activities
- WHY are those activities applied
- Responsibilities
- Tools to support activities
- Costs & efforts of main activities if known

Conformance to extended IEEE maintenance process model

- my definition of maintenance – link to IEEE
- compare with extended IEEE model: statement about differences (why (not) certain activities); order according to IEEE model

3. Assessment of maintenance activities

Assessment of maintenance function

- Quality
- Efficiency
- Reasons for good/bad mark: e.g. (un)limited budget, (un)clear process

Assessment of single activities

- Assessment of main activities identified above
- Assessment of activities with currently highest impact (positive/negative) on overall performance

4. Outlook

Desired & expected changes

- Desired changes (what do you want): tools, personnel, management, structure, budget
- Expected changes (what do you expect): …
- Reasons for gap

5. Comments

- Confidentiality
- Other comments or questions for me
- Further procedure, use of answers,
  master thesis, paper

Extended IEEE Process model

- * = added to original IEEE process
- (…) = original IEEE process, not used in interview process model
-

1. Problem/modification identification
Identify problem/modification*
Assign an identification number
Classify type of maintenance: corrective, adaptive, perfective, emergency
Analyse the modification to determine whether to accept, reject, or further evaluate
Make preliminary estimate of the modification size/magnitude
Prioritise the modification
Assign a modification request to a block of modifications scheduled for implementation

2. Analysis
depending on existing artefacts – only source code available → reverse engineering

1. Feasibility analysis
Impact of the modification
Alternate solutions, including prototyping
Analysis of conversion requirements
Safety and security implications
Human factors
Short-term and long-term costs
Value of the benefit of making the modification

2. Detailed analysis
Define firm requirements for the modification
Identify the elements of modification
Identify safety and security issues
Devise a test strategy
Develop an implementation plan

Programme understanding*
Impact analysis*
Cost estimate*
Prioritisation/planning*

3. Design
Identify affected software modules
Modify software module documentation
Create test cases for the new design, including safety and security issues
Identify/create regression tests
Identify documentation update requirements
Update modification list

4. Implementation
Structure implementation*
Plan integration*
Coding
Unit testing
Integration

(Risk analysis)
Test-readiness review

5. System test
System functional test
Interface testing
Regression testing
Test-readiness review to assess preparedness for acceptance testing
… should be conducted by independent test function, witnessed by user but conducted by IT people, 2-4 people

6. Acceptance test
User test at different levels:
(Perform acceptance tests at the functional level)
(Perform interoperability testing)
(Perform regression testing)

7. Delivery
Installation at customer site
(Conduct a physical configuration audit)
Final testing*
Notification of users/customers
(Develop an archival version of the system for backup)
Deployment*
User Training

8. Maintenance Planning/Management*
- Planning maintainability & extensibility during development/maintenance
- Configuration management
- other control/management aspects: e.g. trace CRs
-

Appendix III: Additional information from interviews

IBM

http://www-5.ibm.com/services/de/ams/

Application Management Services – provided services:

- AMS Advisory Services: professional application development processes, strategies for and planning of application development and management;
- Portfolio AMS: accelerated application development services, AMS for outsourced portfolios, IT governance consulting, and life cycle management of application systems;
- Package AMS and Custom AMS: structured analysis and conception of IT solutions, process mapping and transition planning, implementation of solution, application management services throughout the life cycle of the application portfolio;
- AMS Resource Services: coordination of resource-pools, personnel management either of IBM employees or third party personnel;
- Functional Solutions: business process outsourcing for the areas purchasing, document management, payroll, functional solutions for business process for e-procurement, web content management, e-marketplaces, e-business consulting, e-commerce, CRM, SCM, SAP application management.

Short description of one example maintenance project:

Introduction of the EURO currency

The underlying cause for this project was the decision by several countries to introduce a new currency, the Euro. New business requirements were derived from that decision. It also had an impact on non-member states of the Euro zone because also companies outside the Euro zone must be able to deal with the new currency.

The Euro conversion project at IBM started in 1996 with a decision from the board of IBM Europe. The project followed a top down approach with decisions made and requirements set by the top management. The project lasted until the first half of 2002. This was necessary to support and control the successful transition in all areas. The maximum number of employees working on 200 partly parallel projects within the Euro project was approximately 100.

As mentioned in section 5.1, several specified aspects needed to be specified. One example is the question until what date the book keeping had to be switched to Euro. They also did not know for very long the duration of the transition period. Furthermore the transition rates between the member-currencies were fixed rather late in the process.

AMS uses standard IBM methods for their work. However, special methods were developed for the Euro conversion and the year-2000-adaptation projects. The required activities were not covered by the standard methods. (The interviewee thinks that Accenture did the projects with their own standard methods.) The advantage of a special method is to have an adapted method for all individual projects within the Euro conversion project. Otherwise too much effort would have been required to train all employees in the use of the standard methods for the Euro project. Therefore a central department integrated the required knowledge into a special method and made that available to all concerned.

Engineering company:

<no additional information>

Volkswagen:

Structure K-DOV

Top down:

-> VW Corporate Group

---> Division Management Organisation and Systems

-----> Area Information Systems

-------> Department IS sales = K-DOV

---------> IS sales corporate clients = K-DOV-1 (department of the interviewee)

Related areas:

K-DOV – IS sales, IT service department, provides services for all service and marketing departments of VW

K-VD – Konzern Vertrieb Deutschland: corporate sales and marketing for Germany, client of K-DOV

K-DOI – IS technology, operations of hardware, service for all other departments including K-DOV and K-VD

gedas: IT company, owned by VW but operates as an independent company

Example for relationship

K-VD needs an application to improve their relationship with corporate clients. They sign a contract with K-DOV to organise the required application for them. An external developer like IBM and gedas develops the software for the sales and marketing departments. The software is handed over to K-DOI that runs the software on their servers. K-DOV is responsible to test and manage the software. Everything is paid from the IT budget of K-VD.

SEP – System Development Process

An important element for the work of all IT departments is the VW standard SEP – System Development Process. SEP conformity is required for all projects; otherwise projects do not get the approval from ‘Controlling’.

The five phases of this process are: inception, conception, system design, system implementation, and system deployment. SEP provides activities, methods, tools, results, and support material is available for four development directions/areas:

– Classic: new and further development;
– OO: object oriented new and further development;
– SAP: implementation and adaptation;
– SSW: standard software selection.
–

sd&m

– www.sdm.de

<no additional information>

–

Debeka

– www.debeka.de

<no additional information>