Perspectives on Improving Software Maintenance

Frank Niessink

SIKS Dissertation Series No. 2000-1.

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Graduate School for Information and Knowledge Systems.

Promotiecommissie:
prof.dr. J.C. van Vliet (promotor)
prof.dr. S. Lawrence Pfleeger (University of Maryland)
prof.dr. J.M. Akkermans
prof.dr. P. Klint (University of Amsterdam)
dr.ir. J.J.M. Trienekens (Eindhoven University of Technology)
prof.dr.ir. J.L.G. Dietz (Delft University of Technology)
prof.dr. D.B.B. Rijsenbrij

VRIJE UNIVERSITEIT

Perspectives on improving software maintenance

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor aan
de Vrije Universiteit te Amsterdam,
op gezag van de rector magnificus
prof.dr. T. Sminia,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
wiskunde en informatica
van de faculteit der exacte wetenschappen
op dinsdag 28 maart 2000 om 13.45 uur
in het hoofdgebouw van de universiteit,
De Boelelaan 1105

door

Frank Niessink

geboren te Amsterdam

Promotor: prof.dr. J.C. van Vliet

Preface

Though at times research is a lonely job, I have been lucky to have had the opportunity to do most of my work in cooperation with other people and organizations. Even to such extent, that I want to acknowledge the support of the people and organizations I have worked with and that contributed directly to the research described in this thesis in a separate section (1.6). Here, I want to thank the people that have had a positive influence on my personal life and thereby also contributed to this thesis.

First, I thank my thesis supervisor prof. Hans van Vliet. He has been much more than just a supervisor, he has been a tutor, mentor and teacher at the same time. He has taught me what doing research is all about, and how to tell the story that emerges from the research.

Second, I want to thank my roommates Jacco van Ossenbruggen and Erik Boertjes and my U3-neighbours Mirna Bognar, Michel Oey, Jerry den Hartog and Frank Dehne for their pleasant company over the last four and a half years.

Third, I want to salute my other colleague's. Our daily lunches[1] provided an excellent opportunity to rant about practically everything annoying and thus blow off steam. Thanks to Arno Bakker, Gerco Ballintijn, Florentijn van Kampen, Martijn van Welie, Bastiaan Schönhage, Nico Lassing, Jaap Gordijn, Anton Eliëns, Michel Klein, Guido van't Noordende, Sijtze Douma and of course Jacco and Erik. I am also grateful to Gerco and Jacco for assisting me during the defense.

Finally, I thank my parents, my grandmother, my sister, and other family and my friends for their support. I want to especially thank my parents for teaching me to strive for maximum output – to which I've always silently added 'with minimal input'. But most of all I am grateful to Esther for her continuous love and support. I am doing my very best to maintain it.

[1] "(...) just as lunch was at the centre of a man's temporal day, and man's temporal day could be seen as an analogy for his spiritual life, so Lunch should (a) be seen as the centre of a man's spiritual life, and (b) be held in jolly nice restaurants." From *Life, the Universe, and Everything* by Douglas Adams.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The quality of software is an important aspect of software development and maintenance. Both in industry and in academia, much effort is put into methods, models, and tools that enable software engineers to maintain or improve the quality of software. We distinguish between several approaches to software quality, using two dimensions (Vliet 2000). The first dimension is the product versus process dichotomy. To improve software quality one can focus on improving the quality of the software product itself, for example by making it more user friendly or more reliable. A different approach is to improve the process that creates the software product, assuming that an improved process produces higher quality products. The second dimension is the conformance versus improvement dimension. Here, conformance approaches to software quality are targeted at conforming to some standard. Approaches targeted at improvement on the other hand, aim to implement better methods and working practices to increase quality.

Table 1.1 shows examples of each of the four approaches. ISO 9126 (ISO/IEC 1995*a*, ISO/IEC 1995*b*) is a standard for product quality which defines a tree of quality attributes, including measures to quantify those quality attributes. The 'best practices' include such practices as software configuration management, inspection, testing, etc. ISO 9000 (ISO 1987*a*) is a series of standards that states requirements for quality systems. Software Quality Assurance (SQA) procedures review and audit software processes to check whether the work is done as it should be done. Finally, the Software Capability Maturity Model (CMM) (SEI 1995), SPICE (ISO/IEC 15504) (El Emam, Drouin and Melo 1998), and Bootstrap (Kuvaja, Similä, Krzanik, Bicego, Koch and Saukkonen 1994) are all methods aimed at improving software processes by providing a reference framework against which a software organization can compare itself. Such a comparison – usually termed an assessment – results in the identification of processes and activities which are not

	Conformance	Improvement
Product	ISO 9126	'best practices'
Process	ISO 9000 SQA	Software CMM SPICE Bootstrap

Table 1.1: Four approaches to software quality (Vliet 2000)

performed adequately, thus providing directions for improvement.

A third dichotomy in the software engineering domain is the one between software development and software maintenance. Whether the difference between development and maintenance is real and has consequences for methods, skills and tools needed is the subject of debate (see e.g. Pfleeger and Rosenberg 1998, Schneidewind, Kitchenham, Niessink, Singer, von Mayrhauser and Yang 1999, Kitchenham, Travassos, von Mayrhauser, Niessink, Schneidewind, Singer, Takada, Vehvilainen and Yang 1999, Niessink and van Vliet 2000).

In this thesis we are chiefly concerned with one cross section of the three dimensions, namely the improvement of software maintenance processes. We explore two perspectives on improving software maintenance processes:

- The bottom-up, measurement-based, goal-based approach. In this case, we basically try to solve problems or reach goals by gathering relevant information, deciding on the best course of action, and implementing the solution.

  In this perspective, measurement is used as an enabler of improvement activities. The Goal/Question/Metric (GQM) paradigm is the best known method to translate improvement goals into the metrics that need to be gathered. In section 2.1 we discuss GQM and other approaches to measurement-based improvement.

- The top-down, assessment-based, maturity-based approach. Here, we use a reference framework which is assumed to contain the 'right' activities for our organization. We compare the organization with the reference framework and implement the activities that are missing in the organization, thus becoming more mature.

  Process improvement from this perspective is initiated by a comparison of the organization with an external reference. The best known example of such a reference framework is the Software Capability Maturity Model (CMM).

The Software CMM provides an ordered set of key processes that a 'mature' software organization should have implemented. The Software CMM and other reference frameworks are discussed in section 2.2.

In this chapter an overview of the research and this thesis is given. First, we discuss the context in which the research presented in this thesis has been done. Next, in section 1.2 we describe the research questions investigated. In section 1.3 the design of the research is discussed. Section 1.4 discusses the main contributions of this thesis. In section 1.5 an overview is given of the structure of the remainder of this thesis. Next, in section 1.6 the support is acknowledged of the many people and organizations that were involved in the research presented here. Finally, section 1.7 gives an overview of the work discussed in this thesis that has been published elsewhere.

1.1 Research context

The research described in this thesis was done in the course of two research projects sponsored by the Dutch Ministry of Economic Affairs and several Dutch companies. The first project, called 'Concrete Kit', ran from 1995 until 1997, and the second project, called 'Kwintes', ran from 1997 until 1999. The projects were done in cooperation with the Technical Universities of Delft and Eindhoven, and with Cap Gemini, Twijnstra Gudde and the Tax and Customs Computer and Software Centre of the Dutch Tax and Customs Administration (B/AC).

The acronym Concrete Kit stands for '*Concreti*s*e*ring van *K*waliteitsbeheersing en -verbetering: naar een nieuwe generatie *IT*-tools', meaning 'Concretizing quality control and improvement: towards a new generation IT tools'. The Concrete Kit project arose from the needs of IT organizations such as Cap Gemini and Twijnstra Gudde for quality control and improvement methods aimed at the post-development life cycle phase of IT products. The goal of the project was twofold (Rijsenbrij, Kemperman, van Vliet and Trienekens 1994):

- to gain quantitative, objective, and fundamental insight into quality with respect to the maintenance and management of IT products, and
- to develop methods, techniques, and tools to support the maintenance and management of IT products, specifically taking into account the customers and users of these IT products.

The Concrete Kit project has resulted in a method and supporting tool to specify service level agreements (SLAs) in a customer-focused way, a classification system

for incidents that occur during maintenance and management of IT infrastructures, and a simulation model for helpdesk processes.

The Kwintes project (Rijsenbrij, van Veen, Beekman, Trienekens, van Vliet and Looijen 1996) continued the research started during Concrete Kit. More emphasis was put on quantifying, evaluating, and improving IT services. Kwintes resulted in a revised SLA specification method, practical experience with measurement programs, a measurement maturity model, an information technology service maturity model, and a method to implement service processes using simulation.

Results of these projects have been published in two books (Trienekens, Zwan, Niessink and Vliet 1997, Ruijs, de Jong, Niessink and Trienekens 2000), several publications in international journals, conference proceedings, workshops and a technical report (Bouman, Trienekens and van der Zwan 1999, Horst, Niessink and van Vliet 1999, Kitchenham, Travassos, von Mayrhauser, Niessink, Schneidewind, Singer, Takada, Vehvilainen and Yang 1999, Niessink and van Vliet 1997, Niessink 1998, Niessink and van Vliet 1998*a*, Niessink and van Vliet 1998*b*, Niessink and van Vliet 1998*c*, Niessink and van Vliet 1999*a*, Niessink and van Vliet 1999*b*, Niessink and van Vliet 1999*c*, Niessink and van Vliet 2000).

1.2 Research questions

As described in the previous section, both Concrete Kit and Kwintes focused on developing methods and tools to support the delivery of high quality IT services. Information technology services are defined as activities, sold by one party – the IT service provider – to another party – the customer – to install, maintain, support the usage of, operate, or enhance information technology used by the customer. During the research projects an abstract model was developed to describe the process of delivering and managing IT services.

Figure 1.1 shows this process model. The left part of the lemniscate concerns the specification of IT services (upper arrow) and the evaluation and monitoring of the performance of the service provider (lower arrow). The right part concerns the evaluation and monitoring of service processes (upper arrow) and the design and organization of those processes. The service level agreement (SLA) plays a pivotal role in this scheme.

For example, an industrial organization uses several software systems to monitor its production processes. If the organization wants to outsource the maintenance of the software, it is important that the maintenance organization – the service provider – and the industrial organization – the customer – make explicit what services the customer will receive and with what quality, i.e. the service levels. In this case, the customer's production processes are in operation 24 hours a day, so the

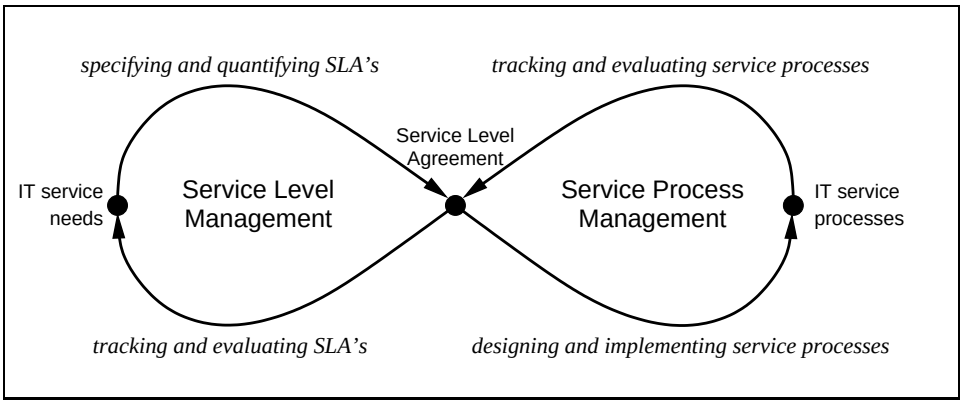


Figure 1.1: Service Level Management lemniscate (Trienekens, Zwan, Niessink and Vliet 1997)

maintenance service might need to be available 24 hours a day as well. However, in order to decide this, the service provider and customer together need to investigate the needs of the customer. This process results in a service level agreement which states what the customer can expect, but also what obligations the customer has. In this case, the service provider might demand that the customer implements its own test environment to conduct user acceptance tests of new versions of the software.

To the service provider, the service level agreement forms the basis for implementing the service processes. Depending on what service levels are agreed on, the service provider might decide to use its existing helpdesk to interface with the customer, or implement a separate helpdesk at the customer's site.

Based on the service level agreement and its service processes, the service provider will measure its performance with respect to the service levels agreed on. These measurements can be used for internal performance tracking, informing the customer about the delivered service, and for process improvement as well.

The delivered service and the service level agreement will be evaluated with the customer on a regular and event-driven basis in order to address specific problems, and to make sure the service level agreements stays in line with the possibly changing service needs of the customer.

The basic premise of the research is that in order to improve IT services, all four phases of the Service Level Management lemniscate need to be taken into account. The IT Infrastructure Library (ITIL) (Central Computer and Telecommunications Agency 1992*a*) gives detailed guidelines for many of the processes that play a role in the delivery of IT services. However, there are a number of aspects of IT service delivery that are not, or not adequately, covered by ITIL. We identified the

following research areas:

1. The structured translation of IT service needs into service level agreements (the upper-left arrow).

2. The implementation of the IT service processes, based on and in conformance with the service level agreements (lower-right arrow).

3. The usage of measurement in support of the previous two activities.

In addition, as a result of a number of case studies done on these three research areas, a fourth research area was identified:

4. It seems that we can distinguish between mature and immature IT service providers, based on certain processes that an IT service provider has implemented. Is IT service process maturity a useful concept to support the improvement of IT services? If so, what characterizes a mature IT service provider and which are the processes that the service provider should implement?

The research described in this thesis focuses on the last two of the four research areas. The other two research areas were addressed by other partners in the Concrete Kit and Kwintes projects. In the next section we describe how the generic questions above were concretized and how the research was conducted.

1.3 Research design

As mentioned in the previous section we focus on the following two research issues in this thesis:

- *Measurement-based improvement*: the usage of measurement for the improvement of IT services and IT service processes.

- *Maturity-based improvement*: the concept of IT service process maturity as a means to guide the improvement of IT services and IT service processes.

These research issues coincide with the two perspectives on process improvement mentioned on page 2. Section 1.3.1 details the first research issue – the measurement-based perspective on process improvement – and section 1.3.2 deals with the second research issue – the maturity-based perspective on process improvement.

1.3.1 Measurement-based improvement

We have concretized the first research area into the following research questions:

1. How to introduce measurement in an IT service organization? What are the necessary steps to set up a measurement program[1] and in which order should they be performed?

2. What are the prerequisites that need to be satisfied in order to improve the likelihood of success of the measurement program?

3. What is – or what should be – the relation between measurement and the maturity of the IT service organization?

Obviously, these research questions are still very broad. In the previous section, we defined IT services as 'activities [. . .] to install, maintain, support the usage of, operate, or enhance information technology used by the customer.' While all IT services concern supporting an organization in its sustained use of information technology, the range of activities needed to deliver IT services is wide. We limit the three research questions listed above to one type of IT service – software maintenance – for the following reasons:

- By limiting the types of IT services investigated, it will be easier to compare research done in different organizations, since all organizations will be software maintenance organizations.

- When doing research into software measurement, gathering enough data in a short period of time to perform statistical analysis is difficult. Individual changes to software usually take limited time and effort to implement, but at the same time are often treated as small projects. Hence, we expect to be able to measure a fair amount of attributes of individual changes, for example effort data, design measures, code measures, etc. At the same time, because individual changes are usually not too big, we also expect to be able to measure a fair number of changes in a limited period of time.

- Available expertise in the area of software engineering at the Vrije Universiteit, e.g. Vliet (2000).

In addition, we have limited the possible measurement applications to the planning and estimation of software maintenance effort, again to improve the possibilities to compare research across organizations.

[1] 'Measurement program' is the phrase used for measurement activities and processes in a (software) organization.

Category	Validation method
Observational	Project monitoring: collect development data *Case study*: monitor project in depth Assertion: ad hoc validation techniques Field study: monitor multiple projects
Historical	Literature search: examine previously published studies *Legacy data*: examine data from completed projects Lessons learned: examine qualitative data from completed projects Static analysis: examine structure of developed product
Controlled	*Replicated experiment*: develop multiple versions of product Synthetic environment experiment: replicate one factor in laboratory setting Dynamic analysis: execute the developed product for performance Simulation: execute product with artificial data

Table 1.2: Summary of software engineering validation models (Zelkowitz and Wallace 1998)

Four case studies were done in four different software maintenance organizations to investigate these questions. All four case studies concerned the planning and estimation of software maintenance effort. In terms of the taxonomy developed by Zelkowitz and Wallace (1997, 1998), depicted in table 1.2, two of the case studies are 'legacy data' because we investigate the measurement programs a posteriori. The other two measurement programs are a combination of a 'case study' and a 'replicated experiment'. They are replicated experiments in the sense that in both organizations the measurement program was implemented in the same way, using the same steps. However, we obviously could not control all factors that are relevant, such as the software process used, so from that point of view the two measurement programs were actually case studies.

1.3.2 Maturity-based improvement

As mentioned in section 1.2, the second research issue emerged as a result of a number of case studies done during the research projects Concrete Kit and Kwintes. It was observed that the success of the application of our methods during the case studies depended on certain characteristics of the organization, such as experience with service level agreements, the existence of standard work procedures, the way

in which incidents were managed, etc. We hypothesized that some organizations were more 'mature' with respect to IT service processes than other organizations and that the maturity of the organizations was an important factor in explaining case study success.

This research issue was concretized into three questions:

1. What arguments can we supply to support the notion of IT service process maturity?
2. How should mature IT service organizations look? Which processes should a mature service provider implement?
3. How can we use the concept of IT service process maturity in practice to support the improvement of IT service providers?

Again, we focus on software maintenance as one possible IT service to reduce the complexity of the research.

The first of these three questions was investigated through literature research. The second question was explored by developing a maturity model in close cooperation with experts in the field of IT services. The last question was examined by performing service process assessments of two software maintenance and support organizations. These case studies are a first step towards validation of the developed maturity model for IT service providers.

1.4 Main contributions

The main contributions of this thesis are the following.

In part I we describe four measurement program case studies. From these four case studies we learn several lessons with respect to success and failure factors of measurement programs. We introduce an abstract process model of measurement-based process improvement. We use this process model to compare different guidelines and frameworks for implementing measurement programs. The comparison reveals that these guidelines agree on the basic activities needed for successful measurement, but at the same time emphasize different aspects. In addition, the usage of the abstract process model shows that these guidelines tend to ignore the *application* of measurement results.

We conclude that the consensus success factors for measurement programs as found in the literature are necessary but not sufficient preconditions for the successful implementation of measurement programs. These, what we call 'internal' success factors, need to be complemented with 'external' success factors that are aimed at securing that measurement programs generate value for the organization.

We propose four external success factors and we suggest a number of activities that can be used to adhere to these external success factors.

In part II we present a new perspective on software maintenance, namely software maintenance as a *service.* Viewing software maintenance as a service implies several consequences for the way in which customers will judge the quality of software maintenance, and hence it has consequences for the processes that are key to delivering high quality software maintenance. We lay down these consequences in a capability maturity model for IT services. We show some preliminary experiences with applying the IT Service CMM in the assessment of two organizations that provide IT services.

1.5 Structure of this thesis

This thesis is structured as follows. In the next chapter we first give a brief overview of the literature with respect to process improvement. As mentioned in this chapter, we distinguish between two perspectives on process improvement: measurement-based improvement and maturity-based improvement. The remainder of the thesis is also structured according to these two perspectives. Part I discusses the research done on software maintenance process improvement from a measurement perspective, part II presents the research done on improving software maintenance from a maturity perspective.

Part I presents four measurement program case studies. Chapters 3 and 4 analyze two measurement programs in retrospect. Chapter 5 reports on two measurement programs implemented in two different organizations with the help of two graduate students. Chapter 6 synthesizes the experiences from the four measurement programs in a measurement maturity model. In the last chapter of part I a first attempt is made to validate the measurement maturity model. The model is compared against related work using an abstract model of the measurement-based improvement process. From the comparison we conclude that there is a fair amount of consensus about the issues that are important in measurement. However, the improvement part of the measurement-based improvement process remains under exposed. A number of activities is proposed to supplement it.

Part II starts with chapter 8 which discusses software maintenance from a service point of view. Specifically, attention is given to the differences between product development and service delivery, and how this applies to software maintenance. We argue that the differences between services and products cause a need for different processes to deliver high quality software maintenance, than present in current maturity models for software development. Chapter 9 presents the IT Service Capability Maturity Model which is aimed to provide these key processes

specifically needed for IT service provision. Chapter 10 describes two case studies during which two software maintenance and support organizations were assessed against the IT Service CMM.

Finally, chapter 11 presents the conclusions of this thesis.

1.6 Acknowledgments

This thesis could not have been accomplished without support from many organizations and individuals. I would like to acknowledge the support of the following organizations:

- Senter, the funding agency of the Dutch Ministry of Economic Affairs that sponsored both Concrete Kit (nr. ITU94045) and Kwintes (nr. ITU96024).
- The principals of Concrete Kit and Kwintes: Cap Gemini, Twijnstra Gudde, and the Tax and Customs Computer and Software Centre of the Dutch Tax and Customs Administration.
- The participating universities: the technical universities of Delft and Eindhoven and of course my own university, the Vrije Universiteit.
- And other organizations that participated in the case studies: ASZ (Automatisering Sociale Zekerheid), Hoogovens, the Dutch ministry of Transport, Public Works and Water Management, and Ericsson Telecommunications.

Also, I would like to acknowledge the support and cooperation of many individuals of the above mentioned organizations:

- The people who participated in the Concrete Kit and Kwintes projects and case studies, especially: Rob Akershoek, Koos Berg, Jacques Bouman, Willem Bor, Paul van der Cruijs, Ivo David, Guus Delen, Jan Derksen, Richard van den Dool, Anneke Duijts, Frank Eggink, Wouter de Jong, Ben Kooistra, Herman Laferte, Yolanda Louwinger, Toos Lubbers, Hanna Luden, Siemen Mandema, Gerrit Matser, Peter Nooteboom, Gert Jan Peek, Daan Rijsenbrij, Leo Ruijs, Lennart Schaftenaar, Johann Schreurs, Douwe Schumacher, Thijs Sommen, Dick Spijker, Hajo Stoel, Jos Trienekens, Albert van Veen, Leo van Veen, Diederik Verwoerd, Alexander Westra, Mark de Wijn, Kees Wittebrood, and Mark van der Zwan.
- Master students did case studies or provided other useful input: Kit Lam, Jack Hulscher, Jeroen Bögels, Micha van der Velden, Laura de Vries, and Mostapha Chriqui.

- Fellow authors: Hans van Vliet, Jos Trienekens, Wouter de Jong, Leo Ruijs, Mark van der Zwan, Bart ter Horst, Barbara Kitchenham, Guilherme Travassos, Anneliese von Mayrhauser, Norman Schneidewind, Janice Singer, Shingo Takada, Risto Vehvilainen and Hongji Yang.

In addition, useful advice on early ideas and papers was given by Rob Donnellan and Shari Lawrence Pfleeger.

Finally, we acknowledge that CMM and Capability Maturity Model are trademarks of Carnegie Mellon University and that IDEAL is a service mark of Carnegie Mellon University.

1.7 Publications

Most of the material presented in this thesis has been published elsewhere. This section gives an overview of previously published work.

The case studies in chapters 4 and 5 were presented at the International Conference on Software Maintenance (Niessink and van Vliet 1997, Niessink and van Vliet 1998*c*). The work on measurement maturity in chapter 6 was presented at the Euromicro Conference on Software Maintenance and Reengineering (Niessink and van Vliet 1998*b*). The material in section 7.1 and section 7.3 is described in (Niessink and van Vliet 1999*a*). Section 7.2 is a summary of the work presented at the European Software Control and Metrics Conference (Horst, Niessink and van Vliet 1999). The external success factors for measurement-based improvement in section 7.4 were presented at the International Software Metrics Symposium (Niessink and van Vliet 1999*b*).

The difference between software maintenance and software development from a service point of view as discussed in chapter 8 was first presented at the Workshop on Empirical Studies of Software Maintenance (Niessink 1998). The workshop resulted in a paper in the Journal of Software Maintenance by nine participants (Kitchenham, Travassos, von Mayrhauser, Niessink, Schneidewind, Singer, Takada, Vehvilainen and Yang 1999). A paper discussing software maintenance from a service perspectives is to appear in the Journal of Software Maintenance (Niessink and van Vliet 2000). The case studies described in section 8.3 and an earlier version of the IT Service CMM as presented in chapter 9 were first published in the journal Software Process – Improvement and Practice (Niessink and van Vliet 1998*a*).

Chapter 2

Related Work

In this chapter we take a look at the literature on process improvement. The aim of this chapter is to provide an overview of the available work. In later chapters we will discuss additional related work where appropriate.

As described in the introductory chapter, we are interested in the improvement of the quality of software maintenance by improving the software maintenance process. Looking at process improvement methodologies, we can distinguish between two perspectives. This division is based on the source of the reference against which improvement is tracked.

Internal reference-based process improvement

This is the perspective that was loosely termed 'bottom-up, measurement-based, goal-based' improvement in chapter 1. The methodologies in this category focus on process improvement with respect to internally determined improvement goals. These methodologies all more or less implement the 'scientific method': based on some observation or question, a hypothesis is formulated, which is next tested by performing an experiment. Based on the outcome of the experiment, the hypothesis can be rejected or (provisionally) accepted, and the organization can be improved by employing the results.

The main example from this category in the software engineering domain is the Goal/Question/Metric (GQM) paradigm.

External reference-based process improvement

The second category, that was called the 'top-down, assessment-based, maturity-based' approach in chapter 1, consists of all methodologies that provide organizations with a prescriptive reference framework against which organizations can assess their processes and find directions for improvement.

> The main example in this category is the Software Capability Maturity Model. However, not only maturity models such as the Software CMM and Trillium belong to this category. Other examples of reference frameworks are collections of best practices such as the IT Infrastructure Library and standards such as the ISO 9000 series.

This subdivision concurs with the subdivisions made by Solingen and Berghout (1999) and Bøegh, Depanfilis, Kitchenham and Pasquini (1999). Solingen and Berghout distinguish between 'top-down approaches' to software process improvement, like the Software CMM, and 'bottom-up approaches', like the Goal/Question/Metric paradigm and the AMI (Application of Metrics in Industry) approach (Pulford, Kuntzmann-Combelles and Shirlaw 1996). In addition, Solingen and Berghout use the phrases 'assessment-based' and 'measurement-based' for that same subdivision. Bøegh, Depanfilis, Kitchenham and Pasquini distinguish three types of software quality methodology: process improvement methodologies, like the Software CMM, metrics methodologies, like GQM, and product quality methodologies, like the ISO 9126 standard.

In the remainder of this thesis we will use 'measurement-based improvement' to refer to internal reference-based process improvement, and 'maturity-based improvement' to refer to external reference-based process improvement. Though the two longer terms indicate the difference between the two classes of approaches more precisely, we will use the shorter phrases instead. Not only because they are more convenient, but also because the research in this thesis is focused on the measurement aspect of internal reference-based process improvement and on the maturity aspect of external reference-based process improvement. Hence, we do not attempt to cover all aspects of software measurement and software process improvement in this chapter. For a general overview of software measurement the reader is referred to Fenton and Pfleeger (1997). An in-depth discussion of the history and theory of software measurement can be found in Zuse (1998). Thomson and Mayhew (1997) give an overview of software process improvement approaches. Zahran (1997) provides an extensive overview of maturity-based process improvement methodologies.

In the next section we discuss related work on measurement-based process improvement and in section 2.2 we discuss maturity-based improvement methodologies.

2.1 Measurement-based improvement

Internal reference-based improvement methodologies take internal sources as reference for the improvement activities. Usually this means that the business strategy

1)	Characterize the current project environment.
2)	Set up goals and refine them into quantifiable questions and metrics for successful project performance and improvement over previous project performances.
3)	Choose the appropriate software project execution model for this project and supporting methods and tools.
4)	Execute the chosen processes and construct the products, collect the prescribed data, validate it, and provide feedback in real-time.
5)	Analyze the data to evaluate the current practices, determine the problems, record the findings, and make recommendations for improvement.
6)	Proceed to step 1 to start the next project, armed with the experience gained from this and previous projects.

Table 2.1: Quality Improvement Paradigm (Basili and Rombach 1988).

and business goals of the organization form the 'leitmotiv' in the improvement efforts. The business goals are translated into improvement goals for the software organization, and these are next translated into measurement goals. A measurement program is used to fulfill the measurement goals, and based on the outcome of the measurement program, decisions can be taken and improvements can be implemented to reach the improvement goals.

Most measurement and improvement methods used in the software engineering domain build upon the Goal/Question/Metric paradigm (Basili and Rombach 1988). The Goal/Question/Metric (GQM) paradigm offers a structured approach to translate high level measurement goals into questions that need to be answered to reach the goals, which in turn lead to the metrics needed to answer the questions. In section 2.1.1 we give an overview of GQM and other goal-based measurement approaches.

Many reports on the implementation of measurement programs mention specific success factors that contribute either to a successful or unsuccessful measurement program. Researchers have aggregated these success factors into lists that should provide guidance to organizations that want to implement measurement programs. We discuss these measurement program success factors in section 2.1.2. Other researchers have tried to capture the processes needed for successful measurement in maturity models. We look at two of these in section 2.1.3.

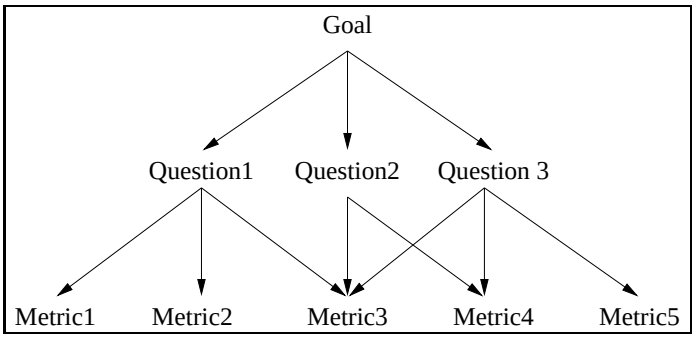


Figure 2.1: Goal/Question/Metric tree

2.1.1 Goal-based measurement

Most goal-based measurement approaches are based on the Quality Improvement Paradigm (QIP) and the Goal/Question/Metric (GQM) paradigm, developed by Basili and Rombach (1988) in the TAME (Tailoring A Measurement Environment) project. The Quality Improvement Paradigm consists of six major steps, see table 2.1. It aims to provide a basis for organizational learning and improvement by facilitating learning from experience in projects and feeding this experience back to the organization. Each new project is regarded as an experiment and available results of every foregoing and ongoing experiment should be packaged and reused. The QIP has a continuous character which is implemented by two feedback cycles: the project feedback cycle and the corporate feedback cycle. The project feedback cycle provides new projects with information about foregoing projects. The corporate feedback cycle provides knowledge to the complete organization by comparing individual projects with the aggregated project data (Solingen and Berghout 1999).

The Goal/Question/Metric (GQM) paradigm is a method which helps determining which measures should be taken to support reaching certain measurement goals, and hence can be used to implement step 2 of the quality improvement paradigm. Based on the measurement goals, questions are formulated that need to be answered. Next, the questions lead to metrics that need to be measured. The information thus gathered provides the answers to the questions. This leads to a tree – or rather, a directed acyclic graph – of goals, questions, and metrics, see figure 2.1 for an example.

The method provides templates which can be used to formulate the goals. The following example originates from research into the causes and effects of interrupts on software development work (Solingen, Berghout and Latum 1998, Solingen and Berghout 1999). The following measurement goal was defined:

Analyze:	interrupts and their effects
for the purpose of:	understanding
with respect to:	- impact on schedule
	- the cost/benefit of interrupts
from the viewpoint of:	project team
in the following context:	project X

Next, questions were formulated that needed to be answered to reach the goal, for example:

- What is the influence of interrupts on the work that was interrupted?
- What factors influence treatment and effort for handling an interrupt?
- Is prevention of interrupts possible?

These questions formed the basis for selecting metrics that should provide answers to these questions. Examples of such metrics were:

- Number of interrupts for current work.
- Number of interrupts for not current work.
- Number of interrupts that are treated immediately.
- Number of interrupts that are planned.
- Number of interrupts per day.

Based on these measurements it was discovered that more interrupts occurred than was expected, and that the department spent about 20% of its total time on handling interrupts. Based on these and other outcomes, action points were defined and implemented to, for example, reduce the number of personal visits in favor of e-mail communication.

GQM has been applied in measurement programs quite often (see for example Basili, Briand, Condon, Kim, Melo and Valett 1996, Birk, van Solingen and Järvinen 1998, Birk, Derks, Hamann, Hirvensalo, Oivo, Rodenbach, van Solingen and Taramaa 1998, Latum, van Solingen, Oivo, Hoisl, Rombach and Ruhe 1998, Fuggetta, Lavazza, Marasca, Cinti, Oldano and Orazi 1998, Panfilis, Kitchenham and Morfuni 1997). These case studies resulted in a number of proposed extensions of or additions to the Goal/Question/Metric method. For example, Panfilis, Kitchenham and Morfuni (1997) report on two extensions they made:

- they found it necessary to rigorously define measures in terms of entities, attributes, units, and counting rules, and,
- the initial GQM plan was subjected to an independent review.

Both Fuggetta, Lavazza, Marasca, Cinti, Oldano and Orazi (1998) and Latum, van Solingen, Oivo, Hoisl, Rombach and Ruhe (1998) stress the necessity of being able to characterize the process structure. This concurs with the recommendations of Pfleeger and McGowan (1990) to take the maturity of the software process into account when selecting measures. Solingen, Lattum, Oivo and Berkhout (1995) describe an extension of GQM (model-based GQM) that includes explicit models of the software process and products. Metrics are next defined according to both the standard Goal/Question/Method method, as well as from the perspective of the process and product models. Both sets of metrics are mutually checked for consistency and completeness.

Though many authors stress the necessity of deriving metrics from specific improvement and measurement goals, not all researchers favor this top-down approach. For example, Hetzel (1993) advocates a bottom-up approach in which a basic set of measurements is defined that is to be measured during every project. The underlying principle behind his bottom-up measurement engineering model is that the primary role of measurement is to support engineering activities. It should stimulate questions and provide insight about the software process and products.

Still, the top-down approaches based on GQM seem to be the ones most used when it comes to implementing measurement programs.

2.1.2 Measurement program success factors

Several authors have synthesized lists of success factors for implementing measurement programs, based on experiences reported in the literature. Hall and Fenton (1997) identify a number of consensus success factors for the implementation of measurement programs. Table 2.2 shows these factors, that were identified after studying other literature, such as Grady (1992) and Pfleeger (1993).

We can distinguish between two groups among the success factors listed by Hall and Fenton: one group is targeted at obtaining acceptance from the developers involved in the measurement program. This category includes the factors 4-9, 11, 13-15. Each of these factors aims to convince the developers that the measurement program is rigorous, useful and not used *against* the developers. The second category consists of factors aimed at a gradual introduction and enhancement of the measurement program: the measurement program should be incrementally implemented, constantly improved, use existing materials, be supported by management, and a well-planned metrics framework should be used (1-3, 10, 12).

1	Incremental implementation
2	Well-planned metrics framework
3	Use of existing metrics materials
4	Involvement of developers during implementation
5	Measurement process transparent to developers
6	Usefulness of metrics data
7	Feedback to developers
8	Ensure that data is seen to have integrity
9	Measurement data is used and seen to be used
10	Commitment from project managers secured
11	Use automated data collection tools
12	Constantly improving the measurement program
13	Internal metrics champions used to manage the program
14	Use of external metrics gurus
15	Provision of training for practitioners

Table 2.2: Consensus success factors

We will use the success factors of Hall and Fenton in the remainder of this thesis (specifically in chapters 3, 4, and 5) as a means to assess the measurement programs investigated.

2.1.3 Measurement maturity

Inspired by the Software CMM some researchers have captured guidelines for measurement programs in the form of maturity models. We discuss two of these measurement maturity models in chapter 7, namely the software measurement technology maturity framework by Daskalantonakis, Yacobellis and Basili (1990-1991) and the process model of software measurement by Comer and Chard (1993).

Unfortunately, we know of no empirical research on the actual usage of these maturity models. For the software measurement technology maturity framework of Daskalantonakis, Yacobellis and Basili a maturity questionnaire has been developed by Budlong and Peterson (1995). However, we are not aware of any publications on the application of measurement maturity assessments and the questionnaire.

2.2 Maturity-based improvement

The basic scheme of external reference-based improvement approaches is a reference framework – defined externally to the organization in question – which *prescribes* the activities, methodologies, practices, and/or tools an organization should implement and/or use.

Some approaches structure the framework in levels in order to facilitate implementation. There are two ways to apply so-called 'maturity levels' to a framework (Zahran 1997):

- *Staged model*: The staged model comprises a number of maturity levels, and each process or process area is tied to a certain level. At each level, an organization implements the processes attached to that level. If we use the term measurement loosely, we can say that a staged model 'measures' the maturity of a complete organization.

  The underlying logic of staged models is that the processes on a certain level form the foundation for the next levels. So, skipping processes or levels is generally not advised, because all processes on and below a certain level are needed for the next level. The Software CMM (Paulk, Curtis, Chrissis and Weber 1993, Paulk, Weber, Garcia, Chrissis and Bush 1993, SEI 1995) is an example of a staged model with five maturity levels, ranging from level 1 – the initial level – to level 5 – continuous improvement.

- *Continuous model*: In a continuous model the processes themselves can be rated along a maturity scale. So the model measures the maturity of individual processes instead of the maturity of an organization. In a continuous model, as opposed to a staged model, it is possible for one process to be implemented at a low level of maturity and another process at a high level of maturity. The SPICE (ISO 15504) (El Emam, Drouin and Melo 1998) model is an example of a continuous model. The maturity (capability) of individual processes can range from level 0 – incomplete – to level 5 – optimizing.

Several authors have compared the Software CMM with the SPICE model (Garcia 1997, Paulk, Konrad and Garcia 1995, Paulk, Garcia and Crissis 1996). Paulk et al. mention as the advantage of a staged architecture that it focuses on the 'vital few' areas that typically block process performance at a particular stage in the organization's life. The maturity levels prioritize general software problems. The advantage of a continuous model is that it provides a more detailed overview of the maturity of an organization by measuring the maturity of individual processes.

A comparison of ISO 9001 and the Software CMM is given by Paulk (1995). Paulk concludes that the biggest difference between the two is the focus of the Soft-

	Assessment	No assessment
Maturity levels	Software CMM SPICE (ISO 15504) Trillium Bootstrap	
No maturity levels	ISO 9000 series	ITIL

Table 2.3: Process improvement methodologies.

ware CMM on continuous process improvement, where ISO 9001 only addresses the minimum requirements for an acceptable quality system. Because of the different requirements of ISO 9001 and the Software CMM, there is no direct mapping between the two. Paulk concludes that an ISO 9001-compliant organization will be somewhere between level one and level three of the Software CMM. Conversely, a Software CMM level two or three organization will probably be considered ISO 9001 compliant.

Often, an assessment method accompanies the process improvement framework to facilitate the comparison of organizational practices with the practices as prescribed by the framework. Generally, three types of assessment are distinguished (Zahran 1997):

- *Self-assessment*: This refers to the situation where the assessment is performed by the organization itself, and mainly by its own personnel. The main objective in this case is to identify the organization's own process capability and initiate a plan for process improvement.

- *Second-party assessment*: In this case the assessment is performed by external assessors and the objective is to evaluate the organization's capability to fulfill specific contract requirements.

- *Third-party assessment*: Here an independent third-party organization performs the assessment. The main objective in this case is to verify the organization's ability to enter contracts or produce software products, and sometimes to provide the fulfillment of certification according to a selected standard.

  The last two variants are also known as 'capability determination'.

Table 2.3 shows examples of different types of improvement methodologies. All maturity frameworks, such as the Software CMM, SPICE/ISO 15504, and Trillium (Trillium 1996), are accompanied by assessment methods. A Software CMM

assessment can be done using the Software Capability Evaluation method (Byrnes and Phillips 1996). The ISO 15504 standard includes a framework for conducting assessments, together with guidelines for the use of the framework in two different contexts: when used for process improvement, i.e. self-assessment, and when used for process capability determination (Rout 1995).

The ISO 9000 series standards, including the ISO 9000-3 guidelines, provide an international standard for quality management and assurance that can be applied to software development and maintenance (Schmauch 1995, ISO 1987*a*, ISO 1987*b*, ISO 1987*c*, ISO 1987*d*, ISO 1991). ISO 9000 certification can be obtained through a registration audit by an accredited, third-party registrar. Guidelines for auditors for conducting audits are described in ISO 10011 (ISO 1990). Schmauch (1995) gives a list of questions that can be used for a ISO 9000 self-assessment. However, there is no formal procedure for performing a self-assessment.

In addition to a reference framework, an improvement implementation method is needed to organize and enable the implementation of changes. The Software Engineering Institute has developed the IDEAL (Initiate, Diagnose, Establish, Act, Leverage) software process improvement method (McFeeley 1996). The method starts with an initiating phase during which the initial improvement infrastructure is established. A software process improvement plan is created to guide the organization through the completion of the other phases. The method uses a capability assessment during the diagnosing phase to determine the current state of the organization. During the establishing phase, the issues found during the diagnosis are prioritized and an action plan is developed. The acting phase consists of the piloting and deployment of new and improved processes. Finally, during the leveraging phase the experiences are evaluated and packaged into a process database to provide input for the next cycle through the model.

In the remainder of this section we discuss the Software CMM and ITIL. The Software CMM is the most well-known maturity model for software process improvement, claimed to be applicable to both software development and software maintenance processes. The IT Infrastructure Library is a set of best practices aimed at IT service providers, and should as such be suitable for software maintenance providers as well.

2.2.1 The Software Capability Maturity Model

The Software CMM measures a software organization's software process capability on a five-level ordinal scale. The software process capability is defined as the range of expected results that can be achieved by following a software process (SEI 1995, p. 9). The model distinguishes the following five maturity levels:

1. *Initial*: The software process is characterized as ad hoc, and occasionally

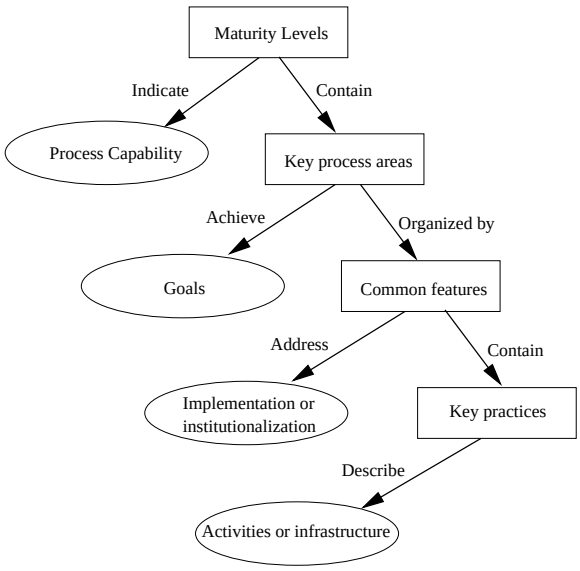


Figure 2.2: The CMM structure (SEI 1995)

even chaotic. Few processes are defined, and success depends on individual effort and heroics.

2. *Repeatable*: Basic project management processes are established to track cost, schedule and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

3. *Defined*: The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. All projects use an approved, tailored version of the organization's standard software process for developing and maintaining software.

4. *Managed*: Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled.

5. *Optimizing*: Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.

Each maturity level is characterized by a number of processes that an organization residing on that level should perform. These processes are grouped in key process areas, see figure 2.2. Each key process area consists of activities that an organization needs to implement. These activities are grouped according to their common features. Five common features are distinguished: commitment to perform, ability to perform, activities performed, measurement and analysis, and verifying implementation. Commitment to perform activities are aimed at securing management commitment by means of organizational policies and establishing leadership. Ability to perform activities are targeted at fulfilling preconditions necessary to successfully implement the software process. Examples are training, resources, and organizational structures. Activities performed describe the activities, roles, and procedures necessary to implement a key process area. Examples are the development of plans, performing the work, tracking it, and taking corrective actions when necessary. Measurement and analysis activities describe the basic measurement practices that are needed to determine the status of the process. The measurements are used to control and improve the process. Finally, verifying implementation activities are aimed at ensuring that activities are performed in compliance with the established process. Typically, reviews and audits by management and the software quality assurance group are used to check this.

Table 2.4 shows the key process areas in the Software CMM. Three process categories are distinguished in the Software CMM: management processes are related to establishing basic project management controls. Organizational processes are aimed at establishing an infrastructure that institutionalizes effective software engineering and management process across projects. Engineering processes are focused on performing a well-defined engineering process that integrates all software engineering activities to produce correct, consistent software products effectively and efficiently.

The first processes that a software organization needs to implement are the six management processes at level two. These processes focus on the management of individual software projects. Organization-wide issues are tackled at the third level. Below we discuss the key process areas of level two. For a description of the key process areas of the third and higher levels we refer the reader to SEI (1995). The six level two processes have the following purposes (SEI 1995):

- *Requirements management* is aimed at establishing a common understanding between the customer and the software project of the customer's requirements to be addressed by the project. The requirements form the basis for managing and planning the software project.

- The purpose of *software project planning* is to establish reasonable plans for

Levels	Management	Organizational	Engineering
Optimizing	Process change management	Technology change management	Defect prevention
Managed	Quantitative process management		Software quality management
Defined	Integrated software management Intergroup coordination	Organization process focus Organization process definition Training program	Software product engineering Peer reviews
Repeatable	Requirements management Software project planning Software project tracking and oversight Software subcontract management Software quality assurance Software configuration management		
Initial	Ad hoc processes		

Table 2.4: The CMM key process areas and process categories (SEI 1995)

performing the software engineering activities and for managing the software project.

- The *software project tracking and oversight* key process area is aimed at establishing adequate visibility into actual progress so that management can take effective actions when necessary.
- When applicable, *software subcontract management* is used to select qualified software subcontractors and manage them effectively.
- *Software quality assurance* provides management with appropriate visibility into the process being used and the products being built.
- The purpose of *software configuration management* is to establish and main-

tain the integrity of the products of the software project, throughout the project's software life cycle.

The Software CMM is said to be applicable both to software development organizations as well as to software maintenance organizations. However, the wording of the key practices is oriented towards software development. Drew (1992) reports on his experiences applying the Software CMM (level two) to a software maintenance organization. He concludes that the issues that arise when applying the Software CMM to a software sustaining organization are significant, but do not prohibit the use of the model. The main difficulties arose with the treatment of requirements and differences in project management needs. The Software CMM treats the requirements as the basis for software projects, whereas in the case of software maintenance, change requests and problem reports are the basis for the maintenance activities. Similarly, project planning activities are aimed at managing large projects, whereas software maintenance projects resulting from change requests and problem reports are of widely different sizes and can be very small.

The Software Capability Maturity Model has received quite some attention in the literature, both positive and negative. The criticism is aimed at two main issues: the maturity model itself on the one hand and the assessment procedure on the other hand. The main points are the following (Bach 1994, Bollinger and McGowan 1991, Gray and Smith 1998, Fayad and Laitinen 1997, Ould 1996):

- The model lacks a formal theoretical basis. It is a 'made up' model, which describes an ideal organization which never existed. Whether the model indeed contains activities needed for a mature software organization is unknown.

  For example, Ould (1996) criticizes the use of statistical analysis of the performance of the processes in use. He claims that treating software development as a manufacturing activity is an unhelpful demand, because it is not a manufacturing process, but an intellectual and sociological activity prone to many changes. Such changes include changes in the type of work being done, changes in the technologies to be used, changes in staff or recruitment policy, etc. Hence, Ould argues that using statistical process control (SPC) is neither appropriate nor necessary for defect prevention and process improvement.

- Construct validity and predictive validity of 'software process capability'. 'Are software process assessments to measure a notional concept such as IQ?' If so, then questions about its nature need to be answered. For example, whether something like 'software process capability' does really exist,

whether it is static or evolving through time, and whether it can be measured. In addition, we need to be able to determine its predictive validity.

- The repeatability and reproducibility of process assessments. The questionnaires used to determine the capability of an organization need to be interpreted. Is this done consistently?

  Fusaro, El Emam and Smith (1998) investigated the reliability of the 1987 CMM maturity questionnaire and the SPICE (version 1) capability dimension and concluded that both assessment instruments have a high internal consistency. Internal consistency is one aspect of reliability, other aspects such as interrater agreement need to be studied as well. However, no published reports on interrater agreement studies of Software CMM assessments are known. There are a number of studies into internal consistency and interrater agreement for SPICE assessments (see El Emam and Goldenson 1996, El Emam, Briand and Smith 1996, Simon, El Emam, Rousseau, Jacquet and Babey 1997, El Emam, Simon, Rousseau and Jacquet 1998, El Emam 1998, El Emam 1999).

  In addition, the questionnaire addresses a subset of all the activities performed by software organizations. Are all relevant activities included? Are irrelevant activities excluded? Also, the scoring scheme used for software capability evaluation (Byrnes and Phillips 1996) is criticized: Bollinger and McGowan (1991) show that only 12 questions determine whether an organization is assessed at either level one or two.

- Irrelevance, incompleteness and questionable organizational impact of software process assessment and resultant improvement plans. Do process assessments assess the right processes, when is the organization ready for improvement and how to ensure that the organization focuses on the relevant issues?

  McGarry, Burke and Decker (1998) analyzed over 90 software projects in one organization where information was available characterizing both the end software product as well as the methods and general processes used to produce that product. They conclude amongst other things that there was not a significant correlation between quantified process maturity and the product measures used to assess product quality. In addition, software productivity and software defect rates improved consistently over a 14-year period, independent of the software process activities implemented based on Software CMM improvements.

- Economics and cost effectiveness of process assessments. Is a Software

> CMM based improvement program worth the costs involved in (repeatably) assessing the organization, and planning and implementing key practices?
>
> According to Herbsleb, Zubrow, Goldenson, Hayer and Paulk (1997) it takes about two years per level to get from level one to level three of the Software CMM. The cost ranges from $500 to $2000 per employee per year. Though return on investments of 7.7 to 1 have been reported (Diaz and Sligo 1997), the question remains whether all process improvement programs are that successful.

Despite the criticism listed above, quite some positive experiences with the Software CMM have been published as well. For example, Humphrey, Snyder and Willis (1991) report on the software process improvement program at Hughes Aircraft which estimates its annual savings to be about $2 million. Other positive experiences are published as well (Daskalantonakis 1994, Diaz and Sligo 1997, Dion 1993, Hollenbach, Young, Pflugrad and Smith 1997). Empirical results are regularly summarized by the Software Engineering Institute (e.g. Herbsleb, Zubrow, Goldenson, Hayer and Paulk 1997).

2.2.2 IT Infrastructure Library

According to the Central Computer and Telecommunications Agency (1992*b*), the primary objective of the IT Infrastructure Library (Central Computer and Telecommunications Agency 1992*a*) is 'to establish best practices and a standard of IT service quality that customers should demand and providers should seek to supply.' ITIL was originally developed by the British government through their Central Computer & Telecommunications Agency (CCTA). Nowadays, ITIL is being maintained by the Netherlands IT Examinations Institute (EXIN). We do not know of any scientific literature describing experiences with applying ITIL. The description below is solely based on the IT Infrastructure Library itself.

Note that ITIL uses a different definition of what an IT service entails, than we use in this thesis (see page 4). ITIL defines an IT service as 'a set of related functions provided by IT systems in support of one or more business areas, which in turn may be made up of software, hardware and communication facilities, perceived by the customer as a coherent and self contained entity'. In our definition an IT service is sold by one party to another party, the ITIL definition defines an IT service as functionality provided by an IT system.

ITIL uses a layered view of IT service management, see figure 2.3. Customers of IT services receive their service through IT service provision activities, such as user support, bug fixes, new releases, replacement of broken hardware, etc. The IT service provision activities build upon the IT infrastructure. The management

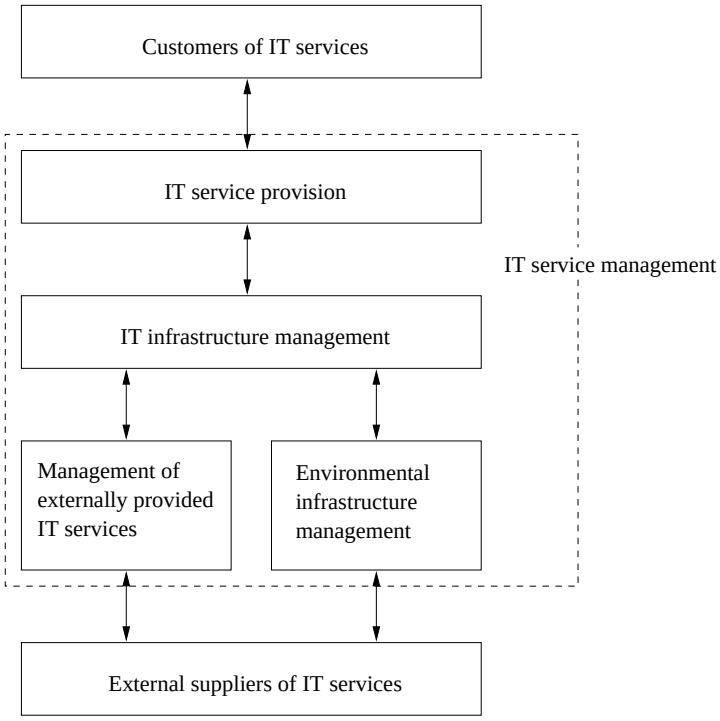


Figure 2.3: ITIL layered viewpoint of IT service management components

of the IT infrastructure is done by activities such as configuration management, change management, network management, etc. In turn, the IT infrastructure management activities need an environmental infrastructure, such as the cable infrastructure, secure power supplies, the office working environment, etc. In addition, some parts of the services might be outsourced, and thus need to be managed as well. The combination of IT service provision, IT infrastructure management, environmental infrastructure management, and externally provided IT services management is called IT service management.

The goal of the IT Infrastructure Library is to offer a systematic approach to the management of IT service provision which provides benefits such as:

- Customer satisfaction with IT services which meet their needs.
- Reduced risk of not being able to meet the business requirements for IT services.
- Reduced costs in developing procedures and practices within an organiza-

tion.

- Better communication and information flow between IT staff and customers.
- Assurance to the IT director that staff are provided with appropriate standards and guidance.
- Greater productivity and best use of skills and experience.
- A quality approach to IT service provision.

In addition, the customer using the IT services should receive benefits such as:

- Reassurance that IT services are provided in accordance with documented procedures, which can be audited.
- Ability to depend upon IT services, enabling the customer to meet business objectives.
- Provision of clearly defined contact points within IT services for enquiries or discussions about changing requirements.
- Knowledge that detailed information is produced to justify charges for IT services and to provide feedback from monitoring of service level agreements.

The library consists of several sets of booklets that contain those 'best practices' in IT service delivery. The booklets are divided into nine sets. The first six sets are called the IT service provision and IT infrastructure management sets. The other three are called the Environmental sets. These latter three sets cover the environmental infrastructure for IT, such as the building, cabling and service facilities. We will only look at the IT service provision and IT infrastructure management sets. The six sets cover the following practices (each practice is described in a separate booklet):

- The Service Support set covers configuration management, problem management, change management, help desk, and software control and distribution.
- The Service Delivery set covers service level management, capacity management, contingency planning, availability management, and cost management for IT services.
- The Managers' set deals with managing facilities management and customer liaison.

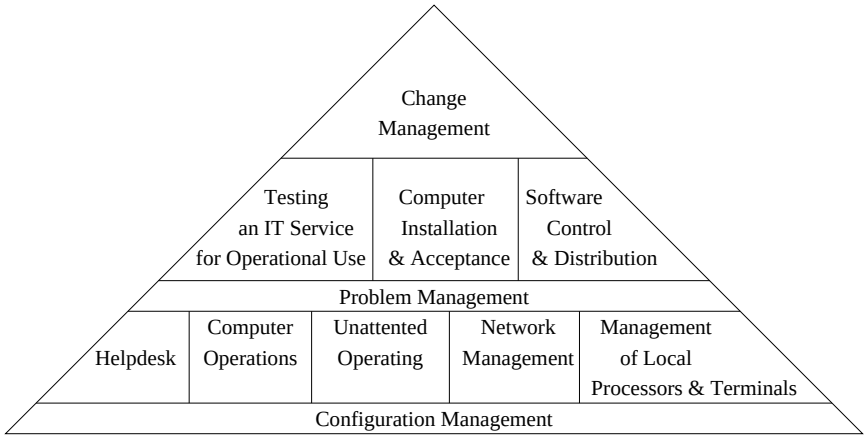


Figure 2.4: Suggested ITIL implementation order (starting from the bottom)

- The Software Support set describes software life-cycle support and testing an IT service for operational use.
- The Computer Operations set covers computer operations management, unattended operating, third party and single source maintenance, and computer installation and acceptance.
- Finally, the Network set describes the management of local processors and terminals.

Each booklet describes the practices in terms of planning; implementation; audits; benefits, cost and possible problems, and tool support. Attention is given to operational procedures, roles, responsibilities, dependencies, support processes, training, etc.

For example, the change management booklet provides directions for operational procedures for change management. The IT service organization should implement procedures for: submitting requests for change (RFC's), logging the RFC's, allocation of priorities, impact and resource assessment, change advisory board meetings, change scheduling, change building, change testing, change implementation, and change review.

Although the booklets cover a wide range of issues regarding IT services, there are still a number of important issues that need more attention. Examples are:

- The specification of service level agreements. Although ITIL does promote the use of SLAs, it does not provide much help on how to develop them.

- The use of service catalogs. ITIL does promote the use of a service catalog (in the service level management booklet) to facilitate the communication with the customers, but again does not say much about the contents or how to develop it.

- ITIL implementation. ITIL itself does not provide much information on the best way to implement the different processes and on how to decide on the best order of implementation, nor is there any support for process assessment. An order for implementation is suggested in Central Computer and Telecommunications Agency (1992*b*), see figure 2.4. Still, it is added that there are no hard and fast rules, and that implementation must be driven by business needs.

- The distinction between service producing processes and service support processes. In our opinion, ITIL does not clearly distinguish between those two types. For example, the ITIL help desk is both used for communication with the end-users (needed for incident handling) and for user support (a service).

While over the years different companies have been selling services that complement ITIL, such as education, training, and consulting on ITIL implementation, ITIL still lacks an overall approach to the improvement of service processes. Improvement is not an integral part of the library.

2.3 Conclusions

In this chapter we have discussed different approaches to process improvement. We have distinguished between approaches that use internal reference points to track improvement against and approaches that use external reference points. Internal reference-based improvement approaches mainly consist of measurement-based methodologies that start from organizational goals that need to be reached. Measurement is used to deliver the necessary information needed to make decisions and change the organization. The best known example of measurement-based improvement methods is the Goal/Question/Metric paradigm.

The second category, external reference-based improvement approaches, consists of frameworks that describe processes and activities needed for high quality software development, software maintenance or IT service delivery. The assumption is that these processes are needed by most, if not all, organizations to be able to deliver high quality software or IT services in a repeatable and controlled manner. Most of these frameworks employ a layered architecture of so-called maturity

levels. Process assessment is used to compare organizational processes with the processes specified by the frameworks. The best known example of maturity-based improvement frameworks is the Software Capability Maturity Model.

If we look at these improvement methods from a software maintenance point of view, we see that the measurement-based approaches are applicable to software maintenance just as well as to software development organizations. These methods generally do not assume anything about the context in which they are applied. In part I of this thesis we report on our experiences with measurement programs in software maintenance environments.

Maturity-based improvement methods on the other hand are domain-dependent by definition. They provide organizations with a set of processes which are specifically needed for a mature organization in the domain the model applies to. The Software CMM assumes that software maintenance and software development are sufficiently similar to be considered one domain. Hence, the Software CMM should be applicable to both maintenance and development. We investigate this issue further in part II of this thesis when we look at the differences and similarities of software development and software maintenance from a service perspective.

Part I

The Measurement Perspective

Part I of this thesis discusses the use of software measurement to improve software maintenance processes. We investigate three research questions in this part:

1. How to introduce measurement in an IT service organization? What are the necessary steps to set up a measurement program and in which order should they be performed?
2. What are the prerequisites that need to be satisfied in order to improve the likelihood of success of the measurement program?
3. What is – or what should be – the relation between measurement and the maturity of the IT service organization?

To investigate this we have done four case studies of measurement programs in four different software maintenance environments. Two of these measurement programs were initiated by the companies themselves; we analyzed both the measurement program and the data that were gathered in retrospect. The other two measurement programs were implemented by two graduate students in two different maintenance organizations. Here we had a fair amount of influence on the introduction and setup of the measurement programs.

The amount of success of these four measurement programs varied widely. We concluded that at least part of the differences in success are due to the maturity of the organization with respect to measurement processes. This led to the formulation of a measurement maturity model that tries to capture those differences in different levels of organizational measurement maturity.

Having developed a first version of this measurement maturity model, we set out to compare our model to existing frameworks in the literature. However, the different nature of the frameworks made a direct comparison difficult, so we decided to construct a generic process model for measurement-based improvement, and use that abstract model as a basis for the comparison. We discovered that the different frameworks agree on the basic processes needed for measurement, disagree on a large number of other aspects of software measurement, and generally pay very little attention to the usage of the results of software measurement. We conclude that the consensus success factors for measurement programs as found in the literature are necessary but not sufficient preconditions for the successful implementation of measurement programs. These, what we call ‘internal’ success factors, need to be complemented with ‘external’ success factors that are aimed at securing that measurement programs generate value for the organization. We propose four external success factors and we suggest a number of activities that can be used to adhere to these external success factors.

Chapters 3, 4 and 5 discuss the four measurement program cases. All four cases are concerned with the measurement of certain characteristics of the maintenance

process in order to help estimating the effort needed to implement changes to the maintained systems. In each chapter, four aspects of the measurement programs are discussed:

1. *Organizational analysis*: why does the organization in question want to use software measurement? What are the goals or problems the organization wants to address using the measurement program? In order to answer these questions the context of the situation needs some attention as well, i.e. the organizational structure, the software process, the relationship with the customer(s), etc.

2. *Measurement implementation*: how has the measurement program been implemented? What measures were selected, what measurement protocols were implemented, how was the measurement program embedded in the software process, etc.

3. *Measurement analysis*: what were the results of the measurement program? Have the measurement goals been reached, how clear are the results, which methods were used to analyze the gathered data, etc.

4. *Organizational implementation*: how have the results of the measurement program been used? What changes has the organization implemented, resulting from the measurement program?

Depending on the characteristics of the case study at hand the emphasis will be on one or two of the questions and the other questions receive less attention.

Chapter 6 describes the measurement maturity model we developed, based on the four measurement program cases. Finally, chapter 7 presents the abstract process model of measurement-based improvement and a comparison of the measurement maturity model with other approaches to measurement program implementation. Four external success factors are proposed and a number of activities is suggested that can be used to adhere to these external success factors.

Chapter 3

Case 1: A Company-wide Measurement Program

This chapter describes a measurement program initiated by a software maintenance organization, which we will call organization A. Organization A is the software maintenance unit of a large Dutch software house. The organization has a large number of customers for which it performs software maintenance, usually both corrective and adaptive maintenance.

One of the problems the organization faces is the fact that customers demand fixed-price contracts for corrective maintenance. These contracts usually have a duration of two or three years. This makes it important to be able to estimate the corrective maintenance workload for new systems in advance. Therefore, organization A started a measurement program targeted at finding relationships between maintenance project characteristics on the one hand and corrective maintenance workload on the other hand.

As described on page 38, the case studies are described in four steps: (1) why did the organization start a measurement program, (2) how was the measurement program designed and implemented, (3) what were the results of the measurements, and (4) how have the results of the measurement program been used. In this case we concentrate on steps 1 and 2 (section 3.1 and 3.2) because the implementation of the measurement program was of such low quality that steps 3 and 4 were not performed.

3.1 Organizational analysis

Organization A maintains a large number of information systems for different customers. The systems maintained differ widely in platforms, programming lan-

guage, size, etc. Some systems have been built by other divisions of the company organization A is part of, other systems have been built by the customer itself, or by third parties. Most of the contracts concern both corrective and perfective maintenance. The corrective maintenance is done for a fixed price per period, adaptive maintenance is billed on a project basis. The maintenance contracts usually have a duration of two or three years. This long duration of the contracts makes it necessary to have some idea of the maintenance workload to be expected, before the contracts are finalized.

3.2 Measurement implementation

The goal of the measurement program, which was started in 1995, was to find relationships between characteristics of the maintenance contract on the one hand, and the effort needed to perform corrective maintenance for that contract on the other hand. The organization decided to gather information about its current contracts. That information was to be analyzed using a multivariate regression tool, developed by the company itself in the course of an ESPRIT project.

The data was gathered using a questionnaire. The questionnaire was filled in by the project leaders. The questionnaire is comprised of two parts: the first part is filled in once for each application and covers the more or less static aspects of the application, such as the size of the application and the technical infrastructure used. The second part of the questionnaire covers the dynamic aspects of the maintenance of the application and is filled in twice per year. It asks for information about the maintenance activities performed, for example the amount of effort spent on corrective maintenance activities and the number of problems fixed.

When we analyzed the questionnaire and the answers filled in on the questionnaires more closely, it turned out that there are a number of problems with both the questions asked and the answers given:

No definitions of used terms Almost none of the terms used in the questions are defined or explained. A few examples:

- One question asks whether a ‘code generator’ is used as part of the programming environment. The fact that one of the questionnaires contained ‘Fortran compiler’ as an answer suggests that without a proper definition of the term code generator different people have a different understanding of that word.
- Another question asks what percentage of the application is ‘data intensive’. What this means is not explained.

- Not only less familiar terms can cause confusion, also more familiar terms like ‘programming language’ need a definition. For example, the questionnaire asks which programming language is used. Unclear is whether batch scripts or shell scripts are to be considered written in a programming language.
- Other examples of terms that need definitions are ‘KLOC’ (refer to Park (1992) why Kilo Lines of Code needs a definition), ‘TP monitor’, and ‘number of used or maintained files’.

Subjective questions A large number of questions can only be answered in a subjective manner. The answer will depend on the judgment of the respondent. A few examples:

- The questionnaire asks whether ‘good test files [are] available?’ What good test files are is left to the respondent to judge.
- Another question asks ‘To what extent constitutes failure of the system a direct threat to the continuity of the business process?’ No further indication of how to judge this is given, nor is it defined what a direct threat constitutes.

Missing data A fair amount of questions is left unanswered because the respondents did not have the required information available. There were three reasons why information was not available:

1. The information requested was not tracked during the project. For example, three questions ask for the number of problems that is solved within two days, between two and five days, and in more than five days, respectively. The answers to these three questions were often estimates, indicated by the often used $\pm$ sign.
2. The information requested was no longer available at the time the forms were filled in. Here, the information would have been available if the forms would have been filled in on time. For example, the questionnaire asks for the increase in size of the application in the previous period. However, if a project leader does not fill in the form immediately at the end of the period, he or she can no longer easily gather that information.
3. Sometimes the information is simply unavailable because it is too difficult or too expensive to gather. For example, one question is concerned with the size of the application measured in function points. This question is often left unanswered.

No measurement protocols In almost all cases the questionnaire gives no guidance on how to measure the attributes asked for. This is important in this particular environment because organization A maintains applications for a large number of different customers, so we cannot assume that things are equal simply because they originate from the same environment. For example, in a development organization which has a coding standard we would be reasonably confident in assuming that code developed by different people is comparable, and hence that we can use a straightforward line count to measure and compare application size (when programmed in the same language). Here, in organization A, we cannot assume this because the code originates from different development organizations. Moreover, the questionnaire gives no indication whatsoever how lines of code should be measured. So, next to the fact that the code could be different in structure and layout across applications, we are in addition not sure how the lines of code were measured.

Also, in a number of cases, it is left undefined what the unit of measurement is:

- The questionnaire requires the percentage of the application that is concerned with batch processing to be given. It remains unspecified whether this is to be measured in a percentage of the lines of code, modules, or programs.
- There is no protocol on how to count function points.
- The percentage of the application that uses a certain hardware platform must be given. Again, how this is to be measured is left unspecified.

The deficiencies of the questionnaire lead us to the conclusion that the data gathered using it are at the least very unreliable.

3.3 Measurement analysis

The dataset gathered by organization A using the questionnaire consists of information about 33 applications. As mentioned in section 3.2, organization A wanted to find relationships between characteristics of maintenance contracts on the one hand, and the effort needed to perform corrective maintenance for those contracts on the other hand.

However, an analysis of the data would be pointless, for two reasons:

1. The dataset is very small, making it hard to find relationships between characteristics of the maintenance contracts and the effort needed for corrective

maintenance. Theoretically, 33 observations is not too small to build a regression model, but it is too small when we want to split the dataset in different parts, e.g. to compare different types of applications, or if we want to investigate multivariate regression models.

2. As described in the previous section, the answers given to the questionnaire are unreliable, making the results of the data analysis unreliable as well. If we do not get any results, we will not know whether that is caused by the unreliable data or by the fact that there are no relationships at all.

We decide not to analyze the dataset.

3.4 Organizational implementation

The measurement program did not result in any changes implemented by the organization.

3.5 Conclusions

We use the success factors as listed by Hall and Fenton (1997), refer to section 2.1.2, to assess the measurement program implemented by organization A. Both from the preceding description of the measurement program in this chapter and the adherence to the success factors as displayed in table 3.1, we conclude that this measurement program is to be considered a failure.

The main failure factor was the lack of a rigorous implementation, as demonstrated by:

- the lack of measurement definitions and protocols,
- the fact that the measurement program was not embedded in the regular work processes of the organization and, in addition,
- organization-wide implementation without organization-wide practices and facilities.

In addition, the measurement program was to tackle a rather ‘big’ problem, which takes quite some data to solve. This measurement program was clearly a bridge too far for organization A.

Success factors	A
Incremental implementation	-
Well-planned metrics framework	-
Use of existing metrics material	-
Involvement of developers during implementation	?
Measurement process transparent to developers	-
Usefulness of metrics data	-
Feedback to developers	-
Ensure that data is seen to have integrity	-
Measurement data is used and seen to be used	-
Commitment from project managers secured	$\surd$
Use automated data collection tools	-
Constantly improving the measurement program	-
Internal metrics champions were used to manage the program	-
Use of external metrics gurus	-
Provision of training for practitioner	-

Table 3.1: Success factors adhered to in organization A

Chapter 4

Case 2: Effort Estimation with Function Points

This chapter describes the second case study. In this case study, we investigated the measurement program implemented by a software support unit of the Dutch Ministry of Transport, Public Works and Water Management, called organization B for short. Organization B supports and maintains one large financial administrative information system, called FAIS, which is used throughout the ministry.

Organization B had problems negotiating the prices of changes with the customer organization. Therefore, it was decided to use function points as a more objective measure of size. In this case, a variant especially tailored to maintenance was used.

Again, the case study is described in four steps: (1) why did the organization start a measurement program, (2) how was the measurement program designed and implemented, (3) what were the results of the measurements, and (4) how have the results of the measurement program been used. In this case the emphasis of the description of the measurement program is on step 3, because we did an a posteriori analysis of the functional size measure employed by organization B.

4.1 Organizational analysis

FAIS is a large financial information system, custom-developed for the Dutch Ministry of Transport, Public Works and Water Management. FAIS is written in a 4GL (Uniface) and uses the Sybase DBMS. The size of the system is approximately 14,000 Function Points. It has been in operational use since the summer of 1994. FAIS is used at 28 locations spread over the country. It has 1200 end users.

The FAIS support organization has three functional units:

- Customer Contacts handles all contacts with the users of the system. This unit develops user specifications for change requests, and takes care of user training.
- Functional Maintenance does the design, implementation and testing of all changes of the system. Corrective maintenance and small changes are handled by one sub-unit (the helpdesk); planned maintenance and new releases are handled by another sub-unit.
- Technical Maintenance takes care of the technical infrastructure and installs new releases.

In this case study we concentrate on planned (i.e. non-corrective) maintenance activities. Incoming change requests are first analyzed by Customer Contacts. This results in one or more change proposals at the level of a FAIS-function. A FAIS-function is a unit of application functionality which can be separately invoked from a menu by the user of the system.

Next, Customer Contacts, representing the customer, and Functional Maintenance have to come to an agreement on the effort needed to implement the change requests. Moreover, an agreement has to be reached on the price the customer has to pay for the change. Reaching the agreement caused troubles for organization B, so it was decided to use a functional size measure as an objective measure of the size of the change. The price of the change would then simply be determined based on the functional size of the change. The planning of the change requests, including effort estimation, was not the target of this measurement program. The functional size measure was solely to be used for change request pricing.

4.2 Measurement implementation

Organization B decided to use function points as a functional size measure. However, traditional function points are not very well suited for measuring the functional size of *changes*. Hence, the organization implemented a function point variant, especially targeted at software maintenance.

In section 4.2.1 we first discuss function points in general. Next, section 4.2.2 discusses the maintenance function points used in this organization. Finally, in section 4.2.3, we describe the measurement process implemented by the organization.

4.2.1 Function points

Function Point Analysis (FPA) is a well-known method to measure the functionality of a system, from the user's point of view. It does so by calculating the number

Function type	Complexity level		
	Low	Average	High
External Input	3	4	6
External Output	4	5	7
External Inquiry	3	4	6
Internal Logical File	7	10	15
External Logical File	5	7	10

Table 4.1: Function types and their weights

of function points of a system in a two-step process:

1. The functional size of a system is calculated by assigning a weight to each *individual* function. The sum of these weights is termed the Unadjusted Function Points (UFP).

2. At the level of the *complete* system, a number of predefined application characteristics, such as processing complexity and transaction rate, result in a Value Adjustment Factor (VAF).

Multiplying UFP and VAF yields AFP: the Adjusted Function Points.

The version of the FPA-model generally used is the one published by Albrecht and Gaffney (1983). Later refinements concern clarification of the counting rules, not the structure of the model.

Albrecht (1979) claims that function points are 'an effective relative measure of function value delivered to our customer'. Various other researchers have also found a strong relation between the number of function points and work effort (see e.g. Kemerer 1987, Banker and Kemerer 1989, Kemerer and Porter 1992, Jeffery and Stathis 1996, Abran and Robillard 1996). FPA is a popular method for development effort prediction, even though various researchers have criticized the underlying methodology (Symons 1988, Fenton and Pfleeger 1997).

To obtain the Unadjusted Function Points of a system, Albrecht distinguishes five function types and three complexity levels for each function type. The five function types are: External Input, External Output, External Inquiry, Internal Logical File and External Interface File. The first three are transaction function types, and the last two are data function types. The UFP associated with an individual function depends on the type of the function and the complexity level associated with that function; see table 4.1. For transaction functions, the complexity level is determined by the number of file types referenced (FTR) and the number of data

File Types Referenced (FTR)	Data Element Types (DET)		
	1-4	5-15	> 15
0-1	Low	Low	Average
2	Low	Average	High
> 2	Average	High	High

Table 4.2: Complexity levels for Input Functions

element types of those files (DET). For data functions, the complexity level is determined by the number of record element types (RET) and the number of data element types of the Internal Logical File or the External Interface File in question. As an example, table 4.2 indicates the complexity levels for Input Functions.

Albrecht's general formula for determining the number of Function Points in a development project already takes into account that such a project generally builds on existing functionality (Albrecht and Gaffney 1983):

$$\text{AFP} = \text{UFP}_{\text{deleted}} \times \text{VAF}_{\text{pre}} + (\text{UFP}_{\text{changed}} + \text{UFP}_{\text{added}}) \times \text{VAF}_{\text{post}} \qquad (4.1)$$

where

- AFP = Adjusted Function Points for the new system,
- $\text{UFP}_{\text{deleted}}$ = Unadjusted Function Points deleted from the system,
- $\text{UFP}_{\text{changed}}$ = Unadjusted Function Points changed in the system, as expected at completion of the project,
- $\text{UFP}_{\text{added}}$ = Unadjusted Function Points added to the system,
- VAF_{pre} = Value Adjustment Factor of the system at the start of the project,
- VAF_{post} = Value Adjustment Factor of the system at project completion.

Though one might be tempted to use this formula for maintenance tasks too, there are at least two reasons for not doing so:

1. removing functionality from a system will most likely be cheaper than developing that same functionality, and
2. adding a small amount, say, n Function Points, to a large function of, say, N Function Points, will most likely be cheaper than adding N Function Points to a function of size n.

4.2.2 From function points to maintenance function points

The Dutch Function Point User Group NEFPUG (nowadays called NESMA) developed a variant of the FPA to remedy the disadvantages of using FPA in a maintenance situation (NEFPUG 1993). In this model, the various Unadjusted Function Point counts are multiplied with a Maintenance Impact Ratio (MIR), indicating the relative impact of a change. Organization B uses a slight variation of this variant. The latter is described below.

Obviously, MIR equals 1 for functions added to the system. For functions deleted from the system, MIR is set at 0.2 in organization B. To determine MIR for changed functions, a scheme similar to that for the complexity levels of functions is used. For example, when a transaction type function is to be changed, that change will affect a subset of the file types referenced and their data element types. If the change involves a small subset of FTR and DET, it is reasonable to assume the change to cost a small fraction of the initial development cost. If the change involves a large fraction of FTR and/or DET, then that change may cost up to the initial development cost. The situation is even worse in the latter case, since such a change not only incurs an effort approximating that for the initial development, but an extra effort to undo (remove) the original function as well.

In organization B, a five point scale is used for MIR-levels for transaction type functions. The resulting scheme for determining MIR is given in table 4.3. The mapping to numerical values is given in table 4.4. Here, %FTR denotes the percentage of file types changed, i.e. %FTR = ($FTR_{changed}$ / FTR_{pre}) $\times$ 100%. The percentage of data element types changed (%DET) is defined in a similar way. Multiplication of UFP with MIR yields the number of Unadjusted Maintenance Function Points (UMFP).

Abran and Maya (1995) found that most adaptive changes are small. To handle these, he proposes a scheme in which the lower complexity levels are further refined. This essentially amounts to applying a Maintenance Impact Ratio, part of the time.

For the situation we are considering, many of the Value Adjustment Factors distinguished by Albrecht and Gaffney (1983) do not apply, simply because we are considering one system only, so that their value is the same for each proposed change. Three of the Value Adjustment Factors were retained, and a new one was added. More importantly, the possible influence of these factors is taken into account at the level of an individual transaction or data function, rather than the complete system. So the complexity of a given function influences the effort to change *that* function; the complexity of other functions, or the system as a whole, does not. The possible effect of the Value Adjustment Factors is simply multiplicative, as for instance in the COCOMO-model (Boehm 1981), and occasionally additive.

%FTR	%DET				
	$\leq 25\%$	$\leq 50\%$	$\leq 75\%$	$\leq 100\%$	$> 100\%$
$\leq 25\%$	Very Small	Small	Average	Large	Very Large
$\leq 50\%$	Small	Average	Large	Very Large	Very Large
$\leq 75\%$	Average	Large	Very Large	Very Large	Very Large
$\leq 100\%$	Large	Very Large	Very Large	Very Large	Very Large
$> 100\%$	Very Large	Very Large	Very Large	Very Large	Very Large

Table 4.3: MIR levels for transaction type functions

MIR-level	Very Small	Small	Average	Large	Very Large
Value	0.125	0.25	0.5	0.75	1.2

Table 4.4: MIR values for transaction type functions

The following Value Adjustment Factors are distinguished:

1. Functional Similarity. This is the reusability factor of Albrecht and Gaffney (1983), though with a quite different semantics. If a change request incurs similar changes in other functions, it is reasonable to assume a copying effect from the first such change to the next ones. In that case, all but the first such change get a multiplicative adjustment of 0.75.

2. Performance Sensitivity. A function is considered performance sensitive if it uses a sufficiently large database table in at least one location. In that case, a multiplicative adjustment of 1.15 is used.

3. Complex Function. Some functions are considered complex. Two levels of complexity are distinguished: complex and very complex. If a complex (very complex) function is added, such incurs a multiplicative adjustment factor of 1.5 (3). If a complex (very complex) function is changed, the additive adjustment is 3 (12).

4. Standard Functionality. Some changes can be easily accommodated with, because they map well onto the primitives of the 4GL being used. In these cases, a multiplicative adjustment factor of 0.5 is used. This factor is not present in Albrecht and Gaffney (1983).

4.2.3 The measurement process

As described in section 4.1, incoming change requests are first analyzed by Customer Contacts. This results in one or more change proposals at the level of a FAIS-function. Customer Contacts and Functional Maintenance independently determine the number of Maintenance Function Points for each change to a FAIS-function. They have to come to an agreement about this number as well as the description of the change, which are then both frozen. The Maintenance Function Points are next used to bill the client organization.

Both the Customer Contacts department and the Functional Maintenance department count the number of maintenance function points for a change request based on detailed counting guidelines. These guidelines give definitions of the concepts used, such as File Types Referenced, Data Element Types, etc., and a procedure on how to count the number of maintenance function points. In addition, the guidelines provide tables of Internal Logical Files and of functions that are considered complex. These tables are aimed at lowering the number of subjective decisions that have to be made during function point counting.

The actual planning of maintenance activities by Functional Maintenance is based on an expert estimate per change to a FAIS-function, and not on the maintenance function point count. This expert estimate is made by Functional Maintenance. We do not know the extent to which the Function Point estimates have influenced the expert estimates. The opposite influence is unlikely, given the detailed counting guidelines that have been used.

4.3 Measurement analysis

In this section we examine the suitability of the used Maintenance Function Point (MFP) model to estimate the effort needed to implement change requests. Because organization B is using the number of maintenance function points to price change requests, we would like to see a fair correlation between the number of function points of a change request and the effort required to implement it. We will compare the quality of the predictions of the MFP-model with the quality of the expert estimates from the dataset. Also, we will decompose the MFP to determine the contribution of model components to the estimate (cf. Abran and Robillard 1996). Finally, we will use analogy-based estimation as another means of assessing the suitability of the dataset for estimation.

There are several criteria to evaluate the predictions of a model (Conte, Dunsmore and Shen 1986). The *coefficient of multiple determination* (R^2) is used to indicate the amount of variance that is accounted for by the independent variables in a linear model. Because R^2 tends to be an optimistic estimate of how well the

One transaction function changed	90
Multiple transaction functions changed	50
Number of changes	140
Total hours spent	6655
Total of expert estimates	6765
Total AMFP	648.88
Hours spent per AMFP	10.26

Table 4.5: Global information on dataset

model fits the population, we also provide the *adjusted* R^2 which compensates for the number of independent variables in the model.

We also use the Mean Magnitude of the Relative Error (MMRE) to indicate the relative amount by which the predictions over- or underestimate the real value. We use PRED(25) to indicate how many of the predictions lie within 25% of the real value. Conte, Dunsmore and Shen (1986) use the simultaneous satisfaction of the two measures

$$\text{MMRE} \leq 25\% \text{ and PRED(25)} \geq 75\% \tag{4.2}$$

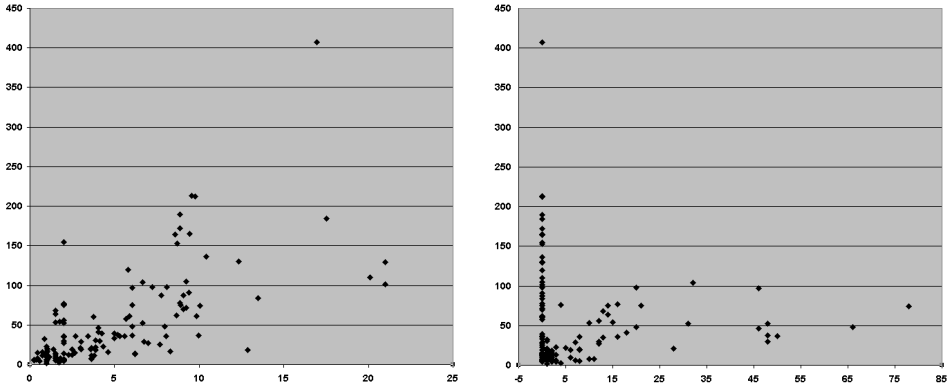
as a criterion for acceptable model performance.

4.3.1 The dataset

Most data is collected at the level of a change to a single FAIS-function: actual time spent to design, implement and test the change, an expert estimate of the change effort, and the number of maintenance function points of the change.

As noted in section 4.2.2, the function points measurement process includes a data function type measurement and a transaction function type measurement. Since FAIS does not interface other systems, the number of External Interface Files is zero. Changes to Internal Logical Files often span more than one FAIS function, or even more than one change request. Such changes may moreover involve data conversions. Though data on this type of change has been collected, they are at a level which is incompatible with the level at which function points are counted. Our analysis is therefore restricted to the transaction function types.

Our initial dataset has 327 entries at the level of a FAIS-function. For 175 entries, actual hours spent have not been recorded at this level. These mostly concern changes which take little time individually, such as the removal of one function. For 12 entries, the data were incomplete. The remaining 140 data points are used in the analysis below; of these, 90 concern a change in one transaction function,



(a) AMFP (x-axis) vs. Work-hours

(b) $\text{DET}_{\text{changed}}$ (x-axis) vs. Work-hours

Figure 4.1: Effort scatter plots

$\text{WE}_{\text{pred}} =$	Std. err.	R^2	R^2 adj.	MMRE	PRED(25)
Expert Estimates	-	0.66	-	57%	39%
$10.26 \times$ AMFP	-	0.46	-	91%	22%

Table 4.6: Comparing the estimates

while 50 concern more than one changed transaction function. See table 4.5 for some global information on this data set and figure 4.1(a) for a scatter plot of effort versus Adjusted Maintenance Function Points (AMFP).

4.3.2 Assessing the maintenance function point estimates

First, we compare the expert estimates with those from the Maintenance Function Point (MFP) model. To predict the Work Effort (WE_{pred}) for change requests using maintenance function points we use the formula $\text{WE}_{\text{pred}} = \text{MFP-ratio} \times \text{AMFP}$. We calculate the MFP-ratio using the total number of adjusted maintenance function points (AMFP) for this dataset and the total number of hours spent working on these 140 change requests, see table 4.5. The MFP-ratio for this dataset is 10.26 hours per AMFP. We compare the estimates of the MFP model with the expert estimates in table 4.6.

It is clear that the expert estimates outperform the MFP model on all quality figures. In the next sections we will explore whether other combinations of model

$WE_{pred} =$	Std. err.	R^2	R^2 adj.	MMRE	PRED(25)
$7.49 \times$ UFP $- 15.06$	44.08	0.40	0.39	110%	26%
$7.98 \times$ UMFP $+ 10.86$	47.27	0.31	0.30	149%	22%
$8.96 \times$ AMFP $+ 6.01$	41.98	0.46	0.45	104%	22%

Table 4.7: Regression models for UFP, UMFP and AMFP

components or other estimation techniques improve the results.

4.3.3 Models based on function point components

As described in section 4.2.2, the main steps taken when computing the number of AMFP for a change request are:

$$\text{UFP} \xrightarrow{\times \text{MIR}} \text{UMFP} \xrightarrow{\times \text{VAF}} \text{AMFP} \tag{4.3}$$

Unadjusted Function Points times the Maintenance Impact Ratio yields the number of Unadjusted Maintenance Function Points. Next, multiplying and/or adding the Value Adjustment Factors results in the Adjusted Maintenance Function Point count. We would expect the R^2 to increase in each of these steps as extra information is added to the maintenance function point count. Table 4.7 displays the regression models for UFP, UMFP and AMFP.

The values of R^2 suggest that inclusion of MIR in the model makes matters even worse. To find a possible reason for this behavior, we will have to consider the more detailed attributes that make up the value of MIR, like $\text{DET}_{\text{changed}}$. We can only do so for changes involving one transaction type function. For changes involving multiple transaction functions, the type of change may differ between the transaction functions that constitute the change. This reduction leaves us with 90 data points, 63 of which concern a change to one function, 19 concern the addition of a function and 8 belong to a rest category (such as the deletion of a function). We perform the regression analysis again on the two subsets for the same variables as in table 4.7, see table 4.8.

Note that we calculated the regression models for new transaction functions twice. Using the 19 new transaction function the regression analysis is quite unsuccessful. If we look at the work-hours for these 19 change requests, we observe that there is one outlier. One change request costs about four times the work-hours of any of the other 18 cases. Though the dataset with new transaction functions is too small to perform further analysis, the figures in table 4.8 do suggest that something is wrong in the estimates for changed functions. Because this set is

Changed transaction functions (n = 63)					
$WE_{pred} =$	Std. err.	R^2	R^2 adj.	MMRE	PRED(25)
$6.16 \times UFP - 9.82$	22.42	0.13	0.12	117%	19%
$3.63 \times UMFP + 16.28$	22.77	0.10	0.09	126%	22%
$3.25 \times AMFP + 16.85$	22.84	0.10	0.08	121%	19%
New transaction functions (n = 19)					
$WE_{pred} =$	Std. err.	R^2	R^2 adj.	MMRE	PRED(25)
$4.15 \times UFP + 4.01$	34.62	0.02	-0.04	130%	21%
$4.15 \times UMFP + 4.01$	34.62	0.02	-0.04	130%	21%
$3.12 \times AMFP + 13.70$	34.44	0.03	-0.03	120%	21%
New transaction functions (n = 18)					
$WE_{pred} =$	Std. err.	R^2	R^2 adj.	MMRE	PRED(25)
$7.78 \times UFP - 19.80$	8.16	0.51	0.48	58%	28%
$7.78 \times UMFP - 19.80$	8.16	0.51	0.48	58%	28%
$5.39 \times AMFP - 0.38$	6.31	0.71	0.69	47%	28%

Table 4.8: Regression models for changed and new transaction functions

sufficiently large (63 cases) we may try to improve the models by investigating alternatives. The fact that the R^2 for the regression model with UFP only is as low as 0.13 is not very surprising: we do not expect the work-hours needed to implement a change in a function to correlate well with the number of (development) function points of that function. We do expect, however, a fair correlation between work-hours and UMFP. The fact that the R^2 for UMFP is even lower at 0.10 suggests a flaw in the mapping from UFP to UMFP.

To further examine this, we try to construct a new MIR that will improve the regression model for UMFP. In the current MFP model, MIR is computed from a table which has as its inputs the percentage of DET changed and the percentage of FTR changed, see tables 4.3 and 4.4. We try to construct a better MIR by performing a stepwise regression on the components that make up MIR: $DET_{changed}$, DET_{pre}, $FTR_{changed}$ and FTR_{pre}. So we are trying to find the model of the form:

$$WE_{pred} = (\beta_0 + \beta_1 \times DET_{changed} + \beta_2 \times DET_{pre} + \beta_3 \times FTR_{changed} + \beta_4 \times FTR_{pre}) \times UFP \quad (4.4)$$

Doing so yields the following model for MIR′:

$$MIR' = 0.19 \times DET_{changed} + 0.02 \times DET_{pre} + 2.03 \quad (4.5)$$

Changed transaction functions (n = 63)					
$\text{WE}_{\text{pred}} =$	Std. err.	R^2	R^2 adj.	MMRE	PRED(25)
$1.23 \times \text{UMFP}' - 3.88$	15.78	0.57	0.56	71%	21%
$1.04 \times \text{AMFP}' + 1.27$	16.81	0.51	0.50	73%	32%

Table 4.9: Regression models with MIR′

Using this new MIR we recompute UMFP and AMFP – resulting in UFMP′ and AMFP′. If we use UMFP′ and AMFP′ on the subset of 63 changed functions we get the results as presented in table 4.9.

We now have a definite improvement in the step UFP $\xrightarrow{\times \text{MIR}}$ UMFP. Comparing table 4.3 with equation 4.5, we observe a striking difference. Whereas the original model assumes that effort is proportional to the relative size of a change, equation 4.5 suggests effort is proportional to the size of the component changed. This observation is in line with results reported in the literature (see e.g. Jørgensen 1995): maintenance effort is highly correlated with size.

The step UMFP $\xrightarrow{\times \text{VAF}}$ AMFP, however, still does not improve R^2. Our attempts to improve VAF by considering its constituents have been in vain. Regression analysis on individual factors and combinations thereof yield quite different parameter values, suggesting that one factor acts as a correction to the other, rather than as an improvement of the model as a whole. This fits in with the observation that these adjustment factors are highly negatively correlated.

4.3.4 Analogy-based estimation

Analogy-based estimation (Shepperd, Schofield and Kitchenham 1996) is an estimation technique in which we make predictions based on a few historical cases from a larger dataset. These historical cases act as analogies for the one for which we are making the prediction. Using the tool Angel (see Shepperd, Schofield and Kitchenham 1996) we can automate the search process for the analogies. Angel determines analogies by computing the Euclidean distance between cases.

To assess the suitability of the dataset for making analogy-based estimations Angel uses jack-knifing; one by one each case is taken from the dataset and an estimate is determined using the remainder of the dataset. The estimates together determine the MMRE and PRED(25) quality figures. This process is repeated for each combination of attributes, yielding the best combination together with the quality figures for that combination.

If we let Angel assess our dataset using the maintenance function point com-

ponents (FTR_{post}, FTR_{pre}, $FTR_{changed}$, DET_{post}, DET_{pre}, $DET_{changed}$, UFP, UMFP, MIR, VAF, AMFP), we obtain the following result:

Variables used for prediction	MMRE	PRED(25)
$FTR_{changed}$, $DET_{changed}$, DET_{pre}, UMFP, VAF, AMFP	39.8%	38%

The values for MMRE and PRED(25) are much better than those for the regression models in tables 4.6 and 4.7. A possible explanation is that the dataset is rather heterogeneous. The scatterplot in figure 4.1(b) illustrates this. Plots for other variables show similar patterns. If the number of attributes is sufficiently large and at the same time discriminates the cases well, analogy-based estimating is a promising alternative to more rigid regression-based models.

4.3.5 Measurement analysis results

In the previous sections we used various FPA variants to predict maintenance effort. We started off with a variant which, like Albrecht's original model, assumes that maintenance effort is strongly determined by the size of a change. The performance of the resulting model proved to be rather poor on the dataset we studied.

Further analysis revealed one particularly weak spot. In organization B the size of a component changed has a much stronger impact on the effort needed than the size of the change itself. The general form of the relation between effort and the size of the component/change is better described as

$$\text{Effort} \approx \text{constant} \times \text{size of the component} \times (1 + \epsilon \times \text{size of change}) \quad (4.6)$$

rather than as

$$\text{Effort} \approx \text{constant} \times \text{size of change} \quad (4.7)$$

On hindsight, this fits in well with other models that use size as the primary factor influencing maintenance effort.

The steps we have taken somehow constitute a calibration of the *structure* of the MFP model, rather than a calibration of its parameters only. Whether the need for such a structural calibration is caused by the particular organization we have been studying, or has more general merit, is still an open question.

Notwithstanding the improvements we have been able to make to the initial model, the results are still rather unsatisfactory. None of the presented models satisfies the criterion for prediction models mentioned in equation 4.2. Whether such is partly caused by the omission of relevant factors, such as the age of a

component or its complexity, is not known. We have simply no other attributes available than those collected for the MFP model. We nonetheless suspect that the heterogeneity of the dataset remains in conflict with the 'one model fits all' flavor of regression-type models. In such cases analogy-based estimation seems to offer an interesting alternative.

4.4 Organizational implementation

Organization B implemented a new procedure to determine the price of the change requests, as discussed in section 4.1. This procedure removed the need for extensive negotiations between the Customer Contacts and the Functional Maintenance department about the price of change requests. From this perspective the measurement program helped solving the problem and can be considered a success.

Our analysis of the data gathered shows that the maintenance function point model used by organization B does contain weak points. However, organization B did not use the results of our analysis to improve the function point model. There were two reasons why the results were not applied. First, despite the weaknesses the organization was content with the model as it was, since it solved their problem of determining the price of changes. Second, a large part of the maintenance activities was to be outsourced in the short term.

4.5 Conclusions

This measurement program can be considered a success, both from a measurement point of view and from an improvement point of view. The implementation of the measurement program was done rigorously, the measurement process was embedded into the maintenance process, etc. This is also illustrated by table 4.10, which compares case B with the success factors for measurement programs as described by Hall and Fenton (1997). The measurement program enabled this organization to use a more objective measure of the size of a change, which eased the negotiations with the customers about the price of a change.

The one thing that organization B failed to do was to check the validity of the used function point model. We have shown in section 4.3 that there is certainly room for improvement of the model.

Success factors	A
Incremental implementation	-
Well-planned metrics framework	√
Use of existing metrics material	-
Involvement of developers during implementation	√
Measurement process transparent to developers	√
Usefulness of metrics data	√
Feedback to developers	√
Ensure that data is seen to have integrity	√
Measurement data is used and seen to be used	√
Commitment from project managers secured	√
Use automated data collection tools	√
Constantly improving the measurement program	-
Internal metrics champions were used to manage the program	-
Use of external metrics gurus	-
Provision of training for practitioner	√

Table 4.10: Success factors adhered to in organization B.

Chapter 5

Cases 3 and 4: Measuring Maintenance Effort

This chapter describes both the third and the fourth measurement program case study. In each of these two case studies, a graduate student implemented a measurement program in a software maintenance organization. Both organizations have trouble estimating the effort needed to implement changes to the software they maintain. The goal of both measurement programs was to determine the main factors that influence the effort needed to implement changes. Both the two measurement programs as well as the maintenance organizations are quite similar, so these two case studies are presented together.

Just like to first two cases, the case studies are described in four steps: (1) why did the organization start a measurement program, (2) how was the measurement program designed and implemented, (3) what were the results of the measurements, and (4) how have the results of the measurement program been used.

5.1 Organizational analysis

Organization C

Organization C is the IT department of a large organization, responsible for carrying out part of the Dutch social security system. As of the beginning of 1996, the organization has been split into a non-profit public body and a private for-profit organization – part of which is the IT department. In the future, the IT department will have to compete with third parties. However, at the time this case study was done (second half of 1996), this is not yet the case and the majority of the customers of the IT department are still departments from the non-profit sibling organization.

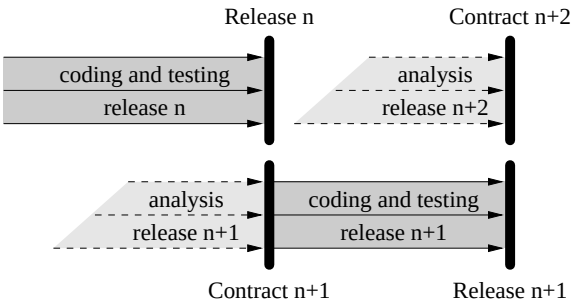


Figure 5.1: Maintenance releases at organization C

The IT department consists of several units, one of which contains the so-called product teams. Each product team develops, maintains and supports multiple systems for one (department of the) customer. Each team is further divided into different groups that each maintain several of the customer's information systems. Our measurement program was introduced into three of the eight product teams.

Each product team consists of about 20 to 30 engineers. Each group in a product team is staffed with between one to five people. Contacts with the customer are handled by the team managers and group leaders. The customer submits change requests that are analyzed by the responsible team manager or group leader. The change requests are then implemented and delivered in the next release, as displayed in figure 5.1. Note that the contract for release $n + 1$ can only be finalized when all of its changes have been analyzed and agreed upon. This point in time roughly coincides with the delivery of release n. Most systems have three or four releases per year, with between one and ten change requests per release.

Officially, change requests undergo five distinct phases before delivery, see figure 5.2. However, during the implementation of the measurement program it turned out that the preparation and functional design are usually done by the same person, the group leader. The same goes for the technical design and building, which are usually done by the same engineer. Therefore, group leaders and engineers often do not know how much time they have spent on each of these phases, which makes it hard to distinguish between them. In our analysis, we will therefore use three phases, indicated by a shaded background in figure 5.2: (1) *analysis*, which consists of preparation and functional design, (2) *coding*, which is the sum of technical design and build, and (3) testing.

In one of the three teams the testing of change requests was not done by the engineers in the team, but was outsourced to a separate test team.

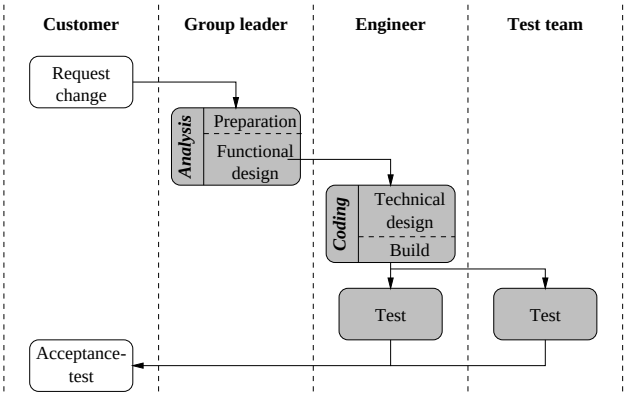


Figure 5.2: Maintenance process organization C

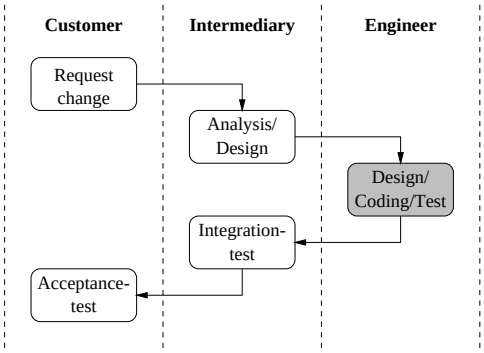


Figure 5.3: Maintenance process organization D

Organization D

Organization D is the IT department of a large Dutch industrial organization. The IT department consists of several units, one of which – the applications unit – is responsible for the development and maintenance of the administrative and process-control systems that the organization uses. The measurements took place at two of the three subunits of the applications unit. Each subunit is staffed with 20 to 30 engineers.

For each of the information systems, there is an intermediary between the client and the maintenance department, see figure 5.3. This intermediary is located at the client site. He or she is responsible for phrasing the change requests. The intermediary is in direct contact with the programmer at the IT department who maintains

the systems of his department. The amount of analysis and design done by the intermediary varies per system: some intermediaries change the functional documentation themselves, others give an informal description of the change and leave it up to the engineer to change the functional documentation. We only collected the effort spent by the engineers, not the intermediaries, as indicated by the shaded box in figure 5.3.

Budgets for maintenance are allocated per system per year. Change requests are implemented, tested and delivered one by one, as opposed to being grouped in releases.

5.2 Measurement implementation

Each measurement program was set up by a (different) graduate student. The goal of both measurement programs was to gain insight into maintenance cost drivers, in order to support the estimation and planning of software maintenance tasks. The measurement programs had a duration of about 7 months.

Both students used the same steps to implement the program:

1. Model the maintenance processes.
2. Determine likely maintenance cost drivers.
3. Develop forms to collect the data.
4. Collect the data by 'walking around'.
5. Use flyers to provide feedback to the engineers.
6. Analyze the data.
7. Present the conclusions to the engineers and management.

During the first step, the students drew up a process model of the maintenance process used in the organizations. They based their model mostly on interviews with managers and engineers. In neither organization a formal standard process existed. The students were able to derive a de facto standard process that was more or less used by all the engineers. However, different groups, even within the same unit or team, would execute different variants of the de facto standard process.

In the second step likely drivers of maintenance costs were determined, based on literature research and interviews with managers and engineers. The upper part of table 5.1 shows the cost drivers mentioned (indicated by a $\surd$) by engineers and management.

Cost drivers mentioned ..... in organization:	C	D
Type of maintenance activity (corrective, adaptive, etc.)	√	√
Changing requirements		√
Work needed to convert data		√
Changed use of the database	√	√
User interface change	√	
Code structuredness, readability, and quality		√
Experience of the engineer with the code		√
Kind of database used		√
Relationship with other applications		√
Relationship with other change requests	√	
Readability, completeness, and structure of the documentation		√
Availability of test sets	√	√
Tests performed	√	
Complexity of the change	√	
Size of the code to be changed		
Size of the change		
Application characteristics		

Table 5.1: Candidate cost drivers

It is surprising that size attributes were not mentioned as possible cost drivers. Because both organizations use source code version control systems, and thus the source code and changes to the code are easily available, it was decided to also collect source code size metrics.

It was decided to not gather application characteristics, such as e.g. programming language, total size of the application, number of users, etc. One reason is that application factors were not mentioned as possible cost drivers in the interviews. Moreover, the number of change requests per application would be relatively small – between one and ten. This makes it difficult to accurately compare the influence of application characteristics on the effort to implement changes.

Using the possible cost drivers, the students developed forms to collect the data. In both organizations, engineers register their hours using an effort registration system. However, in organization C, the effort is not registered per change request, but per release. So, in organization C, the hours planned and spent per change request needed to be registered on the forms. In organization D, this was unnecessary, since the hours per change request were available from the effort registration system. In both organizations, code metrics were gathered after the changes were completed, using the version control systems.

Next, the forms were handed out to the relevant managers and engineers in the organization to be filled in. Knowing exactly who was working on which change request, the students regularly visited these persons to see whether the change requests were ready. This guaranteed the forms would be filled in and collected timely.

During both projects, feedback was given using flyers with information about the status of the project and intermediate results. At the end of the projects both students performed a preliminary data analysis, using statistical tools, and presented those results to the people involved in the measurement programs.

5.3 Measurement analysis

In this section, we present the results of our analysis of the data gathered. Because we have a rather large number of attributes for both organizations, especially when compared to the number of observations, we use principal components analysis (PCA) to determine the main underlying dimensions of the datasets. Using the constructed factors, we build regression models to attempt to explain the effort spent.

The advantage of first applying PCA is that it reduces the large number of attributes into orthogonal factors. The orthogonality of the factors prevents collinearity problems when doing multivariate regression analysis. The disadvantage is that the factors are constructed from the original attributes and cannot be measured directly. So, if we want to design an estimation procedure based on the results of the analysis, we still need to measure the attributes from which the factors are constructed. However, since in these two cases the goal is to find the main factors influencing maintenance effort, and not to immediately develop an estimation procedure, this problem is not insurmountable.

The next section gives an overview of the data gathered. Next, section 5.3.2 describes the results of the principal components analysis. Finally, section 5.3.3 presents the results of the regression analysis.

5.3.1 The datasets

Organization C

The dataset of organization C contains information about 84 change requests concerning 13 different information systems. Of these 13 systems, 12 are written in Cobol, and one system both in Cobol and with Powerbuilder. For 18 change requests, only the first phase (analysis) was performed, i.e. the actual implementation was either canceled or deferred to a later release. For 46 of the 57 change requests

that were both completed and concerned Cobol only, code metrics were gathered from the version control system. For each change request the following attributes were gathered by use of the forms:

Functional Design Does the functional design remain unchanged, is it changed, or is it to be newly designed (functional design).

Database This attribute indicates whether the *structure* of the used database has to be changed due to the change. Since this happened only twice, we did not include this attribute in our analysis.

Complexity The planners were asked to assess the complexity of the change on a five-point-scale, ranging from very simple to very difficult (complexity).

Screens The number of new screens and the number of screens changed due to the change request (screens changed and screens new).

Lists Lists are output of batch programs that provide information to the user about the processing of the batch jobs. Like screens provide the user interface for interactive programs, lists provide the 'user interface' for batch programs.

A list provides information about the number of transactions processed, an overview of the transactions processed, an overview of transactions that failed, an overview of database tables used, etc. We measured the increase of the number of lists (lists new) and the number of lists that were changed (lists changed).

Files Files are all configuration items in the version control system, excluding documentation. Examples are: COBOL program files, COBOL module files, job control files, record definitions and screen definitions. COBOL programs are COBOL files that result in an executable program. Programs implement the main flow-of-control of the system. COBOL modules contain code that is used ('called') by programs or other modules.

Information about the number of files that were to be deleted, changed or added in the course of a change request (files deleted, files new, files changed) was collected. We also separately asked for the number of programs and modules (programs deleted, programs new, programs changed, modules deleted, modules new and modules changed). Note that in general programs changed + modules changed $<$ files changed, because of other types of files that are changed also. Examples are job control files and record definitions.

Using the version control system, we also measured the total size (in lines of code) of the changed files (files loc), and separately of the changed programs

Variable	Mean	Std. dev.	Min.	Max.
effort total	71	56	7	260
effort analysis	14	16	0	76
effort coding	44	45	0	192
effort test	22	23	0	81

Table 5.2: Effort organization C in hours

and changed modules (programs loc and modules loc). We did the same for new files (files new loc, programs new loc and modules new loc). We also measured the size of the change by comparing the old and new versions of files using the UNIX program `diff`. This results in two measures: the number of lines added and the number of lines deleted. Again, these measures were taken for all files together (files loc added and files loc deleted) and separately for programs (programs loc added and programs loc deleted) and modules (modules loc added and modules loc deleted).

Tests The forms also contained two questions about testing, but in the course of the measurement program it turned out that these questions were misunderstood and answered inconsistently. These data were omitted from the dataset.

Table 5.2 shows the mean and standard deviation of the effort spent on the 57 change requests in organization C. There is one outlier in this dataset with respect to total effort. This change request took 472 hours, while the next biggest cost 260 hours and the average effort is 71 hours (excluding the outlier). The large amount of effort for this change request is caused by a large testing effort, while the other characteristics have normal values. After inquiry it turned out that this change request was tested by a new employee. We do not exclude this change request from our dataset; we only ignore it when analyzing testing effort and total effort.

Organization D

In organization D we have collected data on 63 changes that were applied to 9 different systems. Most of these systems are written completely or partially in COBOL. Of the 63 change requests, 5 changes concerned non-COBOL code. These change requests are not used in the analysis. For 52 of the remaining 58 changes, we have collected source code metrics. This dataset also contains one outlier: a change request that took 302 hours to complete. The next biggest change request took 130 hours, the average is 35 hours (excluding the outlier). Since we

Variable	Mean	Std. dev.	Min.	Max.
effort total	35	30	0	130

Table 5.3: Effort organization D in hours

want our analysis to hold for the majority of the data, the outlier is removed from the dataset. This leaves us with 51 datapoints.

Table 5.3 shows the effort spent on change requests in organization D. Next to the total effort data, which was taken from the organization's effort registration system, information was gathered on the hours spent on different phases of the maintenance process – design, coding and testing. However, the hours spent on each phase were to be estimated as a percentage of the total hours per change request. This did not result in correct data, not only because the percentages were estimated, but also because sometimes more than one person would work on a change request. Because the form was filled in by only one of these persons, we only know the effort distribution of that person. Hence, we do not further look at the effort data per phase.

For each change request, the following attributes were measured:

Entities The number of database entities used in the new version that were not used before the change (entities new) and the number of entities whose usage changed, i.e. the usage of one or more attributes changed (entities changed).

Attributes The number of attributes used in the new version of the software that were not used before the change (attributes new), plus the number of attributes no longer used after the change (attributes deleted).

Note that if entities or attributes are changed, i.e. the *usage* changes, this does not imply that the underlying database structure has changed.

Files The total number of changed COBOL files (files), and separately the number of changed modules (modules) and the number of changed COBOL programs (programs) are measured. In addition, the number of new programs or modules (new files) is measured. Note that as opposed to organization C, we only measured COBOL sources at organization D, so programs + modules = files.

Using the version control system, the length of changed files (loc), the number of lines changed (loc added and loc deleted), and the length of the new files (loc new) were measured. Also, for each of these the number of lines changed, added, or new in the procedure division was counted, resulting in loc pd added, loc pd deleted, loc pd, and loc pd new.

Requirements Because programmers were confronted frequently with changing customer requirements during the implementation of change requests, the number of such requirement changes was counted (new requirements).

Data conversion The number of temporary programs needed for conversion purposes (temporary programs).

The following attributes are constructed from attributes that were measured per file, as opposed to the other attributes that were measured per change request. For example, we looked at each source file so see whether goto statements were used. If at least one of the source files that is changed contains goto statements, the variable goto is one, otherwise it is zero.

Code quality The quality of the source code was measured by two ordinal measures: does the source contains goto-statements (goto) and do all programs and modules have a good structure (structure).

Program type The program type was measured to distinguish between interactive programs and batch programs, because engineers consider changes in interactive programs more difficult than changes in batch programs. program type is batch (zero) if all of the changed modules or programs are only used for batch operations, otherwise the program type is interactive (one).

Documentation The quality of the documentation (documentation quality) is set to zero if one or more documents necessary for the change is of low quality, one otherwise.

Experience The experience of the programmer with the code that is to be changed (experience) is one if the programmer has much experience with all files to be changed, zero otherwise.

Difficulty The code to be changed is deemed to be difficult (difficulty is one) if at least one of the files to be changed is considered difficult.

5.3.2 Principal components analysis

Organization C

The first step in principal components analysis is to decide on the appropriate number of factors for the dataset. Using the unrotated factor matrix and a plot of the eigenvalues of the factors (a so-called scree plot) we observe that the common variance in this dataset is best described by three factors. Table 5.4 shows the three factors, after varimax rotation. For readability, all factor loadings between -0.30

Attributes	Factors		
	C1	C2	C3
programs loc added	0.86		
programs changed	0.85		
files loc added	0.82		
programs loc	0.76		
files changed	0.76		0.40
programs loc deleted	0.71		
files loc deleted	0.70		
files loc	0.63		
lists changed	0.59		
lists new	0.31		
files loc new		0.90	
programs loc new		0.84	
programs new		0.83	
files new		0.81	
modules new		0.64	0.33
functional design		0.54	
modules loc new		0.47	
modules loc added			0.87
modules changed			0.82
modules loc deleted			0.79
modules loc			0.57
complexity			0.45
screens changed			0.41
screens new			
Eigenvalue	5.32	4.03	3.46
Percentage of variance	22.2	16.8	14.4
Cumulative percentage	22.2	39.0	53.4

Table 5.4: Rotated factor matrix for dataset C

and 0.30 have been omitted. Together these three constructed variables explain slightly more than 50% of the variance among the attributes used. We see from table 5.4 that the factors can be easily interpreted:

C1. The first factor can be interpreted as the amount of change that affects the flow-of-control. As noted before, programs implement the main flow-of-control. A change in a program file is therefore likely to affect other config-

uration items, such as job control files, as well. For the same reason, such changes can affect the processing information that batch programs return in different lists (lists changed). This factor is thus dominated by flow-of-control effects.

C2. Factor 2 is the increase in the size of the system in terms of new files. The number of new files, new programs and new modules all load on this factor, as well as the size of the new files (files loc new, programs loc new and modules loc new). Also functional design has its highest loading on this factor. This is not surprising, since functional design was coded 0 for no change to the functional design, 1 for a change, and 2 for a new functional design.

C3. The third factor reflects the amount of change in modules.

Two of these factors thus reflect the amount of change to existing code. Factor C1 can be roughly characterized as control flow change, and factor C3 can be labeled algorithm change. Factor C2 denotes the new code, and apparently no distinction as to the type of addition can be made.

Organization D

We also perform principal components analysis on dataset D. Using the unrotated factor matrix we observe that a number of three factors fits this dataset best. Table 5.5 shows the factor loading matrix after varimax rotation. Again, for readability all factor loadings between -0.30 and 0.30 have been left out.

The three factors explain nearly 50% of the variance among the used attributes. Using table 5.5, we see that interpretation of the three factors is again not too difficult:

D1. The first factor can be interpreted as the total size of the code components affected by the change.

D2. The second factor denotes the amount of change.

D3. We see that the third factor can be interpreted as the amount of code added.

Note that this dataset does not reveal a difference in the type of change (control-flow versus algorithmic). On the other hand, it does discriminate between the change itself and the system ‘slice’ affected by the change.

Attributes	Factors		
	D1	D2	D3
files	0.87		
loc pd	0.85		-0.31
loc	0.85		-0.32
modules	0.68		
used-by	0.68		-0.33
programs	0.53		
entities changed	0.43		0.32
relationships	0.36		
goto statements	0.32		
difficulty	0.32		
entities new			
loc pd deleted		0.93	
loc deleted		0.93	
temporary programs		0.72	
loc pd added		0.71	-0.33
loc added		0.67	-0.35
new requirements		0.63	
structure		0.43	
documentation quality			
new files			0.85
loc new			0.83
loc pd new			0.82
experience			-0.64
testing			0.53
program type	0.37		-0.43
attributes new			0.42
attributes deleted			
Eigenvalue	5.70	3.71	3.27
Percentage of variance	21.1	13.8	12.1
Cumulative percentage	21.1	34.9	47.0

Table 5.5: Rotated factor matrix for dataset D

5.3.3 Regression analysis

The next step in our analysis is to investigate whether we can use the factors found in the principal components analysis to explain the effort spent on the change re-

Dependent variable	Independent variables					Adjusted R^2	Std. Err.
	C1	C2	C3	team A	team B		
effort total	0.56	0.41	0.34			0.58	37.1
effort analysis			0.46			0.20	13.6
effort coding	0.60		0.56			0.65	23.8
effort test	0.42			-0.67	-0.80	0.66	13.9
effort test[a]				-0.68	-0.78	0.48	17.1

[a] Without C1

Table 5.6: Stepwise multivariate regression analysis for organization C

quests. We perform multivariate regression analysis, using the factors found in section 5.3.2. We use the beta coefficients of the independent variables to indicate the relative weight of each of the variables in the equation.

Organization C

Input variables for the stepwise regression analysis are the factors found in section 5.3.2, completed with two dummy variables to discriminate between the three teams. The results of the regression analysis is shown in table 5.6. For each dependent variable, the table shows which independent variables entered the regression formula. The numbers shown are the beta coefficients of the variables entered, which allow us to assess the relative importance of each of the independent variables entered in the equation.

If we look at the adjusted R^2's, we see that the equation for the total effort explains 58% of the variance. Using the beta coefficients, we see that C1 is the most influential variable, followed by C2. The formulas for effort coding and effort test both explain about two-third of the variance. The main difference is that for effort test the team dummy variables (team A and team B) play the most important role. The beta coefficients are negative, because the average testing effort of these two teams is considerably lower than for the third team. This is explained by the fact that the third team outsources the testing to a separate test team, while the other two teams test the change requests themselves, see figure 5.2. If we omit factor C1 from the analysis, we see that the explained variance from the two dummy variables alone is almost 50%. Adding C1 increases the explained variance with about 20 percentage points. So, although the explained variance for testing is as high as for coding, its source is rather different.

The explained variance for effort analysis is much lower than for the other

Dependent variable	Independent vars.			Adjusted R^2	Std. Err.
	D1	D2	D3		
effort total	0.33			0.09	28.5

Table 5.7: Stepwise multivariate regression analysis for organization D

equations. It seems that the factors that we have constructed have little influence on the effort needed to prepare the change request. This does suggest that it is important to look at the maintenance *process* carefully and select different metrics for each of the process steps.

Organization D

Using the factors found in section 5.3.2, we again perform stepwise regression analysis. The regression analysis does result in a formula, but the explained variance is not nearly as high as for the dataset of organization C. Only one variable enters the equation: D2 – the amount of code changed.

It is difficult to investigate this further, since we don't know how much of the effort in organization D was spent on design, coding and testing. As mentioned in section 5.3.1, the way in which we attempted to collect information about maintenance subtasks failed to deliver reliable data. Hence, we can only hypothesize on the reasons why the information gathered in this measurement program explains so little about the maintenance effort, as opposed to the results for organization C. We think there are two possible reasons for this difference:

- The maintenance process at organization D is quite dependent on the specific maintenance programmer and his relationship with the customer. The amount of analysis done by the programmer largely depends on how much analysis the intermediary at the customer site has done.
- The effort data were taken from the effort administration system. We do not know – and had no control over – the accuracy with which these data were registered.

5.4 Organizational implementation

Due to different factors neither organization used the results to change the planning process for change requests. In organization C, an attempt was made at continuing the program. However, because the organization did not make anyone specifically

Success factors	C	D
Incremental implementation	-	-
Well-planned metrics framework	√	√
Use of existing metrics material	√	√
Involvement of developers during implementation	√	√
Measurement process transparent to developers	√	√
Usefulness of metrics data	√	√
Feedback to developers	√	√
Ensure that data is seen to have integrity	√	√
Measurement data is used and seen to be used	-	-
Commitment from project managers secured	√	√
Use automated data collection tools	-	-
Constantly improving the measurement program	-	-
Internal metrics champions were used to manage the program	√	√
Use of external metrics gurus	-	-
Provision of training for practitioner	-	-

Table 5.8: Success factors adhered to in cases C and D.

responsible for collecting the metrics, the forms were not filled in anymore and the measurement program came to a halt. In organization D, the manager who had initiated the program became ill, and management support for the program stopped.

In addition, neither organization used the data to implement or improve the planning procedures. In organization C it was felt that more information was needed to build a sound planning procedure. Organization D did not use the data, because, as we have seen in section 5.3, the data does not explain the effort expended.

5.5 Conclusions

From a measurement program implementation point of view both programs were rather successful. Table 5.8 shows how these two measurement programs score when compared to the success factors mentioned by Hall and Fenton (1997). However, in both organizations the programs collapsed after the students had left their position as metrics guru. So from an improvement point of view, the measurement programs both failed. They did not have any impact in terms of improvement to the organizations involved.

Chapter 6

Measurement Maturity

In the previous three chapters we have reported on the four measurement program case studies done. The success of these measurement programs varied widely. The measurement program described in the first case was quite unsuccessful, the measurement program in case two was rather successful, and the measurement programs implemented in cases three and four were successful as long as the students were supporting the program. In this chapter we reassess the cases from a measurement program implementation point of view, i.e. we try to find factors that might explain the success or failure of the measurement programs. From this assessment we conclude that at least part of the success seems to be explained by the capability of the organization with respect to software measurement. We propose a measurement maturity model that captures the differences in capability in different maturity levels.

Section 6.1 presents the reassessment of the measurement program case studies. Next, section 6.2 elaborates on the notion of measurement capability. Section 6.3 presents our measurement capability maturity model. Finally, the last section presents the conclusions of this chapter.

6.1 Case study overview

In this section, we reassess the measurement program cases presented in chapter 3, 4, and 5. We give a short overview of each of the organizations and its measurement program.

A. This organization is a business unit of a major Dutch software house. It maintains and operates information systems for several small and large customers. In 1995, a questionnaire was developed to gather project information in order

to support the bidding process for new projects and new Service Level Agreements.

Estimates were to be derived using an experimental tool developed in the course of an ESPRIT project. There was very little knowledge of the tool and the underlying statistics within the business unit. The unit had undergone several reorganizations in the last few years, and was still very much in a state of flux. The accuracy of the data gathered was, at best, mixed. Quite a few entries were left blank. Quite a few also suspiciously looked like guesstimates. The questions posed sometimes asked for a subjective answer, and were sometimes ambiguous.

B. Organization B maintains and supports a large financial administrative information system for one of the ministries of the Dutch government. Here, maintenance function points – a variant of the standard Albrecht function points – are used to support negotiations between the users and the maintainers about changes to the system.

   This seemed like a measurement-wise organization. There were detailed guidelines as to how and what to measure. The measurement process was strongly supported by management. Everybody knew what the measurements were used for. There were clearly visible pie charts illustrating various performance indicators on the door of the manager's office.

C. Organization C is the IT department of a large organization, responsible for carrying out part of the Dutch social security system. The measurements were done at three of the so-called product teams, which are teams of about 25 engineers, each team being responsible for the maintenance of a number of information systems. The goal of the measurement program was twofold: to gain insight into the cost drivers for change requests and to gain practical experience with the introduction of a measurement program.

   This is also an organization in flux. It is part of a large organization that has been split up and gone commercial. The people still have to get accustomed to their new role: from being an internal maintenance department to being a commercial service provider. The setting up of a measurement program as well as collecting data and analyzing them were done by an MSc student as part of his graduation project. Participants were willing to help, but their attention easily slipped. Management's goals were primarily aimed at establishing a sound organization, and taking measurements was supported insofar this helped to reach the primary goals.

D. The final organization is the IT department of a large Dutch industrial orga-

nization. The measurements took place at two departments, responsible for the maintenance of several administrative systems and process-control systems. The goal of this measurement program was identical to the goal of organization C. As in organization C, the setting up of the measurement program as well as the collection and analysis of data was done by a (different) MSc student as part of a graduation project.

This organization is a stable organization. Its primary process concerns the production of steel. For each information system, there is an intermediary between the client and the maintenance department. This intermediary is located at the client site. He is responsible for the phrasing of change requests. He is in direct contact with the programmer(s) in the maintenance department. The amount of analysis and design done at the client side varies per system. Budgets are allocated per system per year. There is some pressure from the client side to make maintenance costs more 'visible'. The measurement program was started because of this pressure.

Success factors	A	B	C	D
Incremental implementation	-	-	-	-
Well-planned metrics framework	-	√	√	√
Use of existing metrics material	-	-	√	√
Involvement of developers during implementation	?	√	√	√
Measurement process transparent to developers	-	√	√	√
Usefulness of metrics data	-	√	√	√
Feedback to developers	-	√	√	√
Ensure that data is seen to have integrity	-	√	√	√
Measurement data is used and seen to be used	-	√	-	-
Commitment from project managers secured	√	√	√	√
Use automated data collection tools	-	√	-	-
Constantly improving the measurement program	-	-	-	-
Internal metrics champions used to manage the program	-	-	√	√
Use of external metrics gurus	-	-	-	-
Provision of training for practitioner	-	√	-	-

Table 6.1: Success factors adhered to in cases A, B, C and D.

Table 6.1 shows to what extent the organizations adhered the Hall and Fenton success factors. We see a good correlation between the success of the measurement program and the number of success factors adhered to.

From the case studies we have learned the following lessons:

- A sound and rigorous definition and implementation of measures, measurement protocols and measurement processes is an essential prerequisite to success.

- The previous issue needs to be mastered throughout the organization before an organization-wide measurement program can be implemented. In other words, mastering basic measurement activities is a precondition for organization-wide implementation of a measurement program.

- Measurement processes that are integrated in the software (maintenance) process need less attention to be performed adequately than those which are not integrated. In organization B, the counting of the measurement function points was an integral part of the change process, so these activities were done automatically. In organizations C and D the measuring of change characteristics was not part of the normal software process. Practitioners needed to be encouraged regularly to fill in the forms.

- Organization B applied a metric – maintenance function points – without proper validation of the underlying model. Proper analysis of the metrics is needed to validate the measures.

6.2 Measurement capability

We hypothesize that part of the differences in measurement program success can be explained by the differences in measurement capability. We see that organization A tried to establish an organization-wide measurement program. However, the organization had not mastered the basic practices needed for defining measures and measurement protocols. There was no tool support for the measurement process, nor was the measurement process integrated in the software maintenance process. Organization B on the other hand, did a sound implementation of the measurement program: the measures were defined rigorously, the measurement process was integrated into the maintenance process, and the scope of the measurement program was limited and clear. In organizations C and D, the measurement programs ware implemented by two students. The students functioned as the internal measurement guru's and kept the program running by stimulating the practitioners to fill in the measurement forms and providing feedback. We see that in both organization C and D the measurement program stopped as soon as the students left their positions as measurement guru.

This assessment suggests that we can differentiate the amount of measurement capability a software organization has. We define *measurement capability* as 'the

extent to which an organization is able to take relevant measures of its products, processes and resources in a cost effective way resulting in information needed to reach its business goals.'

In chapter 2 we have discussed several guidelines for implementing measurement programs. First, we discussed goal-based measurement approaches in section 2.1.1. These approaches provide guidelines for the derivation of measures from measurement goals, and steps to be taken to package and reuse measurement experience. However, these methods focus mostly on one aspect of software measurement, namely the derivation of measures from measurement goals. Less attention is paid to aspects such as data analysis, measurement protocols, etc.

Second, in section 2.1.2 success factors for measurement programs were discussed. Several authors have identified success factors (e.g. Hall and Fenton 1997, Jeffery and Berry 1993, Rifkin and Cox 1991). After studying other research on measurement, Hall and Fenton (1997) identified a consensus on requirements for measurement program success. We have shown to what extent these success factors were adhered to in our four case studies in table 6.1. However, these success factors do not provide organizations with a clear cut path on how to introduce measurement into their organization, i.e. which steps need to be taken first and which processes need to be in place.

Third, we have mentioned work on measurement maturity models in section 2.1.3. This will be further discussed in chapter 7. For now it suffices to note that the software measurement technology maturity framework by Daskalantonakis, Yacobellis and Basili (1990-1991) lacks a clear focus on measurement processes, and that the work by Comer and Chard (1993) is limited to one maturity level only.

With respect to software process improvement, different approaches exist, most notably the Software Capability Maturity Model (SEI 1995, Paulk, Curtis, Chrissis and Weber 1993, Paulk, Weber, Garcia, Chrissis and Bush 1993). Other improvement paradigms for software are largely based on the Software CMM, including the Bootstrap approach (Haase, Messnarz, Koch, Kugler and Decrinis 1994, Kuvaja, Similä, Krzanik, Bicego, Koch and Saukkonen 1994) and the ISO 15504 standard for software process assessment and improvement (SPICE) (El Emam, Drouin and Melo 1998). Each of these improvement models includes measurement as a means to help improving the software process. However, these methods do not prescribe how the measurement processes themselves should be implemented. For example, the Software CMM does prescribe that in all key process areas measurements should be taken to determine the status of the activities. But only on level 4, measurement is explicitly dealt with by the key process area Quantitative Process Management. Since the Software CMM is concerned with the software process, measurement is only covered insofar it directly deals with improving the software process. The issue of introducing and improving measurement processes is beyond

Level	Measures
5. Optimizing: improvement fed back to process	process and feedback for changing process
4. Managed: measured process (quantitative)	process and feedback for control
3. Defined: process defined, institutionalized	product
2. Repeatable: process dependent on individual	project
1. Initial: ad hoc	baseline

Table 6.2: Process maturity related to measures (Pfleeger and McGowan 1990)

the scope of the Software CMM proper.

On the subject of the relation between measurement and software process improvement, Pfleeger and McGowan (1990) recommend the collection of different measures depending on the organization's maturity level. See table 6.2. Pfleeger (1995) presents a combinational approach to measurement programs, using the Goal/Question/Metric paradigm to derive goals to be met and questions to be answered, and the Software CMM to decide what can be measured – i.e. what is visible. As the process matures, visibility increases and a more comprehensive set of metrics can be measured.

None of these sources gives a structured path to enhance *measurement capability*. The success factors for software measurements, though highly useful, do not differ all that much from the hit list for software reuse, formal specifications, or any major organizational change relating to the software process. They give premises for success, not roads to get there. In a similar way, Pfleeger's work (1990, 1995) gives insight into which measures can be collected at which maturity level. It does not help us to improve the measurement process itself. The Measurement CMM is intended to fill that gap.

6.3 The Measurement Capability Maturity Model

In this section we describe the proposed Measurement Capability Maturity Model. First, the objectives of the Measurement CMM are laid out. Next, the maturity levels of the Measurement CMM are described. Section 6.3.3 touches on the key process areas and finally section 6.3.4 describes the relation between the Measurement CMM and other maturity models.

6.3.1 Primary objectives of the Measurement CMM

The goal of the Measurement CMM is twofold:

1. to enable organizations to assess their capabilities with respect to both software and software process measurement, and,

2. to provide organizations with directions and steps for further improvement of their measurement capability.

The Measurement CMM does this by measuring the measurement capability maturity on a five level ordinal scale and by prescribing processes that have to be in place in order for an organization to reside on that level. This is roughly the same framework as used in the Software CMM, or the People CMM (Curtis, Hefley and Miller 1995*a*, Curtis, Hefley and Miller 1995*b*).

We define *measurement capability* as 'the extent to which an organization is able to take relevant measures of its products, processes and resources in a cost effective way resulting in information needed to reach its business goals.'

An organization that scores high on the Measurement CMM scale should be able to:

- gather relevant information about its own performance with respect to its long and short term business goals;

- continue to collect the relevant information when either the organization itself or its environment changes;

- do so in a cost effective way by reducing the number of collected measures or by using automated measure collection when possible;

- provide an environment in which both management and staff are convinced of the usefulness of measurement and, moreover, are continuously being convinced by the measures themselves.

Note that the business goals themselves are not part of the model, they are input to the model. Measurement goals are derived from the business goals. Organizations with a higher measurement capability should be better able to measure the right measures in order to help reach their business goals.

Also note that measurement capability addresses the ability of organizations to measure processes, products and resources *as is*. Improvement of the software process or products is not part of the Measurement CMM, though higher visibility (i.e. maturity) offers more opportunities to measure (see Pfleeger and McGowan 1990). For example, let us suppose that an organization wants to know how much time

it spends on testing, but the organization does not follow a defined development cycle in which it is clear when the testing phase starts and when it ends. In such a case the organization cannot expect to take valid measurements of time spent on testing without clearly specifying what is meant by testing and without making sure everyone is working according to that specification. Similarly, implementing a configuration management system to ensure that software components are uniquely identifiable is not part of the Measurement CMM. While these software process improvements do improve visibility of the software process, they do not improve *measurement capability.* Moreover, they are already part of software process improvement methods, such as the Software CMM.

6.3.2 The maturity levels of the Measurement CMM

The maturity levels for the Measurement CMM are defined similarly to those of the other capability maturity models. This means that on level 1 – initial – there are no key process areas defined. In essence, level 1 is the level on which all organizations reside that have no key process areas implemented. On level 2 – the repeatable level – organizations have basic measurement processes in place, which means they are able to collect measures during projects. Measures are probably not comparable across projects, since each project potentially has its own measurement goals and defines its own measures. On level 3 – the defined level – this problem is solved, because the organization standardizes its measurement process and determines a basic set of measures that each project has to collect. Also, an organization wide measurement database is created, which contains all historic project data. Level 4 is the managed level, meaning that the organization will be able to assess the costs of different measures. Technology is being used to make the measurement process more efficient. Finally, at level 5 – the optimizing level – the organization is ensuring that measurement processes are not only efficient, but also effective. Measures are regularly judged on their merits and measurement processes are adjusted when necessary to reflect changes in the measurement environment.

More formally, we define the Measurement CMM maturity levels as follows:

1. **Initial**: The organization has no defined measurement processes, few measures are gathered, measurement that takes place is solely the result of actions of individuals.

2. **Repeatable**: Basic measurement processes are in place to establish measurement goals, specify measures and measurement protocols, collect and analyze the measures and provide feedback to software engineers and management. The necessary measurement discipline is present to consistently obtain measures.

3. **Defined**: The measurement process is documented, standardized, and integrated in the standard software process of the organization. All projects use a tailored version of the organization's standard measurement process.

4. **Managed**: The measurement process is quantitatively understood. The costs in terms of effort and money are known. Measurement processes are efficient.

5. **Optimizing**: Measurements are constantly monitored with respect to their effectiveness and changed where necessary. Measurement goals are set in anticipation of changes in the organization or the environment of the organization.

6.3.3 The key process areas of the Measurement CMM

For an organization to reach a certain level other than the first level, certain processes need to be in place. These processes are grouped in key process areas, where *key* merely means that there could be more – non-key – processes, but that those non-key processes do not need to be in place to reach a certain maturity level. An organization can only reach a certain maturity level when it has implemented all key process areas for that level.

Below we present the key process areas for the Measurement CMM. Note that each of these key process areas should be described more thoroughly, in terms of goals and common features (common features define the activities performed and the activities needed to institutionalize the process (see SEI 1995)). So far, we have only specified the purpose of each key process area:

1. Initial: no key process areas.

2. Repeatable:

   (a) Measurement Design: Measurement goals, measures and measurement protocols are established according to a documented procedure, and goals, measures and protocols are kept consistent with each other. Measurement protocols are managed and controlled.

   (b) Measure Collection: Measures are collected according to the measurement protocol.

   (c) Measure Analysis: The collected measures are analyzed with respect to the measurement goals.

   (d) Measurement Feedback: The measurement goals, the measurement protocols, the collected measures and the results of the analysis are made available to the people involved in the measurement process.

3. Defined:

   (a) Organization Measurement Focus: Software measurement activities are coordinated across the organization. Strengths and weaknesses of the measurement process are identified and related to the standard measurement process.

   (b) Organization Measurement Design: A standard measurement process for the organization is developed and maintained and information with respect to the use of the standard measurement process is collected, reviewed and made available.

   (c) Organization Measure Database: Collected measures are stored in an organization-wide database and made available.

   (d) Training Program: People are provided with the skills and knowledge needed to perform their roles.

4. Managed:

   (a) Measurement Cost Management: Costs of measurement are known and used to guide the Measurement Design Process and the Organization Measurement Design process.

   (b) Technology Selection: The information of measurement costs is used to choose and evaluate technology support for the measurement process.

5. Optimizing:

   (a) Measurement Change Management: The measurement capability is constantly being improved by monitoring the measurement processes and by anticipating changes in the software process or its environment.

The Measurement CMM maturity levels together with the key process areas provide organizations with both a measurement scale along which they can assess their measurement capability, and directions for future improvements.

6.3.4 Relationship with other capability maturity models

As mentioned in section 6.3.1, the Measurement CMM does not prescribe the improvement of processes other than measurement processes. The improvement of software processes is covered by the Software CMM. The two models are linked by the processes, products and resources that are subject to measurement on the

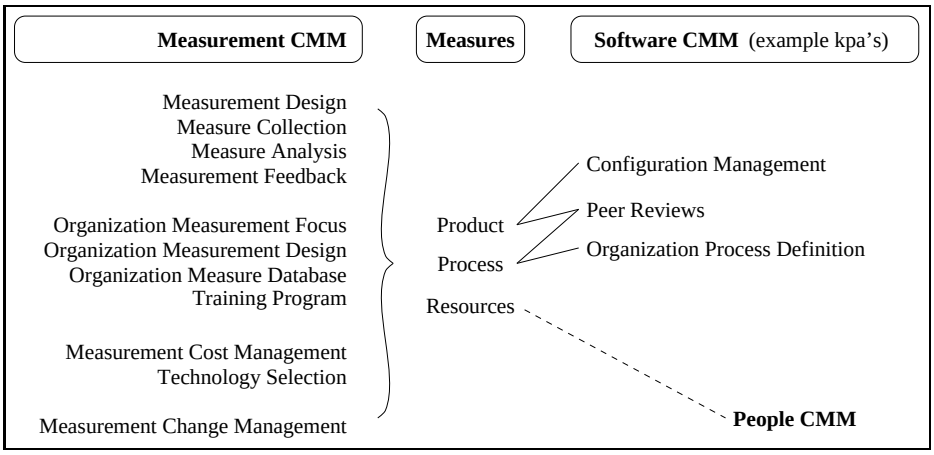


Figure 6.1: The Measurement CMM related to other CMMs

one hand, and are part of the software process – and thus covered by the Software CMM – on the other hand. The same goes for the relationship between the Measurement CMM and the People CMM. We can visualize this as in figure 6.1.

We can see that an organization that has reached level 2 of the Software CMM is able to take detailed measures about software components that are under configuration management. On the other hand, if the organization does not have a standard software process, it will be difficult – if not impossible – to measure the duration of software activities, since they have not been standardized (see Pfleeger and McGowan 1990, Pfleeger 1995).

We observe that an organization that wants to increase the knowledge about and insights into its own software processes, needs to advance on both the Software CMM and the Measurement CMM ladder.

6.4 Conclusions

If we apply the Measurement Capability Maturity Model to the environments discussed in section 6.1, we observe the following:

- Organization A is at level 1. None of the key process areas of level 2 has been fully implemented. The attempt to build an organization-wide project database clearly was a bridge too far. First improvements should concentrate on improving the measurement design and collection process areas.

- Organization B is at level 2. Since the organization is concerned with one

project only, one is tempted to conclude that it is at level 3 as well. However, current measurements are used for one goal only, viz. estimate the size of changes. When we tried to relate these size data to other resource and product characteristics, we encountered considerable difficulties. Clearly, process areas such as Organization Measurement Focus and Organization Measure Database are not fully developed yet.

- Both organization C and organization D are at level 1. The MSc projects concerned all of the process areas of level 2. Clearly, none of these processes are firmly embedded within the organizations yet. For that reason, the measurement program is fragile.

The Measurement Capability Maturity Model provides us with an instrument to assess the various environments in which we implemented a measurement program. It allows us to assign a measurement score to each of the organizations, and explains the success or failure of our measurement efforts. It also identifies areas in which improvements can be sought.

In order to validate the Measurement CMM we would need to show that the model is part of a valid prediction system in which organizations that score higher on the Measurement CMM scale, are better able at developing and maintaining software than organizations that score lower on the Measurement CMM scale. However, such a validation is difficult to perform for several reasons:

- The model would have to be specified into more detail to contain key practices and a related questionnaire to facilitate assessments.

- A full validation, in which we show that organizations implementing the Measurement CMM indeed perform better than those that do not, would probably take several years. We do not have that much time available.

These reasons made it impossible to validate the Measurement CMM in the course of the Concrete Kit and Kwintes research projects. However, as a first step, we compare the Measurement CMM with other literature on measurement programs. This comparison is described in the next chapter.

Chapter 7

Measurement-based Improvement

In the previous chapter, we have developed a preliminary Measurement Capability Maturity Model. In order to validate this model, we have to demonstrate that our model is part of some prediction system in which organizations that score high on the Measurement CMM scale, have a higher probability of achieving successful projects, making a higher profit, etc. However, issues such as the time and money needed made it impracticable to perform such an evaluation within the two research projects covered by this thesis.

In this chapter, we take some first steps to validate the Measurement CMM by comparing it to other literature on implementing and managing measurement program. To facilitate such a comparison of the literature on measurement programs, we have developed an abstract process model of measurement and improvement processes. In section 7.1 we present this Measurement-based Improvement model. In section 7.2 we show how this model can be used to analyze the measurement program set up by an Ericsson department as part of its efforts to reach level four of the Software CMM. Next, in section 7.3 we compare the literature with the Measurement CMM, using the Measurement-based Improvement model. We conclude that the emphasis of most of the literature is on getting the measurement process right, and that there is little attention for the actual usage of the measurement results to generate value for the organization. Section 7.4 proposes a number of 'external' success factors for measurement programs, in addition to the well-known 'internal' success factors. We also show how these success factors could be adhered to by tightly coupling the activities that precede the implementation of the measurement program with the activities that should be performed as a result of the measurement program.

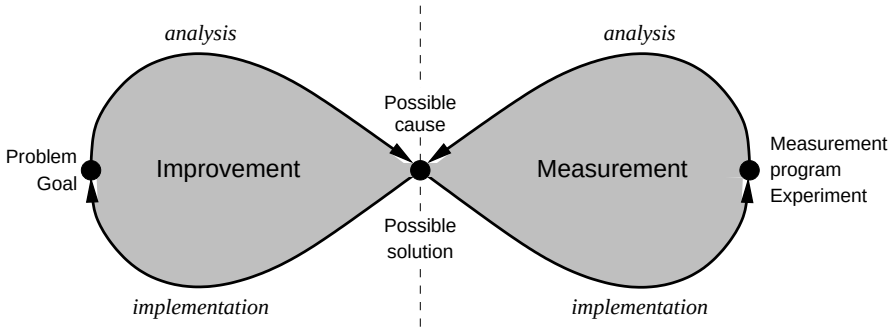


Figure 7.1: A generic process model for measurement-based improvement

7.1 The measurement-based improvement model

Figure 7.1 displays a generic process model for measurement-based improvement. It more or less resembles a 'pretzel', a loaf of bread in the form of a loose knot[1]. The pretzel consists of two parts—the two halves, three concepts—the black dots, and four steps—the four arrows.

The cycle starts with an organizational problem or goal (left black dot). We do not assume anything about the 'size' of the problem or goal. A problem could only affect one developer or the whole organization, in both cases the same steps have to be passed through. The organization analyses the problem (upper left arrow), and arrives at one or more possible causes of the problem and/or possible solutions (middle dot). The analysis will generally be based on a combination of knowledge about the own organization, knowledge from literature ('theory'), and common sense. Next, the organization has to decide whether it has sufficient knowledge to establish the cause of the problem and correct it, or to reach the stated goal. If this is the case, the organization need not traverse the right cycle. In most cases, however, the organization needs to find out which of the possible causes is the real cause of the problem, or which of the possible solutions is the best solution. Or, it may need extra information to implement the solution. To gather this information, the organization can design an experiment or set up a measurement program (lower right arrow). Executing the measurement program or experiment (right dot) results in the gathering of data, which is analyzed and related to the problem or solution at hand (upper right arrow). Finally, the organization solves the problem or reaches the goal by implementing the solutions found (lower left arrow).

[1] Of course, from a mathematical point of view, the figure looks like a lemniscate of Bernoulli.

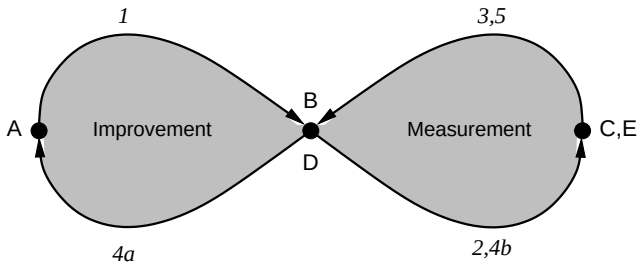


Figure 7.2: An example of measurement-based improvement

Although both the preceding description and the arrows in figure 7.1 suggest a chronological sequence of steps, this is not necessarily the case. The arrows merely indicate causal relations. Hence, the model does not prescribe a single loop through the lemniscate. It is very well possible for an organization to iterate the right loop a number of times before implementing a solution. For example, it may be necessary to first implement an experiment to find the cause of a problem, and then implement another experiment to find a suitable solution. Moreover, organizations might also want to implement a solution and a measurement program in parallel, to monitor the implementation of the solution.

Let us illustrate the model by means of an example, see figure 7.2. Suppose a software maintenance organization has problems planning the implementation of change requests. Often, the implementation of specific change requests takes much more time than planned, and the organization fails to deliver the changed software in time. So, the problem this organization faces is the inaccurate planning of change requests (A). After analyzing the problem (*1*), the organization discovers that it does not know which factors influence the time needed to implement change requests (B). The organization decides to investigate this, and designs (*2*) a short-running measurement program (C) to investigate possible factors. After running this measurement program for a limited period of time, the gathered data are analyzed (*3*). We assume that a number of factors are found that influence the effort needed to implement change requests (D). Next, a planning procedure is developed and implemented (*4a*) in which the factors found are used to plan the change requests. An accompanying measurement program (E) is designed (*4b*) to gather the data needed for the new planning procedure and to monitor the accuracy of the planning (*5*).

We conclude this section with a few remarks on the nature of the presented generic process model of measurement-based improvement.

First, one could wonder whether this model is prescriptive or descriptive. We assume that if software organizations want to improve their processes or products, and use measurement to support those improvements, they will perform the activities as we have described above. That means we use the model as a representation – though very abstract – of what goes on in reality; i.e. it is a descriptive model. One could argue that the model is also a prescriptive model; it tells us which activities to perform when conducting measurement-based improvement. However, because of the high level of abstraction, the model is unsuitable to directly support organizations in their measurement-based improvement efforts.

Second, the model resembles the Goal-Question-Metric paradigm (Basili and Rombach 1988). One could be tempted to map the GQM goal on the left black dot, GQM questions on the middle dot, and the GQM metrics on the right dot. However, the goal of the GQM-paradigm and the goal of the process model are not the same: the goal in the pretzel is an organizational goal, whereas the goal in the GQM-paradigm is a measurement goal. Still, GQM can very well be used to support the design of the measurement program (lower right arrow). Adaptations of GQM (such as described in Latum, van Solingen, Oivo, Hoisl, Rombach and Ruhe 1998, Park, Goethert and Florac 1996) focus on the right side of the pretzel as well.

Third, the distinction made in the model between improvement on the one hand, and measurement on the other hand, corresponds with the distinction made by Kitchenham, Pfleeger and Fenton (1995) between the empirical, real world and the formal, mathematical world. Their structural model of software measurement consists of two parts: an empirical world and a formal world. The empirical world contains entities that can have certain properties, called attributes. The formal world consists of values that measure the attributes of entities, expressed in certain units. Measurement now, is the mapping of a particular entity and attribute from the real world to a value in the formal world. The generic process model reflects the differences between these two worlds: measurement activities (the right half) are concerned with constructing a formal world based on the real world, whereas improvement activities (the left half) are concerned with changing the real world based on the formal world created by the measurement activities.

7.2 Measurement-based improvement at Ericsson

In this section, we use the measurement-based improvement model presented in the previous section to assess the measurement program employed by a department of Ericsson in the Netherlands. The goal is to show the feasibility of the model to describe a successful measurement program, and to show that the mea-

surement program implemented by Ericsson indeed covers all four phases of the measurement-based improvement lemniscate.

Ericsson ETM/BL/RU is a software development centre, more specifically a Unix Development Group (UDG), in the Netherlands, which was assessed at level three of the Software CMM in 1995. The development centre is part of the research and development department of the Ericsson Telecommunications firm. At the moment, the organization consists of 60 employees. The main product of the organization is the Formatting and Outputting Subsystem (FOS). The FOS is a C++ application running on an adjunct processor which processes and outputs charging data produced by the central processor of an Ericsson AXE telephone exchange. The customers of the UDG are sharply focussed on cost and time to market of software development. At the same time, performance (speed), fault tolerance and security demands are high.

These high and potentially conflicting demands make it important for the UDG to be able to communicate with its customers about the cost of software quality, i.e. how much it costs and how long it takes to develop software of a certain quality. Consequently, the UDG also needs to be able to quantitatively manage and control its software process. Therefore, the organization is currently implementing the CMM[2]. level four key process areas Statistical Process Management, Organization Process Performance, and Organization Software Asset Commonality.

For each of the four phases of the measurement-based improvement model we investigate whether the UDG has implemented them, how the activities are implemented, and how the activities contribute to the goal of a quantitatively controlled software process. First, we look at the organizational context of the measurement program and the reasons why Ericsson UDG decided to implement level four of the Software CMM. Next, section 7.2.2 discusses how the measurement program was implemented, what is being measured and how the measurement program is embedded in the normal development process. Section 7.2.3 shows the results of the measurement program. What data was gathered, and what analyses were done. The last of the four stages is described in section 7.2.4. Here we discuss the organizational changes and improvements that were implemented based on the results of the measurement program.

7.2.1 Organizational analysis

In this section, we discuss the organizational context which led the UDG to implement a measurement program. The challenges facing software development at Ericsson's Unix Development Group (UDG) are posed from three general sources:

[2]UDG uses draft C of the Software CMM version 2.0, which contains different key process areas on level 4 than version 1.1 of the Software CMM.

the telecommunications market, the overall organization and the product. These are discussed below.

Ericsson is a global organization. Development of new products is distributed over development centres all over the world. Ericsson Telecommunication B.V. is one of the design centres involved in developing software products for the AXE telephone exchange. Each design centre develops part of the product to be delivered. This distributed development places a high demand on lead-time precision of development. All the local companies must deliver new products in orchestration so that the final product reaches the customer on time. Hence, estimating software project cost and lead-time are important for Ericsson. In addition, the telecommunications market is highly competitive, new competitors and new products emerge very quickly. For the Unix Development Group this translates to developing innovative and high quality software products fast and in orchestration with many other development centres distributed world-wide. The main product of the UDG is the Formatting and Outputting Subsystem (FOS). This C++ application formats call-related data generated by the central processor of an AXE telephone exchange. The formatted data is then sent to post-processing systems of the AXE operator. The nature of the data being formatted places high reliability demands on the FOS. The FOS runs on an adjunct processor and must match the ever-increasing capacity and performance (speed) of the AXE exchange.

To meet the high and potentially conflicting demands on the software processes, Ericsson uses the Software Capability Maturity Model to get a grip on software development. Started in 1993, Ericsson reached the third level in 1995. In order to further control and manage its software process, UDG decided to start implementing the fourth level of the Software CMM. The goal UDG wanted to reach is to be able to quantitatively visualize and control its software development process. Specifically, the UDG wants to control the fault content in the packages it delivers to its customer.

7.2.2 Measurement implementation

One of the key process areas of level four of the Software CMM is concerned with Statistical Process Control. SPC (Montgomery 1985) is a well-known theory, using numerical data and statistical methods to visualize and control the performance of the primary process. Measurement activities were actually started when Ericsson progressed towards level three in 1995. Process measures like test results, product volume, lead-time and effort per product and process activity were collected. However, the enormity of the available data and the difficulty in identifying key performance indicators made a structured way of handling all this information impossible.

Up to this point, analyses and reporting of measurement results had been restricted to a small group of project-, process- and quality managers. There was very little feedback to the developers about the results of the measurements. The main purpose had been to provide visibility to a select audience of managers and customers.

The measurement system which was set up next was based on a set of prerequisites. First of all, no new measures were to be defined. Implementation should start with the available data. Second, the measurement system should not become a separate process. All measurements must have a place in the normal workflow of software development and project management. Third, the information yielded by the measurement system should provide a solid base for decisions.

The first step taken was to define organizational templates for measurement definitions in order to ensure the reliability and validity of the measures. Also, the purpose and usefulness of each measure was to be defined. These templates triggered analyses of the historical database yielding a small number of well-defined critical process performance indicators. These were introduced through a pilot. Pilots at the UDG mean that a single project will use a limited number of measurements for the duration of that project. If a pilot is successful, the piloted measurements are integrated into the software development processes.

One of the goals of the measurement program was to control the fault content in products. This goal was translated to a goal for each of the test phases of a project. Projects at Ericsson's UDG have three test phases of which the basic test is the earliest in the development life cycle. The purpose of measuring fault density is to help prevent faults and possible risks from slipping through to the next test phase in the development process. The first step taken was to analyze historical data on fault density basic test in order to get a good insight in the usability, validity and reliability of the measure. It turned out that approximately two-thirds of the components with unusual high or low fault densities in basic test also caused problems in subsequent test phases. The decision was made to pilot the statistical use of this measurement in the next project.

Fault density basic test is the number of basic test faults reported per thousand non-commented source statements. So basically there are two measurements involved; the number of basic test faults and the number of statements. Measuring these was already part of the normal basic test activities. So defining the measurements formally did not involve very upsetting changes for developers. The formal definition of the measurements did however improve the reliability of the measurements.

The measurements are defined by Measurement Data Definitions (MDDs). These workinstruction-like MDDs were inspected by the developers responsible for executing the measurement. Through these inspections, differences in interpre-

tation by the developers were resolved. It also helped in clarifying the purpose of the measurements and raising awareness of statistical methods. At the same time a Measurement Result Definition (MRD) was drafted. This definition describes how to get from the basic data to a fault density basic test chart, and how to interpret and use the results.

At the UDG a series of presentations was given in the early stages of the definition of the MRD. The goal was twofold. First, to introduce developers and management to the MRD and to make clear that interpretation of results would be rigorously defined. Second, to get feedback from developers on the influencing factors listed in the MRD.

In the next section we discuss how UDG analyses the data gathered and how the numbers are interpreted to form a basis for further decision making.

7.2.3 Measurement analysis

The use of statistical methods contributed to improving lead-time precision. Control charts on fault content have helped to prevent defects from slipping through to subsequent development phases.

The MRD 'Fault Density Basic Test' features two presentations: a cumulative line-chart for tracking the trend of the results and a control chart. The MRD is executed by the developers each time they finish a basic test report. The presentation of all basic test report results is the responsibility of Quality Management. The control chart is presented on a weekly basis, the trend line chart monthly.

For fear of misinterpretation or hasty conclusions based on incomplete information, results are always presented in combination with analysis results. These analyses are triggered by control limits defined in the MRD.

The control chart features two sets of limits: control limits and specification limits. Control limits represent the normal statistical variance of basic test results. These limits were arrived at by standard statistical methods based on normal historical data. Normal historical data are all results from a regularly executed process. Any measurement results that were not the output of a normally executed software development process were excluded from the data set prior to calculating the control limit. The result was a bandwidth of normal and expected behavior. The calculated upper control limit was however higher than the goals set on fault content. So in order to ensure that process results conform to the goals, specification limits were set. Any value outside the specification limits may pose a risk for goal achievement. Specification limits are typically narrower than control limits. Values outside specification limits are analyzed on possible (technical) risks for subsequent test phases and statistical deviations. With a measure like fault density, it may very well be that a single reported fault on a very small component produces

a result outside specification limits.

Any value outside control limits may indicate irregularities in process execution or the product itself. If a package is outside control limits an additional analysis on process executions is performed. Results outside the limits thus trigger analyses. The result of the analysis combined with other measurement results and an evaluation of influencing factors determine if, and what action is to be taken.

7.2.4 Organizational implementation

The last stage of the measurement-based improvement process model is concerned with actual usage of the measurement results to improve the organization. When a software component displays more defects than may be expected from the statistically determined variance of the process, a technical and statistical analysis is done. In some cases this has led to the identification of technical issues. These were solved before the component was handed over to the next phase. In other cases structural process issues were identified and solved by process improvements.

Two examples where results exceeded the limits illustrate how measurement and analysis triggered action at the UDG. The first instance was a component whose basic test results exceeded the specification limits. The component in question was (statistically) normal. The team performed a technical risk analysis and identified four risks of which they proposed to solve two. The team spent five extra hours on the package before it was handed over to the next test phase. During this subsequent test phase the test team was asked to evaluate the risks and the solutions implemented by the developers of the previous phase. The test team indicated that had the risks not been solved, they would have had to spend forty extra hours on the test runs. This ratio of one to eight was also found in other components of which identified risks were solved before hand-over to the next test phase.

In another instance, a component's result exceeded the control limits. Not only did the package harbor significant technical risks, on further analysis flaws in the organization and the process were uncovered. The risks were solved before hand-over to the next phase. In addition, the process and organization issues uncovered were reported to senior management, and resulted in a change of the development process.

The examples given led to two major changes in the organization's standard software process. The first instance, where a risk was solved before hand-over, led to a work instruction for the test team. The test team already evaluated the basic test record before accepting a component. This evaluation was extended with the instruction not to accept a component with basic test results outside the limits without a risk analysis and action plan. The second instance led to addition of an extra process step and a better set-up of team planning.

7.2.5 Summary

Using the measurement-based improvement process model, we have shown why the Ericsson UDG implemented a measurement program, how it was set up, how the results were analyzed and interpreted, and how the results were used to actually improve the organization and its software process. We can safely conclude that this measurement program is a successful program. We feel that an important determinant of its success is the fact that each of the four steps from the process model received sufficient attention: the measurement was driven by a clear goal, the implementation of the measurement program was done rigorously, the data analysis was appropriate, and the results of the measurement program were fed back into the organization by using the results to improve the software process.

7.3 Comparing measurement program guidelines

In this section we use the generic process model described in section 7.1 to compare different frameworks for measurement programs. In each subsection we indicate which activities and processes the respective frameworks prescribe, and position these activities and processes on the generic process model.

We discuss measurement program guidelines from six different sources: the Measurement Technology Maturity Model described by Daskalantonakis, Yacobellis and Basili (1990-1991); the Measurement Maturity Model presented by Comer and Chard (1993); the Goal-oriented Measurement Process described by Briand, Differding and Rombach (1996); the success factors for measurement programs identified by Jeffery and Berry (1993); the success factors for measurement programs by Hall and Fenton (1997); and the Measurement Capability Maturity Model described in chapter 6.

7.3.1 The software measurement technology maturity framework

Daskalantonakis, Yacobellis and Basili (1990-1991) define a framework to be used to assess the measurement technology level of an organization. The article defines five levels of measurement technology maturity, divided into 10 themes, listed in table 7.1.

The five levels of maturity are similar to the Software CMM; i.e. initial, repeatable, defined, managed and optimizing. However, the model does not prescribe any processes like the Software CMM does. Instead, the model gives characterizations of each of the ten themes on each maturity level. For example, the characterizations of the third theme, ‘scope of measurement’, look as follows (Daskalantonakis, Yacobellis and Basili 1990-1991):

1	Formalization of the development process
2	Formalization of the measurement process
3	Scope of measurement
4	Implementation support
5	Measurement evolution
6	Measurement support for management control
7	Project improvement
8	Product improvement
9	Process improvement
10	Predictability

Table 7.1: Software Measurement Technology Maturity Framework themes

Level 1. Done occasionally on projects with experienced people, or not at all.

Level 2. Done on projects with experienced people. Project estimation mechanisms exist. Project focus.

Level 3. Goal/Question/Metric package development and some use. Data collection and recording. Existence of specific automated tools. Product focus.

Level 4. Metric packages being applied and managed. Problem cause analysis. Existence of integrated automated tools. Process focus.

Level 5. Have learned and adapted metric packages. Problem prevention. Process optimization.

The major difference between the approach taken by Daskalantonakis *et al.* and the other approaches described in this chapter is that the first uses a more declarative description of the different levels of measurement technology maturity. Instead of describing the activities organizations need to implement, the results of these activities are specified. Other approaches put much more emphasis on the activities and processes themselves that organizations need to implement; i.e. they follow an imperative approach. This declarative nature of the measurement technology maturity framework makes it difficult to map it onto our generic process model. We have to translate the declarative specification of the themes into corresponding activities. Figure 7.3 shows our approximation of how the ten themes could be placed in the model.

Themes two, three, and four are concerned with the implementation of a measurement process. This measurement process includes automation of data collection, evaluation, and feedback. Themes seven, eight, and nine prescribe the usage

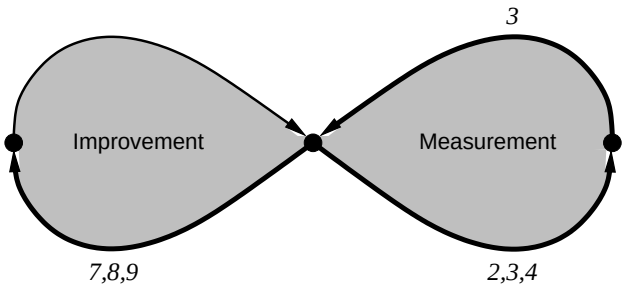


Figure 7.3: Mapping of the Software Technology Maturity Framework on the generic process model

of measurement data to improve projects, products, and processes, respectively. Unfortunately, the descriptions are more in terms of the results of the improvements than in terms of improvement activities to perform.

The other themes do not fit into the generic model. Theme one is concerned with the formalization of the development process, and essentially coincides with the Software CMM, and hence does not fit into the pretzel. We were not able to translate themes five, six, and ten into corresponding activities. Theme five, 'measurement evolution', is concerned with the types of measures that are taken. Measuring different kinds of measures does not necessarily change the activities needed, so we cannot place this theme in our process model. The same holds for the sixth theme, 'measurement support for management control'. This theme is specified in terms of the type of support management receives from measurement. Different types of support do not necessarily require different activities. Theme ten, 'predictability', describes how the predictability of measures increases as the maturity level of an organization increases. Again, this theme cannot directly be translated into measurement or improvement activities.

7.3.2 A measurement maturity model

Comer and Chard (1993) describe a process model of software measurement that can be used as a reference model for the assessment of software measurement process maturity. Unlike the maturity models described in sections 7.3.1 and 7.3.6, the measurement maturity model of Comer and Chard does not define different levels of maturity. The model consists of four key processes, derived from different sources:

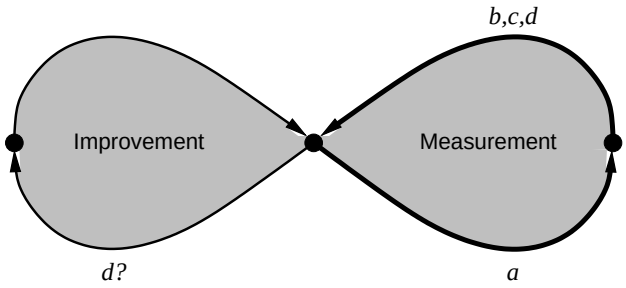


Figure 7.4: Mapping of the Measurement Maturity Model on the generic process model

a. **Process Definition** This process includes activities such as: specification of the products, processes, and resources in need of tracking or improvement; identifying goals of the organization and the development environment; derivation of metrics which satisfy the goals.

b. **Collection** Activities in the collection process include defining the collection mechanism, automation of the measurement gathering, implementing a measurement database, and data verification.

c. **Analysis** Data analysis.

d. **Exploitation** Exploitation of analyses to improve the software development process.

Unfortunately, Comer and Chard do not elaborate on the processes ‘analysis’ and ‘exploitation’, which makes it somewhat difficult to map them onto the generic process model. We assume the process ‘analysis’ consists of analyzing the gathered data, and relating the data to measurement goals. The exact borders of the ‘exploitation’ process are undefined. Especially the extent to which this process covers the actual activities needed to improve the software process remains unclear.

Figure 7.4 shows how the four processes in our opinion map onto the generic process model. The process ‘exploitation’ has been placed on the lower left arrow with a question mark, because the paper provides insufficient information to decide to what extent that process is meant to cover the implementation of solutions.

1	Characterize the environment
2	Identify measurement goals and develop measurement plans
3	Define data collection procedures
4	Collect, analyze, and interpret data
5	Perform post-mortem analysis and interpret data
6	Package experience

Table 7.2: Process for goal-oriented measurement

7.3.3 Goal-oriented measurement

Briand, Differding and Rombach (1996) present a number of lessons learned from experiences with goal-oriented measurement. Goal-oriented measurement is described as 'the definition of a measurement program based on explicit and precisely defined goals that state how measurement will be used'. The process for goal-oriented measurement consists of six process steps, displayed in table 7.2.

During the first step the relevant characteristics of the organization and of its projects are identified. Typical questions to be posed are: What kind of product is being developed? What are the main problems encountered during projects? The characterization is intended to be mainly qualitative in nature. In the second step, measurement goals are defined, based on the characterization made during the first step. Measurement goals are defined according to Goal-Question-Metric templates (Basili, Briand, Condon, Kim, Melo and Valett 1996, Basili and Rombach 1988), based on five aspects: object of study, purpose, quality focus, viewpoint, and context. Having defined the measurement goals by means of the GQM templates, data collection procedures are defined during step three. Step four is concerned with the actual collection, analysis, and interpretation of the gathered data. Step five puts the data in a broader perspective by e.g. comparing the gathered data of one project with the organization baseline. The final step consists of packaging the data analysis results, documents, and lessons learned in a reusable form.

Figure 7.5 shows how the six steps of goal-based measurement map onto the pretzel. Step one is concerned with the analysis of the organization and its problems and goals, and thus corresponds with the upper left arrow. Steps two and three deal with the translation of organizational goals into measurement goals and the design of the measurement program (lower right arrow). The last three steps consist of the collection, analysis, interpretation, and packaging of the measurement data, hence they belong to the upper right arrow.

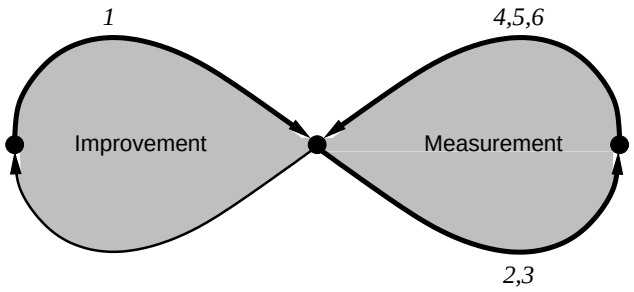


Figure 7.5: Mapping of the goal-oriented measurement process on the generic process model

C1	The goals of the measurement program are congruent with the goals of the business.
C2	The measured staff participate in the definition of the measures.
C3	A quality environment has already been established.
C4	The development processes are sufficiently stable.
C5	The required granularity can be determined and the data is available.
C6	The measurement program is tailored to the needs of the organization.
C7	There is senior management/sponsor commitment.
C8	The objectives of the measurement program are clearly stated.
C9	There are realistic assessments of the pay-back period(s).

Table 7.3: The Berry and Jeffery success factors (context).

7.3.4 A framework for evaluation and prediction of metrics program success

Jeffery and Berry (1993) identify a number of organizational recommendations for the establishment of measurement programs. The recommendations originate from other literature (such as Grady and Caswell 1987, Fenton 1991, Basili 1990, Selby, Porter, Schmidt and Berney 1991, Musa, Iannino and Okumoto 1987). Jeffery and Berry distinguish four perspectives on measurement programs:

- *Context*: the environment in which the measurement program is developed and operated.

- *Inputs*: factors or resources that are applied to the measurement program.

I1	The measurement program is resourced properly.
I2	Resources are allocated to training.
I3	At least three people are assigned to the measurement program.
I4	Background 'research' into the measurement programs and their effectiveness is being done.

Table 7.4: The Berry and Jeffery success factors (input)

PM1	The program is promoted through the publication of success stories, and it encourages the exchange of ideas.
PM2	A firm implementation plan is available.
PM3	The program is not used to assess individuals.
PR1	The measurement team is independent of the software developers.
PR2	Clear responsibilities are defined.
PR3	The initial collection of measures is properly 'sold' to the data collectors.
PC1	There are important initial measures defined.
PC2	There are tools, acquired or developed, for automatic data collection and analysis.
PC3	There is a 'persistent' measures database.
PC4	There is a mechanism for changing the measurement system in an orderly way.
PC5	Measurement is integrated into the process.
PC6	Capabilities are provided for users to explain events and phenomena associated with the project.
PC7	The data is 'cleaned' and used promptly.
PC8	Well-defined objectives determine the measures.
PT1	Adequate training in software measurement is carried out.
PT2	Everyone knows what is being measured and why.

Table 7.5: The Berry and Jeffery success factors (process). *PM* stands for process motivation and objectives, *PR* for process responsibility and metrics team, *PC* for process data collection, and *PT* for process training and awareness.

P1	The measures are clear and of obvious applicability.
P2	The end results provide clear benefits to the management process at the chosen management audience levels.
P3	Constructive feedback on results is provided to those being measured.
P4	The measurement system is flexible enough to allow for the addition of new techniques.
P5	Measures are used only for predefined objectives.

Table 7.6: The Berry and Jeffery success factors (product)

- *Process*: the method used to develop, implement, and maintain the program.
- *Product*: the measures taken, reports produced, and other output of the program.

From the literature, the authors synthesize a list of questions to be asked to assess measurement program success, divided into the four perspectives. Basically, the questions ask whether each of the success factors is being adhered to. Tables 7.3, 7.4, 7.5, and 7.6 list the success factors as they were published in Offen and Jeffery (1997).

The authors use the list of questions to assess three measurement programs in three different organizations. A scoring scheme is used in which each question is assigned a score between zero and four, depending on the extent to which the requirement of each question was fulfilled. The authors conclude that the scores obtained by means of the list of questions support their own qualitative assessment of the success of the three measurement programs described in (Jeffery and Berry 1993).

Figure 7.6 shows how the success factors from the four tables fit onto the process model. We see that 7 of the 9 context factors apply to the first phase of the process model. Most of the factors apply to the measurement implementation phase. The measurement analysis phase is covered by one input factor, three process data collection factor and one product factor. Finally, the organizational implementation phase is touched upon by three factors. However, two of them are warnings: *PM3* forbids the usage of the measurement program to assess individuals, and *P5* says to only use the measures for predefined objectives. Only *P2* states that end results should provide clear benefits to the management process.

7.3.5 Success factors for measurement programs

Hall and Fenton (1997) identify a number of consensus success factors for the im-

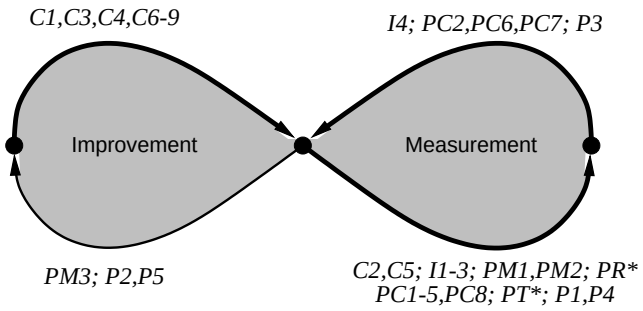


Figure 7.6: Mapping of the Jeffery and Berry success factors on the generic process model. *PR** and *PT** are abbreviations for all *PR* and *PT* success factors.

plementation of measurement programs. Table 7.7 shows these factors, that were identified after studying other literature (such as Grady 1992, Pfleeger 1993). A closer look at the success factors shows that they are mainly targeted at reducing the risk of failure. For example, the motivation given by Hall and Fenton for factor six – usefulness of metrics data – is not that the measurement program should have added value for the organization, but rather that the usefulness should be obvious to the practitioners. From the 15 success factors, 10 are targeted at gaining the acceptance of the practitioners involved (4-9, 11, 13-15). The other five factors are concerned with reducing the risk of failure by advocating a gradual introduction and improvement of the program. The measurement program should be incrementally implemented, constantly improved, use existing materials, be supported by management, and a well-planned metrics framework should be used (1-3, 10, 12).

Figure 7.7 shows how the success factors can be mapped onto the generic process model. The majority of the success factors mentioned by Hall and Fenton refer to the implementation of measurement programs. Some are concerned with the collection and analysis part, and only one success factor is concerned with the usage of the measurement data (factor nine). That factor is marked with a question mark, because Hall and Fenton motivate it in terms of acceptance of the measurement program by the practitioners, rather than in terms of added value of the program to the company.

7.3.6 The measurement capability maturity model

In chapters 3, 4, and 5 we have described a number of measurement program case studies. From these case studies we concluded that some organizations are better at software measurement than other organizations. Part of this difference can be

1	Incremental implementation
2	Well-planned metrics framework
3	Use of existing metrics materials
4	Involvement of developers during implementation
5	Measurement process transparent to developers
6	Usefulness of metrics data
7	Feedback to developers
8	Ensure that data is seen to have integrity
9	Measurement data is used and seen to be used
10	Commitment from project managers secured
11	Use automated data collection tools
12	Constantly improving the measurement program
13	Internal metrics champions used to manage the program
14	Use of external metrics gurus
15	Provision of training for practitioners

Table 7.7: Consensus success factors

explained by the fact that their *measurement capability* is higher; i.e. they are more mature with respect to software measurement. Measurement capability was defined as 'the extent to which an organization is able to take relevant measures of its products, processes and resources in a cost effective way, resulting in information needed to reach its business goals' on page 80.

Our Measurement CMM defines five different levels of organizational measurement capability, similar to the Software CMM:

1. **Initial:** The organization has no defined measurement processes, few measures are gathered, measurement that takes place is solely the result of actions of individuals.

2. **Repeatable:** Basic measurement processes are in place to establish measurement goals, specify measures and measurement protocols, collect and analyze the measures and provide feedback to software engineers and management. The necessary measurement discipline is present to consistently obtain measures.

3. **Defined:** The measurement process is documented, standardized, and integrated in the standard software process of the organization. All projects use a tailored version of the organization's standard measurement process.

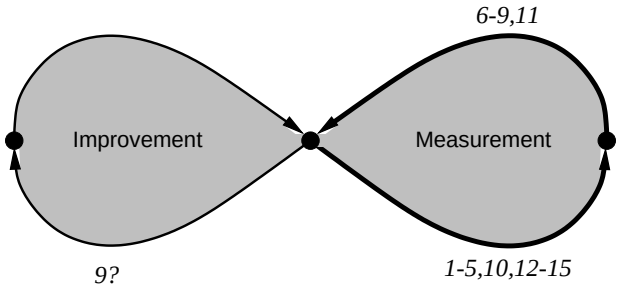


Figure 7.7: Mapping of the Hall and Fenton success factors on the generic process model

4. **Managed:** The measurement process is quantitatively understood. The costs in terms of effort and money are known. Measurement processes are efficient.

5. **Optimizing:** Measurements are constantly monitored with respect to their effectiveness and changed where necessary. Measurement goals are set in anticipation of changes in the organization or the environment of the organization.

Each of the maturity levels is defined by a number of key process areas that an organization needs to implement. When an organization has implemented all level-two key process areas, the organization is considered to be at level two of the M-CMM. When the organization implements both the level two and three key process areas, it is at level three, etc. The key process areas of the Measurement CMM are listed in table 7.8, numbered by maturity level.

Figure 7.8 shows the M-CMM applied to the generic process model. It is not surprising that all of the key process areas map onto the right half of the 'pretzel'. After all, we made a clear choice in the development of the Measurement CMM to focus on the measurement capability of software organizations, thereby ignoring their capability with respect to improvement. Our argument in chapter 6 was that the improvement capability is already covered by process improvement methods, such as the Software CMM. We assumed that the organizational goals are defined outside the scope of the M-CMM, so they are invariable from a measurement point of view. The measurement process then starts with the translation of business goals into measurement goals, and ends with the interpretation of the gathered data and feedback to the owner of the business goals.

2a	Measurement Design
2b	Measure Collection
2c	Measure Analysis
2d	Measurement Feedback
3a	Organization Measurement Focus
3b	Organization Measurement Design
3c	Organization Measurement Database
3d	Training Program
4a	Measurement Cost Management
4b	Technology Selection
5a	Measurement Change Management

Table 7.8: Measurement CMM key process areas

7.3.7 Differences and similarities

In the previous sections, we have compared six different measurement program frameworks, using the generic process model for measurement-based improvement. There are a number of issues that deserve attention.

First, if we look at the intersection of the guidelines provided by the different approaches, we see that there is quite some consensus on what activities are needed to successfully design and implement measurement programs. Though the frameworks all stress different aspects of measurement programs, they agree on the basics of measurement programs, such as practitioner support, proper data analysis, feedback, etc.

Second, each of the approaches seems to offer guidelines that the other approaches do not offer. This probably is partly due to the different structure and nature of the frameworks. However, it does suggest that beyond the basic requirements for measurement programs, there are a number of issues on which consensus has not been reached yet. For example, only Briand et al. prescribe the packaging of measurement experiences in a reusable form. Of the frameworks discussed, Hall and Fenton are the only ones to advocate the use of external measurement guru's.

Third, if we look at each of the pretzels used to show how the activities of the different approaches map onto the measurement-based improvement process, we see that none of the frameworks covers the complete cycle. Either the frameworks only cover measurement activities, or they only partly cover the improvement activities. One could argue that this is logical, since these measurement program frameworks focus on the implementation of measurement programs, and not on improvement activities. However, failure factors for measurement programs sug-

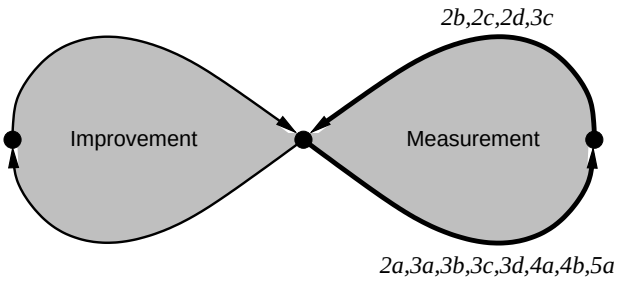


Figure 7.8: Mapping of the Measurement CMM key process areas on the generic process model

gest otherwise. Take for example the failure factors for measurement programs as suggested by Verdugo, reported by Fenton and Pfleeger (1997, p. 511):

1. Management does not clearly define the purpose of the measurement program and later sees the program as irrelevant.

2. Systems professionals resist the program, perceiving it as a negative commentary on their performance.

3. Already burdened project staff are taxed by extensive data-collection requirements and cumbersome procedures.

4. Program reports fail to generate management action.

5. Management withdraws support for the program, perceiving it to be mired in problems and 'no-win' situations.

From these failure factors we see that both support from practitioners for the measurement program, as well as management support, is important. A measurement program will fail if practitioners do not see the value of it, but it will also fail if management fails to take action based on the generated data. This means that a successful measurement program needs more than carefully designed metrics, accurate data analysis, measurement databases, etc. It needs to be used. Any measurement program that does not generate any action is to be considered a failed measurement program. After all, like the proof of the pudding is in the eating, the proof of software measurement is in its usage for improvement.

In the next section, we will explore possible success factors for the left hand side of the measurement-based improvement process model. In addition, we will suggest actions that can be taken to ensure the fulfillment of these success factors.

7.4 Success factors for measurement-based improvement

In this section we first describe a number of common uses of measurement programs in section 7.4.1. From those uses we derive a number of success factors that are external to measurement programs. Next, in section 7.4.2, we propose a number of steps organizations could take to fulfill the success factors identified in section 7.4.1.

7.4.1 Possible uses for measurement programs

Measurements should generate value to the organization. This value is determined outside the measurement program proper. A major factor determining the success of a measurement program is whether or not it actually does create that value. In the previous section we have shown that measurement program success factors such as listed by Hall and Fenton (1997) focus on the measurement program internals. In this section, we investigate different situations in which a measurement program may be used to gather data needed to solve organizational problems or help reach organizational goals. The purpose is to derive additional success factors, external to the measurement program, but nevertheless essential for the success of the measurement program.

Measurement programs can serve many purposes, and hence create value to a software organization in different ways. In our experience, the main kinds of purposes for which measurement programs are used are:

- **Reporting** A situation where there is a contractual obligation to reach certain targets. For example, a software maintenance organization may guarantee in its service level agreements some level of availability of a system, or some maximum down-time. The actual performance of the organization is then monitored, and results are reported to the customer. Often, the agreement explicitly states penalties incurred in case of non-fulfillment of the agreement.

  What measurements need to be taken for reporting purposes can fairly easily be derived from the service level agreement at hand. However, the measurement program's value can be improved by not only measuring the service levels covered by the service level agreement, but also factors that enable

the organization to predict situations that might cause the service levels to be violated. For example, if a service level agreement includes a threshold on the maximum response time of certain information systems, the service provider might want to measure the load of the server that runs the software. That way the service provider can prevent high response times by keeping the server load low enough.

- **Monitoring performance** In this situation, someone (usually management) sets the standards, usually in terms of a set of performance indicators, and measurements serve to see whether these levels of performance are met. The main difference with the reporting case is that the 'customer' of the data is external in the reporting case, while it is most often internal in this case.

  It is vital for the success of this kind of measurement program that the organization has a clear plan on how to act if the desired performance is not being achieved. For example, if the organization wants to measure schedule slippage, it also needs to be prepared to take measures to improve schedule and planning accuracy. The latter type of measure is often not dealt with explicitly. As a result, the organization is likely to play the ostrich in case expectations are not met.

- **Learning** The organization has a problem but does not immediately see a solution to it. First, it needs to investigate the problem more thoroughly, and find the root causes, or main underlying factors, that cause the problem.

  For example, a software maintenance organization performs corrective maintenance for a large number of customers for a fixed price per period. It needs to be able to estimate the corrective maintenance workload (i.e. the expected number of bugs) to be able to set a reasonable price. The software maintenance organization starts a measurement program to identify the main factors that determine the corrective maintenance workload. If those factors are found, the organization could use this information in the form of an estimation procedure to support the bidding process for new contracts. Probably, the organization will also want to keep monitoring both the factors and the actual corrective maintenance workload for the different contracts in order to calibrate the estimation procedure.

- **Performance improvement** In this case, certain relations between (product and/or process) variables are assumed or formulated. For example, a software development organization assumes that the later bugs are fixed during the development process, the more expensive the fix is. The organization decides to strive for phase containment of faults (Hevner 1997). A measure-

ment program is then started to gather the necessary data. Next, the data are analyzed, and actions are taken based on the outcome of the analysis. For example, measures can be taken to improve the in-phase detection of faults. This process usually is a cyclic one, whereby hypotheses get formulated, refined or rejected, and new hypotheses guide the next cycle.

- **Organizational health** This is kind of a check-up. The organization is compared against a set of norms (usually created externally). In this case, it is most interesting to consider the case where the norms are *not* met. What kind of action, if any, will be taken in that case? And how does the check-up help in deciding what the best actions would be? In the case of an assessment of the software process against a set of norms like put down by the Software CMM (SEI 1995, McFeeley 1996), the assessment results in a list of recommendations for improvements. In the case of a benchmark against industry averages, the actions that should be taken as a result of the comparison are less clear.

- **Navigation** In this situation, management determines a destination, or at least a direction for travel. Next, a plan is made how to get there. During the subsequent journey, measurements are used to answer questions like 'How well am I following the plan?', 'Have I reached my destination yet?', or 'Was the journey worth it?'. Again, it is generally worthwhile to pay special attention to cases where the answer to these questions is negative.

From this list of typical applications of measurement programs, four success factors – external to the measurement program – emerge:

1. Various assumptions underlie the measurement program. These assumptions should be made explicit and it should be decided if and when these assumptions are tested. These assumptions often take the form of a cause-effect relation between anticipated changes and a desired result.

2. Different outcomes can result from a measurement program. An organization should consider all possible – negative and positive – outcomes and decide how to act on them. Often, only one of these possible outcomes is satisfactory: performance is OK, targets are met, etc. It is the other possible outcomes that are most interesting from our point of view: what happens if the performance is not OK, targets are not met, etc. If it is not specified what to do in those cases, there is quite a chance that *nothing* will be done.

3. The organization should act according to the outcomes of the measurement program, in order to reach the goals set or solve the problems identified. This

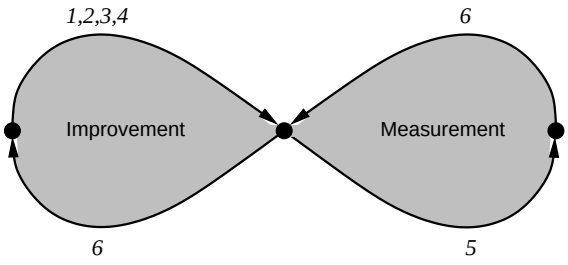


Figure 7.9: Steps for measurement-based improvement

applies to both negative and positive outcomes. If the organization does not act, the value of the measurement program degrades, and it will sooner or later, but usually sooner, come to an end.

4. The organization should monitor the changes implemented, in order to verify that these changes indeed constitute an improvement for the organization. Measurement involves modeling, and thus abstracting away from many aspects. We should verify that our model captures reality sufficiently well, and keeps doing so if reality changes over time. Also, it should be verified whether the desired outcome is brought about (by the changes implemented or for any other reason).

In the next section, we propose a number of activities that organizations can follow to fulfill the success factors described above.

7.4.2 Steps for measurement-based improvement

In this section we describe steps an organization could take to fulfill the success factors identified in the previous section. These steps are illustrated using an example, based on the measurement program described in chapter 4.

1. Determine a valuable outcome of the measurement and/or improvement program. The organization in question explicitly determines what results it expects from the measurement and/or improvement program and how these results are to be measured.

   For example, a software maintenance organization and its (only) customer have difficulties determining a fair price for change requests. Together, the software maintenance organization and the customer decide to implement

function points – more specifically maintenance function points – as a means to determine the price of change requests. Hence, the valuable outcome of this improvement initiative is to have an objective mechanism to determine a fair price for change requests.

2. Assumptions about relationships between changes to be made and results to be obtained are made explicit.

   Note how the organization assumes that: (1) the maintenance function points will indeed be an objective measure of the volume or size of a change request, and (2) the number of maintenance function points of change requests are correlated with the effort needed to implement those changes, which is needed for a reasonable fair price.

3. Develop a plan to obtain this outcome. Improvement can only be achieved by changing the organization in one or more respects. The plan determines how and what is going to be changed. It is decided which assumptions need to be tested before the changes are implemented, and which are checked during or after implementation of the changes. Hence, this step also results in the measurement goals to be fulfilled by the measurement program.

   In our example, the software maintenance organization and the customer design a new change request planning procedure which includes the counting of maintenance function points of each change request to determine its price. Because this is the first time this particular function point model is being used, it is decided to use the model for a while and then analyze its behavior. Specifically, correlation with the effort needed to implement the changes will be investigated.

   So, the measurement program to be implemented needs to fulfill two requirements: (1) provide the necessary information to apply the maintenance function points, and (2) provide the information needed to evaluate the maintenance function point model.

4. Follow-up scenarios are developed. For each possible outcome of the measurement program, scenarios are developed that describe how to act on that particular outcome.

   If the correlation between maintenance function points and effort is lower than a certain value, the model structure will be adjusted. For example, the model makes certain assumptions about the cost of deleting functions: it states that deletion of a function costs 20% of the effort needed for building the function. If needed, that factor of 0.2 can easily be adjusted using the effort data.

This completes the first phase (the upper-left arrow) of the measurement-based improvement process model. One way to continue from here is to set up a measurement program, analyze its results, and only then implement some changes. In that case, we apparently are not quite sure yet whether the assumptions made in the previous steps really hold. In case we are very confident about these assumptions, we may decide to just implement the changes, and not bother about measurements at all. An intermediate form is to do both at the same time: some changes are implemented, and at the same time a measurement program is started to be able to do an a posteriori check on the viability of those changes. In general, it depends on the situation at hand which of these continuations is to be chosen. In the example we are considering here, it is reasonable to follow the last one identified. So we will start to use function points as an objective effort measure, and at the same time we start a measurement program in order to be able to test our function point model.

5. Design and implement the improvements and the measurement program.

   The new planning procedure is implemented. The measurement program to gather the function points and the effort data is implemented. The organization develops a detailed measurement protocol and counting guidelines. The measures to be taken are the input data for the function points, i.e. data element types, record element types, etc., and the effort data per function changed, needed for the evaluation.

6. Act upon the measurement program (step 5) according to the scenarios developed in step 4.

   After a while, the measurement data are analyzed and, depending on the outcome, one of the scenarios developed in step 4 is executed. In this case, the function point model assumes that the size of a function changed and the size of the change itself contribute equally to the effort needed for the change (i.e. changing a function of size $2x$ takes twice as much effort as changing a function of size x and changing 60% of a function takes twice as much effort as changing 30% of a function). However, the analysis shows that the size of the function is much more important for determining the effort than the relative portion of the function that is changed. Hence, the function point model is changed to reflect these findings, and the procedures are adapted to the new version of the model.

The steps listed are an example of how an organization could implement measurement-based improvement, making sure that the success factors described in the previous section are taken into account.

Note that though the example as described above is fictional, it is based on a real measurement program, as described in chapter 4. In reality, the organization did not make the assumptions listed in step 2 explicit. We were asked to analyze the function point model. The fact that we were indeed able to analyze it was a mere coincidence: the effort data was for a large part recorded on the level of changes to individual functions, where it could have been recorded at a more coarse level of granularity just as well.

When we discovered that the model needed structural changes to improve its correlation with effort, the organization was not prepared to make those changes. One of the reasons was the fact that the organization was rather happy with the model, despite the low correlation, because it solved part of the problem, i.e. it provided an objective pricing mechanism for change requests. The fact that the function model could need calibration was not explicitly recognized up front. A commitment to act upon the outcome of the measurement program was not made. Not surprisingly, our findings were not implemented, and things stayed as they were before our analysis of the function point model.

7.5 Conclusions

In this chapter we have introduced a generic process model for measurement-based improvement. In section 7.2 we have used the process model to describe the measurement program of one of the departments of Ericsson in the Netherlands. We have shown how the measurement program of Ericsson contains activities from each of the four steps of the measurement-based improvement process model. In section 7.3, we have used the process model to assess and compare different guidelines for implementing measurement programs from the literature, amongst which our Measurement Capability Maturity Model that was introduced in chapter 6.

From the comparison we draw three conclusions:

- there is quite some consensus on the basic activities needed to successfully implement measurement programs; but,
- at the same time, different frameworks emphasize widely different aspects of measurement program implementation.

In addition, the assessment also reveals that:

- there is almost no consensus on, nor description of, activities needed to successfully *use* the results from measurement programs.

All the examined guidelines for implementing measurement programs focus on the internals of measurement programs. In section 7.4, we have argued that

to guarantee the success of measurement programs, one should also take external factors into account. These external factors are aimed at making sure that the measurement program generates value for the organization. By discussing different uses of measurement programs we have identified four external success factors of measurement programs:

1. The various assumptions underlying the measurement program should be made explicit. It should be decided if and when these assumptions are tested.

2. Different outcomes can result from a measurement program. An organization should consider all possible – negative and positive – outcomes and decide how to act on them.

3. The organization should act according to the outcomes of the measurement program, in order to reach the goals set or solve the problems identified.

4. The organization should monitor the changes implemented, in order to verify that these changes indeed constitute an improvement for the organization.

Our external success factors complement the success factors such as presented by Hall and Fenton. Together, these success factors cover all four phases of the measurement-based improvement process model as presented in section 7.1. In addition, we have shown how an organization could adhere to the external success factors by explicitly addressing these issues before designing and implementing a measurement program.

Part II

The Maturity Perspective

As explained in chapter 2 we distinguish between improvement methodologies based on internal references and on external references. In part I we have examined the usage of the former in software maintenance environments, in this part we investigate the latter.

A large class of external reference-based improvement methods is formed by maturity models, of which the Software Capability Maturity Model is the best known. These maturity models contain processes which are claimed to be *key* for software organizations to become more mature with respect to software development. Most of these maturity models also claim to cover software maintenance.

However, if we look at software development and maintenance from a service perspective we see that software maintenance has more service-like aspects than software development. This has a number of consequences for the way in which customers may determine the quality of software maintenance. This also means that the processes which can be considered key for mature software maintenance organizations are different than the key processes for software development organizations. We have developed a maturity model which contains processes that we consider key for mature IT service providers. We have chosen the scope of the maturity model to be IT services in general because software maintenance and other IT services, such as system operations, user support, infrastructure maintenance, are quite similar with respect to the key processes needed.

Chapter 8 explains the differences between services and products in general and then looks at the differences between software maintenance and development from a service perspective. Chapter 9 presents the IT Service Capability Maturity Model. Finally, chapter 10 reports on two pilot assessment studies done using the IT Service CMM.

Chapter 8

Software Maintenance from a Service Perspective

In this chapter we investigate the differences between software maintenance and software development from a service point of view. We show that there are differences between products and services in general. These differences affect the way in which customers assess the quality of products and services. In particular, service quality is assessed on two dimensions: the technical quality – what the result of the service is – and the functional quality – how the service is delivered.

We argue that software maintenance can be seen as providing a service, whereas software development is concerned with the development of products. Consequently, customers will judge the quality of software maintenance different from that of software development. This means that to deliver high quality results, both the functional quality and the technical quality dimension is important for software maintenance. To provide high-quality software maintenance, different and additional processes are needed than provided by the Software Capability Maturity Model.

During the Concrete Kit and Kwintes projects, a number of case studies have been undertaken to test and evaluate methodologies to improve IT services. Examples are the service level agreement specification method, the use of standard service level agreements, evaluation of service quality, etc. These case studies had mixed results. We observed that some service providers were more mature as regards their service capabilities than others. These practical experiences gave additional indications that IT service providers need different processes than software development organizations.

We have captured the issues arising from the service viewpoint and from the experiences with the case studies in a maturity model targeted at organizations

that provide information technology services. This IT Service Capability Maturity Model (IT Service CMM) is described in chapter 9.

This chapter is structured as follows: in section 8.1 we discuss a number of differences between services and products in general, and between software maintenance and software development in particular. In section 8.2, we show how these differences affect organizations that maintain software. In particular, we argue that there are a number of processes that should be considered *key* to becoming a mature software maintenance organization, but are not part of the much used Software Capability Maturity Model. Next, in section 8.3 we discuss a number of case studies done during the Concrete Kit and Kwintes projects. In section 8.4 we further elaborate on the key processes for software maintenance organizations, based on the case studies presented in section 8.3. Section 8.5 discusses related work on factors that influence the quality of software maintenance. Finally, section 8.6 presents our conclusions.

8.1 Services versus products

In the service marketing literature, a wide range of definitions exists of what a service entails, see Grönroos (1990) for a list of examples. Usually, a service is defined as an essentially intangible set of benefits or activities that are sold by one party to another. The main differences between products and services are (Zeithaml and Bitner 1996):

Intangibility This is considered to be the most basic difference between products and services. Services – being benefits or activities – cannot be seen, felt, tasted, or touched, like products can. Consequently,

- services cannot be inventoried,
- services cannot be patented,
- services cannot be readily displayed or communicated, and
- pricing is more difficult.

Heterogeneity Because services are created by activities, and activities are performed by humans, services tend to be more heterogeneous than products. Consequently,

- service delivery and customer satisfaction depend on employee actions,
- service quality depends on factors which are difficult to control, such as the ability of the customer to articulate his or her needs, the ability and

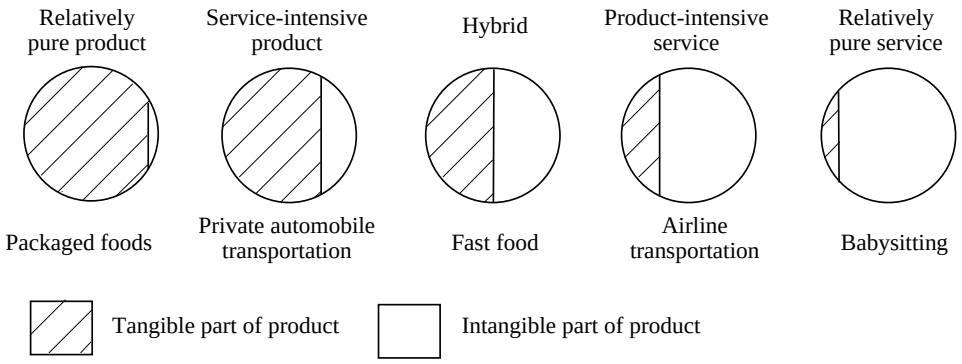


Figure 8.1: The product-service continuum (Berry and Parasuraman 1991)

willingness of personnel to satisfy those needs, the presence or absence of other customers, and the level of demand for the service, and

- these complicating factors make it hard to know whether the service was delivered according to plan or specifications.

Simultaneous Production and Consumption Services are always produced and consumed simultaneously, whereas for products production and consumption can be separated. For example, a car can be produced first, sold a few months later, and then be consumed over a period of several years. For services on the other hand, the production and consumption has to take place in parallel. The production of the service creates the 'set of benefits', whose consumption cannot be postponed. For example, a restaurant service – preparing a meal, serving the customer – has largely to be produced while the customer is receiving the service. Consequently,

- customers participate in and affect the transaction,
- customers may affect each other,
- employees affect the service outcome, and
- centralization and mass production are difficult.

Perishability Services cannot be saved or stored. Consequently,

- it is difficult to synchronize supply and demand with services, and
- services cannot be returned or resold.

The difference between products and services is not clear-cut. Often, services are augmented with physical products to make them more tangible, for example,

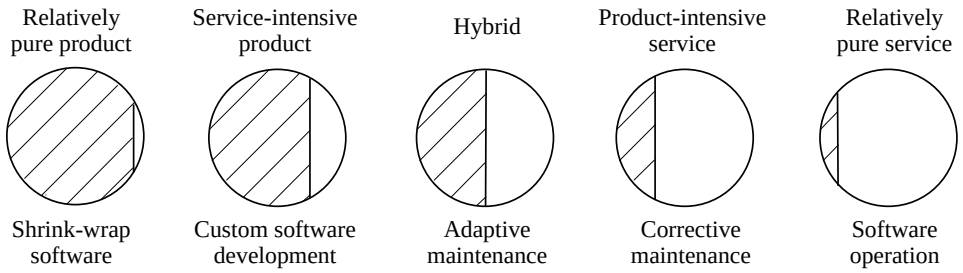


Figure 8.2: The product-service continuum for software development and maintenance

luggage tags provided with a travel insurance. In the same way, products are augmented with add-on services, for example a guarantee, to improve the quality perception of the buyer. In the service marketing literature (e.g. Berry and Parasuraman 1991), a product-service continuum is used to show that there is no clear boundary between products and services. This product-service spectrum is a continuous dimension with pure products on one end and pure services on the other end, and product-service mixtures in between. Figure 8.1 shows some example products and services, positioned on the product-service continuum.

As figure 8.1 shows, products and services can be intertwined. In the case of fast-food, both the product – the food itself – and the service – fast delivery – are essential to the customer. This means that the quality of such a product-service mix will be judged on both product and service aspects: is the food quickly served, and does it taste well.

If we turn to the software engineering domain, we see that a major difference between software development and software maintenance is the fact that software development results in a *product*, whereas software maintenance results in a *service* being delivered to the customer. Software maintenance has more service-like aspects than software development, because the value of software maintenance is in the activities that result in benefits for the customers, such as corrected faults and new features. Contrast this with software development, where the development activities do not provide benefits for the customer directly, but it is the resulting software system that provides the benefits.

As said above, the difference between products and services is not clear-cut. Consequently, this goes for software development and software maintenance as well. Figure 8.2 shows the product-service continuum, as displayed in figure 8.1, but with examples from the software engineering domain.

8.2 Service quality

Though we argued in the previous section that we can view software maintenance as a service and software development as product development, we did not yet mention why this would be beneficiary. In order to do so, we again turn to the literature in the area of service management and marketing.

Grönroos (1990) states that there are two dimensions that determine the experienced quality of services:

- The technical quality of the outcome. This dimension is formed by the result of the service, *what* the customer is left with when the service has been delivered.
- The functional quality of the process. This dimension is determined by the way in which the customer receives the service, in other words *how* the service is delivered.

So, both technical and functional quality determine how a customer perceives the service. Service marketeers often use the gap model to illustrate how differences between perceived service delivery and expected service can come about, see figure 8.3. The difference between the perceived quality of the services and the expected quality (gap 5) is caused by four other gaps (Zeithaml and Bitner 1996):

Gap 1 The expected service as perceived by the company differs from the service as expected by the customer.

Due to inadequate market research, lack of communication between contact employees and management, and insufficient relationship focus, the service provider has a perception of what the customer expects which differs from the real expected service.

For example, the service organization aims to satisfy certain availability constraints (e.g. 99.5% availability), while the actual customer concern is with maximum downtime (e.g. no longer than one hour per failure).

Gap 2 The service specification differs from the expected service as perceived by the company.

Caused by a lack of customer-driven standards, absence of process management, lack of a formal process for setting service quality goals, poor service design and inadequate service leadership, the service designs and standards will not match the service requirements as perceived by the company.

For example, the customer expects a quick restart of the system, while the standard procedure of the maintenance organization is focused on analyzing the reason for the crash.

Gap 3 The actual service delivery differs from the specified services.

Service delivery does not follow the service designs and standards because of deficiencies in human resource policies, failures to match demand and supply, and customers not fulfilling their role.

For example, customers bypass the helpdesk by phoning the maintainer of their system directly, and thus hinder a proper incident management process.

Gap 4 Communication about the service does not match the actual service delivery.

Communication by the service provider about its delivered services does not match the actual service delivery because of ineffective management of customer expectations, overpromising, and inadequate horizontal communications (i.e. insufficient communication between sales and operations and between advertising and operations and differences in policies and procedures across the organization).

For example, a customer is not informed about the repair of a bug he or she reported.

The fifth gap is caused by the four preceding gaps. Hence, perceived service quality can be increased by closing the first four gaps, thus bringing the perceived service in line with the expected service.

To summarize so far, we see that the quality of services is determined by two quality dimensions: the technical quality – what is the result – and the functional quality – how is the result reached. We also showed how the gap between the perceived service delivery and expected service delivery is caused by several other gaps in the service provider's organization.

The question is, how does this all translate to the area of software engineering? Our argument is that since software maintenance organizations are essentially service providers, they need to consider the issues mentioned in this section. They need to manage their product – software maintenance – as a service to be able to deliver high quality software maintenance.

Looking at the gap model presented in figure 8.3, we notice a number of processes emerge which pertain to the quality of the delivered services. To close the gaps a service provider needs to:

- Translate customer service expectations into clear service agreements (Gap 1).
- Use the service agreements as a basis for planning and implementing the service delivery (Gap 2).

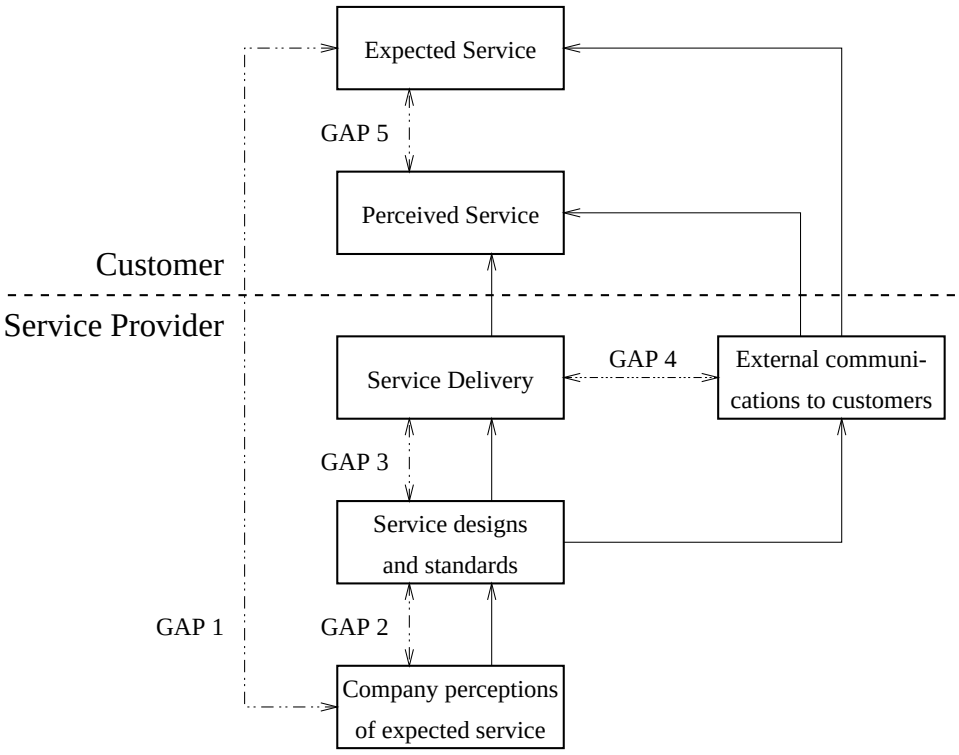


Figure 8.3: Gaps model of service quality (Zeithaml and Bitner 1996)

- Ensure that service delivery is done according to planning and procedures (Gap 3).
- Manage communication about the services delivered (Gap 4).

The next section discusses a number of case studies done in the course of the Concrete Kit and Kwintes research projects which provide more insight in the activities software maintenance organizations, or other IT service providers, could use to implement the four processes listed above.

8.3 Case studies

As mentioned in chapter 1, we use a generic process model (repeated in figure 8.4) as the basis for our research. Guided by the lemniscate, different research issues have been identified, including the specification of service level agreements, eval-

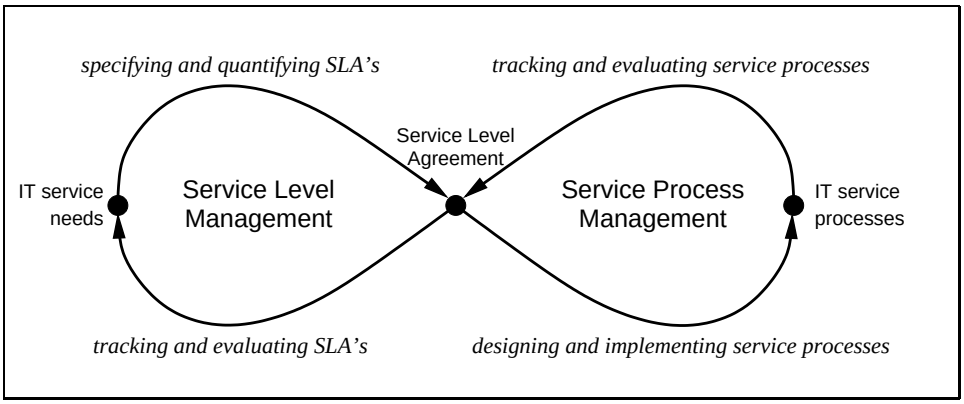


Figure 8.4: Service Level Management lemniscate

uation of service quality, the use of service catalogs and problem management. These issues have been investigated in several case studies that are presented below. The case studies demonstrate the occurrence of the gaps identified in the previous section.

To facilitate the translation of diffuse IT needs of customers into measurable service level agreements (upper-left arrow of the lemniscate) a SLA specification method was developed. Several case studies were performed to evaluate and improve the method. Sections 8.3.1 and 8.3.2 describe two of them.

An important question with respect to service level agreements is whether the right levels have been established. In one case study we investigated the use of ServQual to evaluate service level agreements (lower-left arrow). This case is described in section 8.3.3.

According to ITIL, problem management is an important aspect of service provision. The ITIL Problem Management process aims at minimizing the impact of failures ('incidents') and on correcting root causes of failure. This makes it part of both the upper-right and the lower-right arrow of the IT service lemniscate. In the case study described in section 8.3.4 we looked at the problem management process.

As mentioned in section 2.2.2, ITIL advocates the use of a service catalog but does not provide directions on how to implement it. Therefore, we did two case studies on the development of service catalogs, described in sections 8.3.5 and 8.3.6.

8.3.1 Case A – developing a service level agreement

This case study was part of an education improvement program undertaken by a Dutch university. Part of the program is the supply of notebooks to all students, including different services such as end-user support and repair maintenance. The notebooks and services are delivered by a large Dutch IT service provider.

During the case study a service level agreement between the service provider and the university was developed. The service level agreement (SLA) specification method (Bouman, Trienekens and van der Zwan 1999) was used to derive the needed service levels, taking the students – the end-users – as the starting point. This was the first time that this service provider used the SLA specification method to develop service level agreements, and it was also the first time they delivered this particular service. Despite the lack of experience, the service level agreement was developed according to the method without major problems.

8.3.2 Case B – developing a generic service level agreement

This case study was held in the information technology department of a large Dutch governmental organization. The study was part of a larger program to implement a quality system in the organization. The case study concerned the introduction of the SLA specification method and the development of a generic service level agreement.

This organization had a quite formal organizational structure, but at the same time this formal structure was being ignored to be able to react to organizational and technical problems. The organization was not used to draw up service level agreements with its customers. Agreements between the department and its customers were in the form of effort obligations, not results. No quality methodologies, such as ITIL or ISO 9000, were being used.

It seemed to us that this organization was not quite ready for the introduction of generic service level agreements, without first gaining practical experience with the use of result oriented contracts.

8.3.3 Case C – evaluating service quality

During this case study, the quality of the services delivered by the IT department of a decentralized governmental organization was evaluated. We used ServQual (Zeithaml, Parasuraman and Berry 1990) to measure the perceived quality of the IT services by the end-users. ServQual is a measurement method targeted at measuring the quality of services. See (Pitt, Watson and Kavan 1995, Watson, Pitt and Kavan 1998) for examples of the application of ServQual in measuring IT service quality.

The IT department manages and maintains the IT infrastructure of the local governmental organization and provides end-user support. The department does use service level agreements, but these are mainly used to specify procedures and opening times, and do not address concrete and measurable service levels.

The case study was quite successful: users of the IT services were very well capable of detailing their opinion on the quality of the service provision. Apparently, the evaluation of service quality does not depend on the presence of specified quality levels.

8.3.4 Case D – incident and problem management

This organization is the IT department of a large organization, responsible for carrying out part of the Dutch social security system. As of the beginning of 1996, the organization has been split into a non-profit public body and a private for-profit organization – part of which is the IT department.

The IT department provides a large number of IT services to its customers, which are mainly departments from the sibling organization. To manage the communication with customers regarding those services, the department has implemented helpdesk management and problem management processes. The implementation of these processes has been based on ITIL. Helpdesk Management is used to guarantee the continuity of services, while Problem Management is used to improve the level of service in the future. So, Helpdesk Management deals with *incidents*, whereas Problem Management is concerned with solving the *problems* that cause these incidents.

The goal of this case study was to assess the quality of the Problem Management process. It soon became apparent that the organization was not able to execute the Problem Management process properly, because the Help Desk Management process did not result in the necessary data needed to adequately analyze and solve problems. For example, many incidents were not classified in the right incident category, or not classified at all. This resulted in a low validity of the incident database: it was estimated that more than 30% of the incidents were coded incorrectly.

It was found necessary to first implement a clear and consistent registration of the incidents that occur during service delivery, before attempting to improve the problem management process.

8.3.5 Case E – developing a service catalog

This case study was done in the central IT department of a large Dutch governmental organization. The IT department develops, operates, and maintains hardware

and software for the decentralized governmental organization. The goal of the case study was to investigate the possibility to use a service catalog to improve communication between the IT department and its customers. The purpose of the service catalog would be to facilitate the negotiation of service levels by providing a set of services combined with standard service levels that the IT department is able to provide, together with standard prices.

When the case study started the IT department had already developed a document that was supposed to be a service catalog. However, closer investigation showed that this document did not contain the information necessary to negotiate service levels: it hardly contained any quantitative data and no indications of costs of services. Further research showed that the organization did not only omit this information from the service catalog, but also that it did not have the necessary data available. This made it impossible to implement a full scale service catalog during the time-span of the case study.

8.3.6 Case F – developing a service catalog

This case study was done with an IT organization that delivers a wide spectrum of IT services, ranging from PC installation to system management. The organization uses ITIL to implement its service management processes. The organization has been split in a number of business units that work together to deliver integrated services. The organization has been using result-oriented service level agreements for a number of years and generally looks like an IT service provider that has become of age.

The goal of this case study was to implement part of a service catalog. The organization felt that a service catalog would be a good step towards their goal of quality improvement. The case study had the full commitment of both management and employees and resulted in a prototype service catalog that was used in the negotiations with a large customer.

8.3.7 Lessons learned

Although the six case studies discussed cover a wide range of issues and different organizations, we feel that several important lessons can be learned from these case studies. The most important lesson is that IT service improvement can only be successful if the organizational preconditions have been fulfilled.

The case studies were conceived as experiments to test service improvement techniques and methodology developed in the course of our projects. The success, or lack of success, of some of these case studies can be easily interpreted in terms of the gap model. In particular:

- Gaps 1 and 2 were successfully bridged in case A. The SLA specification method is specifically aimed at guiding the dialogue between customer and service provider as to the contents of the service. The resulting service level agreement provided a solid basis for planning and implementing the service. The case study has not been carried on long enough to observe the actual service delivery.

- In case B, these same gaps were not successfully bridged. In particular, the jump from nothing to a generic service level agreement was not successful.

- In case C the quality of the service as perceived by the customers was measured. This case studies suggests that it is very well possible to determine the size of gap 5, independently of the actual quality of the IT services and the IT service provider.

- Gap 4 was very visible in case D. Incident management was not done consistently, and this caused problems in the (internal as well as external) communication about the service delivered.

- The IT department in case E wanted to develop a service catalog, containing the services provided by the organization, including service levels and the costs of different services. However, the organization did not have sufficient (quantitative) insight into its own processes to develop a full-fledged service catalog. Thus gap 3 was not closed.

- In case F the organization successfully developed and used a service catalog. The service catalog was based on the experience of the company with service level agreements in the past. Hence, this organization had enough insight into its own service delivery to successfully bridge gap 3.

Several of our case studies were rather unsuccessful, mainly because the organization was not 'ready' for the new methodology introduced. The reason that an organization is not ready can be caused by cultural issues, but also by the lack of certain practices that are needed for the improvement. For example, the lack of historical data on services provided makes it impossible for the IT department from case E to develop a full fledged service catalog. Another example is the problem management process of the IT department in case D which cannot be properly executed due to the low quality of the incident database. Apparently, some practices need other practices to be implemented, before they can be carried out properly:

- problem management needs consistent incident management;

- implementation of a service catalog needs historic information on service level agreements and performance.

On the other hand, there are a number of case studies that were successful, despite the apparent low maturity of the organizations. For example, the ServQual evaluations of the service delivered by the case C organization were successful despite the lack of measurable service level agreements. Another example is the successful use of a service level agreement between customer and service provider in case A, despite the fact that this is the first time the IT organization provides this particular service. Apparently, practices such as service evaluation and service specification and reporting can be introduced in any IT service organization.

8.4 Bridging the gaps

In this section, we discuss four processes that may help bridge the gaps identified in section 8.2. These processes were derived from the gaps, the case studies presented in the previous section, discussions with maintenance managers, and a critical look at the Software CMM from a maintenance perspective.

8.4.1 Gap 1: management of commitments

It is important that maintenance commitments be planned and documented. This works best if the maintenance organization and customer work together towards the specification of relevant and realistic maintenance service commitments (often called a Service Level Agreement — SLA), based on the needs of the customer. The actual maintenance services delivered, the specified service levels and the customer's service needs are reviewed with the customer on a regular basis. As a result of this evaluation, the service level agreement may have to be adjusted to stay in line with possibly changing maintenance needs.

There are two basic issues involved here: first, the maintenance service to be delivered is specified in a contract – the service level agreement – containing *measurable* service levels. Second, the service levels specified should address the business needs of the customer.

The service level agreement documents the maintenance services to be delivered. It covers the purpose, scope and goals of the services, their specification, and other agreements. The service level agreement functions as a means to close gap 1 by setting expectations for the maintenance service. It should at a minimum specify:

1. the maintenance services itself, i.e. a specification of the services to be delivered;

2. with what levels of service, i.e. how fast, how reliable, etc., specified in a measurable manner. Service levels need to be measurable because the organization has to report the realized service levels.

3. the conditions the customer should obey. Examples of such conditions could be that the customer should use a certain format for documenting change requests or, in case of a bug, provide the maintenance department with the input that caused the fault to manifest itself.

4. what happens if the maintenance organization does not reach the agreed upon service levels while the customer did not violate the customer conditions.

5. when and what will be reported to the customer regarding the actual delivered maintenance services.

6. when and how the service level agreement will be reviewed.

7. under which circumstances (calamities) service is not guaranteed.

The commitments as documented in the service level agreement should be derived from the maintenance needs of the customer (as opposed to just the capabilities of the maintenance organization). These maintenance needs should be related to the business processes of the customer, its information technology, its business strategy, etc. This ensures that the maintenance organization thinks about what the customer needs and thus helps to close gap 1.

8.4.2 Gap 2: maintenance planning

The maintenance activities as specified in the service level agreement have to be planned. This includes the planning of the maintenance activities themselves, the transfer of the results thereof to the customer, the estimation of resources needed, the scheduling of maintenance activities, and the identification of possible risks.

In a normal, non-emergency situation, changes are often bundled into releases. There are various ways of deciding on the contents and timing of the next release. For example, releases may be scheduled at fixed time intervals, while there also is a fixed number of people available for doing maintenance. The next release will then contain all changes which could be handled within that time frame. One may also negotiate and fix the contents of the next release in advance, and allocate the number of people accordingly. This planning of releases is part of the maintenance planning process. Stark and Oman (1997) describe an observation of several strategies applied in practice.

An important characteristic of software maintenance is that the maintenance activities are often *event*-driven. The submission of change requests and bug reports drives the maintenance work. Hence, an important aspect of the planning of maintenance is the estimation of the event-driven maintenance workload, i.e. the number of change requests and bug reports expected and the effort needed to process them.

Explicitly basing the planning of maintenance activities on the commitments as agreed upon with the customer helps to close gap 2.

8.4.3 Gap 3: maintenance activity tracking

The service level agreement states which maintenance activities are to be carried out, and how fast, reliable, etc. this should be done. In order to be able to report on the performance of the maintenance organization in this respect, information about the actual maintenance activities is to be gathered. The purpose of the maintenance activity tracking process is to provide this information, monitor maintenance activities, and take corrective actions if necessary.

For example, when the customer reports a bug, information about the bug itself (originator, type, etc.) is recorded, as well as the reporting time, the time when corrective action was started and ended, and the time when the bug was reported fixed. If these data indicate that the average time to fix bugs exceeds the level as specified in the service level agreement, the maintenance organization might assign more maintenance staff to this system, put maintenance staff on point-duty at the customer site, renegotiate the agreed upon service level, or take any other action to realign agreement and reality.

By keeping a strict eye upon the performance of the maintenance organization, and adjusting the maintenance planning and/or renegotiating the commitments with the customer when required, gap 3 is narrowed.

8.4.4 Gap 4: event management

Event management concerns the management of events that cause or might cause the maintenance activities carried out to deviate from the agreed upon levels of maintenance service. Events can be either:

- Requests for changes from users or other stakeholders. For example, requests for a new feature in the software;

- Incidents that cause or will cause service levels to be lower than agreed upon if no action is being taken. For example, a server that is down might cause the

specified maximum down-time to be exceeded if it is not restarted quickly enough.

The main purpose of event management is to manage all events that occur during software maintenance. Event management encompasses a number of activities that should ensure that incidents and change requests are resolved in time and that affected groups, including the customer, are kept informed. These activities thus contribute to both the functional as well as the technical quality of software maintenance. A subset of possible event management activities is:

- An event management library system is established as a repository for the event records.

  This event management library system (often in the form of a 'helpdesk system') should provide for the storage, update, and retrieval of event records, the sharing and transfer of event records between affected groups, and should help in the use of event management procedures.

  This supports the communication with the customer about the maintenance services delivered. It also supports the maintenance department itself, in its role of a historical data base of changes. The event management system thus helps to close gap 4.

- Events are identified, recorded, reviewed, and tracked according to a documented procedure.

  Each event is recorded in the library system, the impact of the event is assessed and documented, and 'action items' are formulated and initiated to resolve the event.

  This activity reinforces that the maintenance activities carried out are kept in accordance with the maintenance commitments and the maintenance planning, thus helping to close gap 3.

- Standard reports documenting the event management activities and the contents of the event repository are developed and made available to affected groups and individuals.

  This activity helps keeping the customer informed about the progress of activities to resolve incidents or process change requests. The communication with the customer not only pertains to individual change requests, but also their bundling into releases. There thus is a relation with the maintenance planning process. Keeping the customer informed will help manage customer expectations, again narrowing gap 4.

8.5 Related work

In his book *Practical Software Maintenance,* Thomas Pigoski laments that software maintenance organizations need to realize that they are in the customer service business (Pigoski 1996, pp. 171-172). Apparently, this is not widely recognized yet. Within the software engineering domain, including software maintenance, the focus is on product aspects. The final phases of software development supposedly concern the delivery of an operations manual, installing the software, handling change requests and fixing bugs. In practice, the role of the information systems department is much broader during the deployment stage, as illustrated by the ubiquitous help desk.

Published evaluations of software maintenance practices tend to concentrate on the narrow issue of efficiently handling change requests and bug fixes (e.g. Briand, Kim, Melo, Seaman and Basili 1998, Onoma, Tsai, Tsunoda, Suganuma and Subramanian 1995, Singer 1998, West 1996). For example, a common denominator in these papers is the emphasis that is placed on a presence of what is termed a bug tracking system, historical data base of changes, or change management. The wording is such that the *internal* use of this information gets emphasized. The information is considered important for the maintainers: they must be able to track similar bugs, they must be able to retrieve the status of each change, and so on. By taking a service perspective, we additionally stress the *external* use, i.e. in the communication with the customer, of essentially the same information in what we call event management.

Stålhane, Borgersen and Arnesen (1997) did a survey to find those aspects of quality that customers consider most important. The most important result of their study is the strong emphasis customers place on service quality. The top five factors found in their study are: service responsiveness, service capacity, product reliability, service efficiency, and product functionality. They also quote an interesting result from a quality study in the telecommunications domain. On the question 'Would you recommend others to buy from this company?', a 100% yes was obtained for the category users that had complained and got a satisfactory result. For the category users that had not complained, this percentage was 87%. Apparently, it is more important to customers to get a satisfactory service than to have no problems at all.

Finally, Pitt et al. also argue that software maintenance has a significant service component. They have used ServQual as a measure of IS service quality (Pitt, Watson and Kavan 1995, Watson, Pitt and Kavan 1998).

8.6 Conclusions

We have shown in this chapter that the quality of services is determined by two quality dimensions: the technical quality – what is the result – and the functional quality – how is the result reached. We also showed how the gap between the perceived service delivery and expected service delivery is caused by several other gaps in the service provider's organization.

The question is, how does this all translate to the area of software engineering? Our argument is that since software maintenance organizations are essentially service providers, they need to consider the issues mentioned in this section. They need to manage their product – software maintenance – as a service to be able to deliver high quality software maintenance.

We have presented six case studies from which we have deduced a number of processes which pertain to the quality of the delivered services. To close the gaps a software maintenance organization needs:

- *Management of commitments*: translate customer service expectations with respect to software maintenance into clear service agreements (Gap 1).
- *Maintenance planning*: use the service agreements as a basis for planning and implementing the maintenance activities (Gap 2).
- *Maintenance activity tracking*: ensure that maintenance is done according to planning and procedures (Gap 3).
- *Event management*: manage communication about the maintenance activities carried out (Gap 4).

We have elaborated on each of these four processes to indicate how they could help software maintenance organizations to narrow the four gaps.

If we compare the processes as identified above with the key processes in the Software Capability Maturity Model (see table 2.4 on page 25), we observe that the processes identified in this chapter are not present in the Software CMM. Most notably:

- In the Software CMM, the planning of the software projects is based on the software requirements, which are derived from the system requirements. Requirements elicitation is not part of the requirements management key process area. Our process 'management of commitments' does include the 'elicitation' of the maintenance needs of the customer organization.
- The Software CMM uses the software requirements as basis for the planning of the software project. If the requirements change, the project planning

needs to be changed. However, in a maintenance environment the activities are often not *requirement*-driven, but *event*-driven: bug reports and feature requests initiate the work. Our process 'maintenance planning' takes this into account.

- In the Software CMM, tracking is aimed at monitoring project progress. The tracking is done against the software development plan. Because software maintenance is often event-driven, we need to monitor the number of events next to the progress of the activities.
- The Software CMM does not contain processes that deal with the management of communication between the software organization and the customer.

In the next chapter we present the IT Service Capability Maturity Model that aims to capture the lessons learned in this chapter in a maturity model. Although this chapter was primarily concerned with software maintenance, we have chosen to develop a maturity model not exclusively aimed at software maintenance, but at all types of IT services. One reason is that the research questions with respect to maturity-based improvement were targeted at IT services in general. The second reason is that we felt that the four processes identified in this chapter can be fairly easily applied to other IT services beside software maintenance.

Chapter 9

IT Service Maturity

The IT Service Capability Model is a maturity growth model akin to the Software Capability Maturity Model (SEI 1995). The structure of the model is similar to that of the Software CMM, but its application domain is different. Whereas the Software CMM targets software development processes[1], the IT Service CMM targets the processes that we consider key to producing high quality IT services. IT services are provided by operating, managing, installing, or maintaining the information technology of a customer or supporting the users of that technology. So, software maintenance is one of the possible IT services that can be provided.

In the next section, we discuss the primary objectives of the IT Service CMM. Section 9.2 explains the structure of the model. In section 9.3, we present the five maturity levels of the model and in section 9.4, we give an overview of the key process areas of the model. Section 9.5 presents two of the level two key process areas in more detail. Finally, section 9.6 presents our conclusions.

Note that the IT Service CMM uses the same structure and wording as the Software CMM as described in SEI (1995). Consequently, the text in this chapter is more formal than in other chapters in this thesis, most notably the specification of the key practices in section 9.5.

What we describe in this chapter is the version of the IT Service CMM as developed during the Kwintes project (Niessink and van Vliet 1999*c*). This version describes all five levels of the IT Service CMM and the key processes at each level. However, only the level two key processes have been detailed to the level of individual key practices. The key practices of higher level key process areas remain to be specified.

[1]The Software CMM is claimed to be suited for both development and maintenance processes, but difficulties implementing the model in a maintenance-only organization were reported by Drew (1992).

9.1 Primary objectives of the IT Service CMM

The objective of the IT Service CMM is twofold:

1. to enable IT service providers to assess their capabilities with respect to the delivery of IT services, and,

2. to provide IT service providers with directions and steps for further improvement of their service capability.

The IT Service CMM fulfills these goals by measuring the capability of the IT service processes of organizations on a five level ordinal scale. Each level prescribes certain key processes that have to be in place before an organization resides on that level.

More formally, we define *IT service process capability* as the range of expected results that can be achieved by following a service process. *IT service process performance* represents the actual results achieved by following an IT service process. *IT service process maturity* is the extent to which a specific process is explicitly defined, managed, measured, controlled and effective. The IT Service CMM focuses on measuring and improving the IT service process maturity of IT service organizations.

An organization that scores high on the IT Service CMM scale should be able to:

- deliver quality IT services, tailored to the needs of its customers;

- do so in a predictable, cost-effective way;

- combine and integrate different services, possibly delivered by different service providers, into a consistent service package;

- continually improve service quality in a customer-focused way.

9.2 The structure of the IT Service CMM

The IT Service CMM is based on the Software CMM. Where applicable, the descriptions of the IT Service CMM maturity levels and key process areas are adjusted from SEI (1995). The structure of the Software CMM and the IT Service CMM are largely the same, see figure 9.1. The model consists of five maturity levels, which contain key process areas. For an organization to reside on a certain maturity level, it needs to implement all of the key processes for that level, and lower levels.

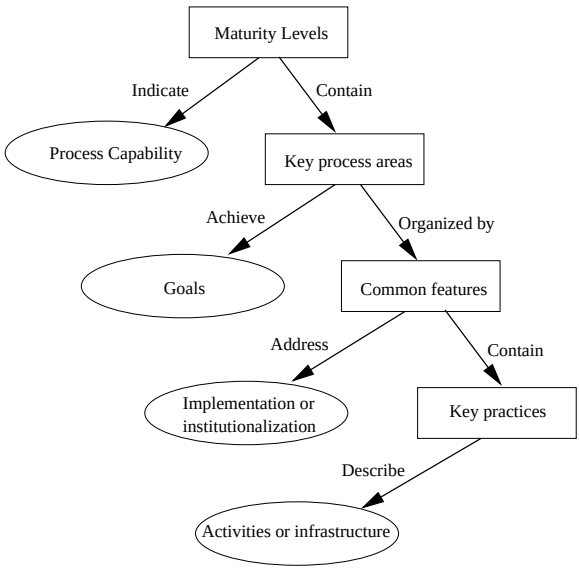


Figure 9.1: The CMM structure (SEI 1995)

Each key process area is structured using common features. Common features are practices that, when performed together, guarantee that the key process area is implemented and institutionalized. Common features consist of key practices that describe activities that have to be performed or infrastructures that have to be present.

9.3 The maturity levels of the IT Service CMM

We measure the service process maturity of organizations on a five level ordinal scale. The first – initial – level has no associated key process areas. This is the level where all IT service organizations reside that have not implemented the level two key process areas. Level two is the repeatable level. Organizations that have reached level two will be able to repeat earlier successes in similar circumstances. Thus the emphasis of level two is on getting the IT services right for one customer. On level three, the defined level, the service organization has defined its processes and is using tailored versions of these standard processes to deliver the services. By using common organization-wide standard processes, the process capability to deliver services consistently is improved. At level four, the managed level, organizations gain quantitative insight into their service processes and service quality.

By using measurements and an organization-wide measurement database organizations are able to set and achieve quantitative quality goals. Finally, at level five, the optimizing level, the entire organization is focused on continuous process and service improvement. Using the quantitative measurements the organization prevents problems from recurring by changing the processes. The organization is able to introduce new technologies and services into the organization in an orderly manner.

More formally, we define the five maturity levels as follows:

Initial level The IT service delivery process is characterized as ad hoc, and occasionally even chaotic. Few processes are defined, and success depends on individual effort and heroics.

Repeatable level Basic service management processes are established to track cost, schedule and performance of the IT service delivery. The necessary discipline is in place to repeat earlier successes on projects with similar services and service levels.

Defined level The IT service processes are documented, standardized, and integrated into standard service processes. IT services are delivered using approved, tailored versions of the organization's standard service processes.

Managed level Detailed measurements of the IT service delivery process and service quality are collected. Both the service processes and the delivered services are quantitatively understood and controlled.

Optimizing level Continuous process improvement is enabled by quantitative feedback from the processes and from piloting innovative ideas and technologies.

9.4 The key process areas of the IT Service CMM

For an organization to reside on a certain maturity level, it needs to implement all key processes for that maturity level – and those for lower levels. The term *key* process area merely means that these processes are seen as the key to reach a certain maturity level. There might be more – non-key – processes, but these are not strictly necessary to reach the next maturity level.

Table 9.1 gives an overview of the key process areas. The key process areas are grouped into three process categories: management, enabling and delivery. The first group is concerned with the management of services. The second category deals with enabling the delivery process by means of support processes and standardization of processes. The third category consists of the processes that result

Process categories **Levels**	**Management** Service planning, management, etc.	**Enabling** Support and standardization.	**Delivery** Actual service delivery.
Optimizing	Process Change Management	Technology Change Management	Problem Prevention
Managed	Quantitative Process Management		Service Quality Management
Defined	Integrated Service Management	Organization Process Focus Organization Process Definition Training Program	Service Delivery
Repeatable	Service Commitment Management Service Delivery Planning Service Tracking and Oversight Subcontract Management	Configuration Management Event Management Service Quality Assurance	
Initial	Ad hoc processes		

Table 9.1: Key process areas, assigned to process categories

in the consistent, efficient delivery of services according to the appropriate quality levels. Below we present the key process areas for each of the maturity levels of the IT Service CMM.

9.4.1 Level 1: Initial

There are no key process areas prescribed for level one.

9.4.2 Level 2: Repeatable

The key process areas for level two are concerned with establishing the processes that enable the organization to repeat earlier successful services in similar situations. We distinguish between two kinds of processes that an organization has to implement on this level. The first category deals with service management: the planning, specification, tracking and evaluation of services. The second category is concerned with service support: processes that support the activities that actually

deliver the services.

The management processes on this level look as follows. First, the service provider and the customer draw up an agreement about the services to be delivered, the quality of the services – specified in terms of service levels – and the costs of the services (Service Commitment Management). To ensure that the service levels are realistic, the service provider draws up a service plan that shows the feasibility of the service levels (Service Delivery Planning). During service delivery, the service provider tracks the realized service levels and reports these to the customer on a regular basis to demonstrate that the provider has indeed delivered the services against the promised service levels (Service Tracking and Oversight). After a period of service provision, the customer and the service provider review the service level agreement to see whether it still conforms to the IT needs of the customer (Service Commitment Management). Just like the organization draws up a service level agreement with its customer, the organization should also use service level agreements when it delegates parts of the service delivery to third parties (Subcontract Management).

We identify three support processes that a level two organization needs to implement. First, almost all IT services concern the management, operation or maintenance of hardware and software components. Therefore, where necessary for consistent service delivery, these components are put under configuration control. This ensures that at all times the status and history of these components is known, and that changes are controlled (Configuration Management). Second, during the period that the services are delivered, events can occur that need to be resolved by the service provider. These events range from simple requests for service to serious incidents that prevent the customer from using its information technology. All these events need to be identified, tracked, resolved and reported to the customer (Event Management). To service the request and to resolve incidents, changes to the configuration may be necessary. The change requests are evaluated by the configuration control board with respect to the service level agreement and risk for the integrity of the configuration. Only after a change request has been approved by the configuration control board, will the configuration be changed (Configuration Management). Finally, to ensure the quality of the services, the service provider deploys quality assurance techniques, such as reviews and audits (Service Quality Assurance).

Next follows a description of the level two key process areas:

1. Service Commitment Management:
   Purpose: Services are specified and realistic service levels are negotiated with the customer in order to deliver services that satisfy the customer's need for IT services. The delivered services, the specified service levels and the

customer's service needs are reviewed with the customer on a regular basis. When necessary, the service level agreement is adjusted.

There are two basic issues targeted by this key process area: first, the service to be delivered is specified in a contract – the service level agreement – containing *measurable* service levels. Second, the service levels specified should address the business needs of the customer.

2. Service Delivery Planning:
   Purpose: The service delivery is planned in order to ensure that the specified services can indeed be delivered according to the agreed upon service levels.

3. Service Tracking and Oversight:
   Purpose: Service delivery is being tracked. The realized service levels are compared with the specified service levels and are reported to the customer and management on a regular basis. Corrective actions are taken when actual service delivery deviates from the specified service levels.

   The service provider reports to the customer the actual services delivered, the actual service levels, and, when relevant, calamities that hindered accurate service delivery. The service level reports are used as input for the evaluation of service level agreements (see Service Commitment Management).

4. Subcontract Management:
   Purpose: Select qualified IT subcontractors and manage them effectively.

   The service provider can select and hire subcontractors to delegate parts of the service. If this is the case, the service to be delivered by the subcontractors is laid down in a service level agreement. The service provider keeps track of the actual services delivered by the subcontractor and takes corrective actions when the actual service levels deviate from the specified service levels.

5. Configuration Management:
   Purpose: The integrity of products which are subject to or part of the IT services is established and maintained.

   Configuration Management involves the identification of the relevant hardware and software components which need to be put under configuration control. This includes components owned by the customer that are being managed by the service provider, components owned by the provider that are used by the customer and components owned by the provider that are used to deliver the service. Changes to the configuration are evaluated with

respect to the service level agreement and with respect to possible risks for the integrity of the configuration.

6. Event Management:
   Purpose: Events regarding the service are identified, registered, tracked, analyzed, and resolved. The status of events is communicated with the customer and reported to management.

   This key process area concerns the management of events that cause or might cause service delivery to deviate from the agreed upon service levels. Events can be either:

   - Requests for service from users. For example, requests for a new feature in the software;
   - Incidents that cause or will cause service levels to be lower than agreed upon if no action is being taken. For example, a server that is down might cause the specified maximum down-time to be exceeded if it is not restarted quick enough.

   To resolve requests for service and incidents, changes to the configuration might be necessary. The decision whether to implement the change request that results from a service request or incident is the concern of Configuration Management.

7. Service Quality Assurance:
   Purpose: Management is provided with the appropriate visibility into the processes being used and the services being delivered.

   Service Quality Assurance involves the reviewing and auditing of working procedures, service delivery activities and work products to see that they comply with applicable standards and procedures. Management and relevant groups are provided with the results of the reviews and audits. Note that where Service Tracking and Oversight is concerned with measuring service quality in retrospect, from an external point of view, Service Quality Assurance is concerned with measuring quality in advance, from an internal point of view.

9.4.3 Level 3: Defined

At level three, an organization standardizes its processes and uses tailored versions of these standard processes to deliver the IT services. The goal is to establish a more predictable performance of the processes and hence increase the ability

of the organization to draw up realistic service level agreements. The level three key process areas each fall into one of the three process categories: management, enabling or delivery.

The first category – service management – is concerned with the tailoring of the standard service processes to the customer and the service level agreement at hand. Also, the actual service processes need to be integrated with each other and with third party service processes (Integrated Service Management). The second category – enabling – deals with making standard processes available and usable. The organization develops and maintains standard processes for each of the services it delivers. Usually, organizations will provide several services to one customer at the same time. Hence, not only the service processes themselves, but also the integration of these processes has to be standardized as much as is feasible (Organization Process Definition). To coordinate process efforts across services and organizational units and over time, organizational support is institutionalized (Organization Process Focus). Also, to teach people how to work with the standards and how to perform their roles, a training program needs to be put in place (Training Program). The third category – service delivery – concerns the actual delivery of the services to the customer using the tailored service processes (Service Delivery).

The level three key process areas are described as follows:

1. Organization Process Definition:
   Purpose: Develop and maintain a usable set of service process assets that improve process performance across services, and provide a basis for cumulative, long-term benefits to the organization.

   This key process area involves the creation and maintenance of standard service processes, and a process database which contains historic data on used processes, including the service level agreements, the service planning, the service level reports and the event management database. Based on historic service processes a service catalog is developed and maintained which contains the services and service levels that the organization provides.

2. Organization Process Focus:
   Purpose: Establish organizational responsibility for service process activities that improve the organization's overall service process capability.

   The activities needed to assess, develop, maintain and improve the organization's service processes are resourced and coordinated across current and future services.

3. Training Program:
   Purpose: Develop the skills and knowledge of individuals so they can perform their roles effectively and efficiently.

4. Integrated Service Management:
   Purpose: Integrate the IT service and management activities into a coherent, defined IT service process that is derived from the organization's standard service process.

   The service planning is based on this tailored service process and describes how its activities will be implemented and managed. The service planning takes the organization-wide capacity and availability of resources into account. Cooperation with third parties that also deliver IT services or products to the customer, is planned. Note that these third parties can be external providers or organizational units of the customer itself. An example of this could be the customer having its own helpdesk which relays reports of hardware failures to the service provider. Procedures need to be put in place on how these reports will be delivered to the service provider and whether the helpdesk or the service provider will inform the user of the status of the report. An example which involves coordination with third parties that deliver products to the customer, is software development. Suppose a third party is developing software for the customer that is to be managed and maintained by the service provider. Involvement of the service provider in the development process can ensure that the maintainability of the software is sufficiently being taken into account during development.

5. Service Delivery:
   Purpose: Consistently perform a well-defined service delivery process that integrates all service delivery activities to deliver correct, consistent IT services effectively and efficiently.

   Service Delivery is the actual execution of the service delivery activities according to a tailored version of the services' defined service processes (which is the output of the Integrated Service Management key process area). Because the service activities depend on the particular services being provided, there is no fixed list of activities to be performed. However, all services should perform the activities as defined in the level two key process areas. The list of activities will be filled in depending on the services at hand. For example, in the case of software maintenance the general service activities can be extended with the software engineering tasks mentioned in the key process area Software Product Engineering of the Software CMM (SEI 1995, pp. 241–261).

9.4.4 Level 4: Managed

At level four, organizations gain a quantitative understanding of their standard processes by taking detailed measures of service performance and service quality (Quantitative Process Management) and by using these quantitative data to control the quality of the delivered services (Service Quality Management).

There are two level four key process areas:

1. Quantitative Process Management:
   Purpose: Control the process performance of the service delivery quantitatively.

2. Service Quality Management:
   Purpose: Develop a quantitative understanding of the quality of the services delivered and achieve specific quality goals.

9.4.5 Level 5: Optimizing

At level five, service providers learn to change their processes to increase service quality and service process performance (Process Change Management). Changes in the processes are triggered by improvement goals, new technologies or problems that need to be resolved. New technologies are evaluated and introduced into the organization when feasible (Technology Change Management). Problems that occur are prevented from recurring by changing the processes (Problem Prevention).

The level five key process areas are:

1. Process Change Management:
   Purpose: Continually improve the service processes used in the organization with the intent of improving service quality and increasing productivity.

2. Technology Change Management:
   Purpose: Identify new technologies and inject them into the organization in an orderly manner.

3. Problem Prevention:
   Purpose: Identify the cause of problems and prevent them from recurring by making the necessary changes to the processes.

9.5 Examples of level two key process areas

In this section we present two of the level two key process areas that are used to implement the processes identified in section 8.4. Section 9.5.1 presents the goals

and key practices of the Service Commitment Management key process area. This key process area is targeted at aligning the service to be delivered with the service needs of the customer. In section 9.5.2 the Event Management key process area is presented. The Event Management key process area is targeted at managing and controlling the handling of all kinds of events that happen during the delivery of services, and at managing the communication with the customer about those events.

Each of the key process areas is described in terms of its goals and key practices. The key practices are divided into five types: commitment to perform, ability to perform, activities performed, measurement and analysis, and verifying implementation (SEI 1995).

Where necessary, key practices are augmented with examples of *typical* content of documents or implementation of activities. Organizations should normally consider these elaborations mandatory, except when there is a good reason not to implement them. Boxed text is meant to give examples or clarify the key practices.

9.5.1 Service Commitment Management

The main purpose of Service Commitment Management is to ensure that the service commitments between service provider and customer, and hence the actual services delivered, are based on the IT service needs of the customer. The service commitments specify (amongst other things) the results of the services to be delivered. These results should contribute to fulfilling (parts of) the IT service needs of the customer.

The activities in this key process area are targeted at ensuring that the service commitments are based on the IT service needs, and stay in line with possibly changing IT service needs. This is enforced by periodic evaluations of the service commitments with respect to the IT service needs and by evaluations of the actual services delivered.

Goals

Goal 1	**Service commitments are documented.**
Goal 2	**Service commitments are based on current and future IT service needs of the customer.**

Commitment to Perform

Commitment 1	**A service manager is designated to be responsible for negotiating service commitments.**

The service commitments consist of external and internal commitments. External commitments can be both agreements with the customer on the services to be delivered, and agreements with third parties on out-sourced services. Internal commitments are agreements between internal groups and individuals on the resources and activities needed to accurately deliver the agreed services. The service commitments to the customer are set down in a service level agreement. Commitments by a third party are set down in a separate service level agreement between the organization and the third party, see also the Subcontract Management key process area. The internal commitments are described in the service delivery plan.

Commitment 2 **The IT service is specified and evaluated according to a written organizational policy.**

This policy minimally specifies that:

1. The IT service needs of the customer are identified and documented.
2. The IT service needs of the customer are reviewed by:
   - the customer, and,
   - the service manager.
3. The IT service needs of the customer are used as the basis for negotiating the service commitments with the customer.
4. The service commitments are documented.
5. The service commitments are reviewed by:
   - the customer,
   - the service manager,
   - senior management, and,
   - other affected groups.
6. The service commitments are evaluated on a periodic basis.

Ability to Perform

Ability 1 **Responsibilities for developing the service commitments are assigned.**

1. The service manager, directly or by delegation, coordinates the development of the service commitments.

Ability 2 **Adequate resources and funding are provided for developing the service commitments.**

Ability 3 **Service managers are trained to perform their service commitment management activities.**

Examples of training include:

- negotiation methods and techniques,
- the application domain.

Activities Performed

Activity 1 **The IT service needs of the customer are identified according to a documented procedure.**

This procedure minimally specifies that:

1. The IT service needs are identified in cooperation with the customer.
2. The IT service needs are reviewed by the customer.

Activity 2 **The IT service needs are documented.**

The IT service needs typically cover:

1. The business strategy and IT strategy of the customer.
2. The business processes supported by the IT.
3. The relevant IT components.
4. Expected changes to the business strategy, IT strategy, business processes and IT components.
5. Current IT services used by the customer.

Activity 3 **The service commitments are documented.**

The service commitments minimally cover:

1. The purpose, scope, and goals of the services to be delivered.
2. Specification of the services to be delivered.

3. Specification of the quality levels of the services to be delivered.

   Service quality levels specify the minimum or maximum value for all relevant attributes of the service. Service quality levels should be specified in a measurable way, because the service levels of the delivered services have to be reported to the customer, see the key process area Service Tracking and Oversight.
   Examples of performance attributes of IT services include:
   - the guaranteed availability of a system,
   - the maximum response time of a system,
   - maximum processing times of service requests.

4. The service delivery schedule.

   The service delivery schedule specifies when certain service activities will take place that have an effect on the service levels. Examples of such activities are:
   - the delivery and installation of new software releases,
   - planned outage of systems for maintenance purposes,
   - upgrading hardware due to increasing performance demands.

   Note that the service delivery schedule both contains service activities that take place at a fixed moment in time, for example new releases, or activities that are executed when certain other conditions are met, for example installing additional hardware to meet increasing performance demands.

5. Specification of the service conditions.

   Service conditions are resolutive conditions that the customer has to fulfill (i.e. the service provider is exempted from delivering the service according to the service quality levels, if the customer does not fulfill the service conditions).

6. Specification of calamities.

Calamities are situations in which the service provider is exempted from delivering the service according to the service quality levels. Note that these calamities are subject to negotiation. Examples of such situations are:

- natural disasters such as earthquakes, storms, tidal waves,
- civil unrest, strikes, riots, war,
- power failures, telecommunication failures.

7. Agreements on reviewing actual service delivery.

Part of the service commitments are agreements on how and when the service provider will report on the delivered services to the customer. The service delivery reports minimally cover the actual service levels as compared to the service levels specified in the service commitments.

Refer to the key process area Service Tracking and Oversight for practices concerning the tracking and review of actual service delivery.

8. Planning of service evaluation.

Part of the service commitments are agreements on how and when the service commitments will be evaluated.

Refer to Activity 4.

Activity 4 **Service commitments are evaluated with the customer on both a periodic and an event-driven basis.**

The primary purpose of periodic and event-driven service evaluations of the service commitments with the customer is to ensure that the actual services delivered stay in line with current and future IT service needs of the customer.

1. The current IT service needs of the customer are identified and documented.

Refer to Activity 1 and Activity 2.

2. The current IT service needs of the customer are compared with the previously identified IT service needs.
3. The current IT service needs of the customer are compared with the previously established service commitments.
4. If necessary, the service commitments are adapted to the new IT service needs.

Activity 5 **Actual service delivery is evaluated with the customer on both a periodic and an event-driven basis.**

1. Actual service delivery is compared with the service commitments.

   Refer to the key process area Service Tracking and Oversight for practices concerning the tracking of actual service delivery.

2. Service delivery risks are addressed.
3. Non-conformance to the service commitments is addressed.
4. Significant issues, action items, and decisions are identified and documented.
5. Action items are assigned, reviewed, and tracked to closure.

Measurement and Analysis

Measurement 1 **Measurements are made and used to determine the status of the service commitment management activities.**

Examples of measurements include:

- work completed, effort expended, and funds expended in the service commitment management activities compared to the plan.

Verifying Implementation

Verification 1 **The service commitment management activities are reviewed with senior management on a periodic basis.**

The primary purpose of periodic reviews by senior management is to provide awareness of, and insight into, service process activities at an appropriate level of abstraction and in a timely manner. The time between reviews should meet the needs of the organization and may be lengthy, as long as adequate mechanisms for exception reporting are available.

1. The technical, cost, staffing, and schedule performance is reviewed.
2. Conflicts and issues not resolvable at lower levels are addressed.
3. Service delivery risks are addressed.
4. Action items are assigned, reviewed, and tracked to closure.
5. A summary report from each meeting is prepared and distributed to the affected groups and individuals.

Verification 2 **The service commitment management activities are reviewed with the service manager on both a periodic and event-driven basis.**

1. Affected groups are represented.
2. Status and current results of the service commitment management activities are reviewed.
3. Dependencies between groups are addressed.
4. Conflicts and issues not resolvable at lower levels are addressed.
5. Service delivery risks are reviewed.
6. Action items are assigned, reviewed, and tracked to closure.
7. A summary report from each meeting is prepared and distributed to the affected groups and individuals.

Verification 3 **The service quality assurance group reviews and/or audits the service commitment management activities and work products and reports the results.**

Refer to the Service Quality Assurance key process area.

At a minimum, the reviews and/or audits verify:

1. The activities for reviewing and developing service commitments.

9.5.2 Event Management

The main purpose of the key process area Event Management is to identify, record, track, analyze, and resolve events that occur during service delivery. An event is an occurrence that – if not resolved – eventually will cause the service provider to break its service commitments. Two types of events are distinguished: service requests and incidents. Service requests are requests by the customer for certain service activities to be performed. Note that these activities should fall within the bounds of the service commitments. For example, the customer asks for an extra workplace to be installed. Incidents are events that need to be resolved in order to meet the service commitments. For example, if a system goes down it has to be restarted before the maximum downtime will be exceeded.

Events are always concerned with one or more IT components. Events are resolved by action items.

Goals

Goal 1 **Event management activities are planned.**

Goal 2 **Events are identified, recorded, analyzed, tracked, and resolved.**

Goal 3 **Affected groups and individuals are informed of the status of events and action items.**

Commitment to Perform

Commitment 1 **A written organizational policy is followed for implementing event management (EM).**

This policy typically specifies that:

1. Responsibility for EM for each service is explicitly assigned.
2. EM is implemented throughout the duration of the service commitments.

3. A repository for storing event information is made available.
4. The event repository and EM activities are audited on a periodic basis.

Ability to Perform

Ability 1 **A group that is responsible for coordinating and implementing EM for the service (i.e., the EM group) exists.**

The EM group coordinates or implements:

1. Creation and management of the service's event repository.
2. Development, maintenance, and distribution of EM plans, standards, and procedures.
3. Management of the access to the event repository.
4. Changes to the event repository.
5. Recording of EM activities.
6. Production and distribution of EM reports.

Ability 2 **Adequate resources and funding are provided for performing the EM activities.**

1. A manager is assigned specific responsibility for EM.
2. Tools to support the EM activities are made available.

Examples of support tools include:

- workstations and/or portable computers,
- event management software.

Ability 3 **Members of the EM group and related groups are trained in the objectives, procedures, and methods for performing their EM activities.**

Examples of related groups include:

- service quality assurance group,
- configuration management group,
- end-users, and
- service engineers.

Examples of training include:

- the standards, procedures, and methods to be followed for EM activities, and
- the role, responsibilities, and authority of the EM group.

Activities Performed

Activity 1 **An EM plan is prepared for each service according to a documented procedure.**

This procedure typically specifies that:

1. The EM plan is developed in the early stages of, and in parallel with, the overall service delivery planning.
2. The EM plan is reviewed by affected groups.
3. The EM plan is managed and controlled.

Activity 2 **A documented and approved EM plan is used as the basis for performing the EM activities.**

The plan covers:

1. Estimates of the event workload.
2. The EM activities to be performed, the schedule of the activities, the assigned responsibilities, and the resources required (including staff, tools, and computer facilities).

Activity 3 **An event management library system is established as a repository for the event records.**

This library system:

1. Provides for the storage, update, and retrieval of event records.
2. Provides for the sharing and transfer of event records between affected groups.
3. Helps in the use of event management procedures.

   Refer to Activity 4.

4. Provides for the archival and retrieval of historic event information.

5. Supports production of EM reports.

Activity 4 **Events are identified, recorded, analyzed, reviewed, and resolved according to a documented procedure.**

This procedure typically specifies that:

1. The events are recorded in sufficient detail so that the content and the status of each event are known. This typically includes:
   - a unique identifier,
   - description of the event,
   - date and time of occurrence,
   - name and contact information of the person who reported the event,
   - the configuration items concerned, and
   - relevant characteristics of the situation in which the event occurred.
2. The impact of the event to the service commitments is assessed and documented.
3. Action items resulting from events are:
   - identified,
   - assessed for risk,
   - documented,
   - planned,
   - initiated,
   - communicated to the affected groups and individuals,
   - tracked to closure, and
   - evaluated.

Activity 5 **Affected groups and individuals are informed of the status of events on both a periodic and event-driven basis.**

Examples of affected groups and individuals include:

- configuration management group,
- service delivery group,
- service manager,
- users.

Activity 6 **Standard reports documenting the EM activities and the contents of the event repository are developed and made available to affected groups and individuals.**

Examples of reports include:

- event description and status,
- action item description and status,
- summary of events by configuration item,
- summary of events during a certain period.

Activity 7 **Event repository audits are conducted according to a documented procedure.**

This procedure typically specifies that:

1. There is adequate preparation for the audit.
2. The integrity of the event repository is assessed.
3. The facilities of the event management library system are reviewed.
4. The completeness and correctness of the repository are verified.
5. Compliance with applicable EM standards and procedures is verified.
6. The results of the audit are reported to the service manager.
7. Action items from the audit are tracked to closure.

Measurement and Analysis

Measurement 1 **Measurements are made and used to determine the status of the events in the event repository.**

Examples of measurements include:

- number of events unresolved,
- average leadtime of closed events,
- percentage of events not closed within the maximum time.

Measurement 2 **Measurements are made and used to determine the status of the EM activities.**

> Examples of measurements include:
>
> - number of events processed per unit time,
> - number of action items completed per unit time,
> - effort expended and funds expended in the EM activities.

Verifying Implementation

Verification 1 **The EM activities are reviewed with senior management on a periodic basis.**

Verification 2 **The EM activities are reviewed with the service manager on both a periodic and event-driven basis.**

Verification 3 **The EM group periodically audits event repositories to verify that they conform to the documentation that defines them.**

Verification 4 **The service quality assurance group reviews and/or audits the EM activities and work products and reports the results.**

9.6 Conclusions

In this chapter we have presented a capability maturity model aimed at organizations that deliver IT services. This IT Service Capability Maturity Model is aimed at providing processes that help organizations close the service gaps as described in chapter 8.

If we compare the IT Service CMM with the Software CMM we see that the models differ in two major ways:

Customer focus versus system focus The delivery of services in the IT Service CMM is based on measurable service levels that are established by the customer and the service provider together. Software development projects in the Software CMM are based on system requirements, derived outside the scope of the Software CMM.

The level two practices of the Software CMM are aimed at managing the software requirements and changes therein. The software development plan

is based on those requirements, and the tracking of the software project is aimed at monitoring whether progress is according to plan.

The IT Service CMM on the other hand, aims at fulfilling the service levels specified in the service commitments. These service commitments are derived from the service needs of the customers, which in turn are based on the business needs and strategy of the customer. Both service commitments and the actual service delivered are reviewed with the customer on a regular basis.

The key process areas Service Commitment Management and Service Delivery Planning implement respectively the 'management of commitments' and 'maintenance planning' processes identified in section 8.4. These processes are aimed at closing respectively the gap between the service as expected by the customers and the company perceptions of the expected service (gap 1) and the gap between the company perceptions of the expected service and the service designs and standards (gap 2). See figure 8.3 on page 129.

Requirements-driven versus event-driven The IT Service CMM explicitly deals with the fact that software maintenance and other IT services are driven by events that occur during the delivery of the service. The Event Management key process area deals with the management and control of these events. In addition, the Service Delivery Planning and Service Tracking and Oversight key process areas require the service organization to explicitly deal with the estimation and tracking of the expected and actual number and type of events.

The key process areas Service Tracking and Oversight and Event Management implement respectively the 'maintenance activity tracking' and 'event management' processes identified in section 8.4. These processes are aimed at closing respectively the gap between the service designs and standards and the actual service delivery (gap 3) and the gap between the actual service delivery and the external communication to the customers (gap 4). See figure 8.3 on page 129.

Obviously, next to differences there also are similarities. Both the Software CMM and the IT Service CMM require organizations to implement subcontract management, configuration management, and quality assurance. However, internally these key processes still differ because the primary focus of the Software CMM is on software development whereas the IT Service CMM focuses on service provision.

In order to validate the IT Service CMM we need to show that it is part of a valid prediction system in which organizations that score higher on the IT Service CMM scale, are delivering higher quality services than organizations that score

lower on the IT Service CMM scale. However, such a validation is rather difficult to perform for several reasons:

- Due to the fact that the development of the IT Service CMM started halfway during the second of the two research projects covered in this thesis, little time was left to apply the model in practice.
- For the same reason, the model itself has not been specified fully. Only the level two key process areas have been specified down to the level of individual activities. The higher level key process areas remain to be done.
- A full validation, in which we show that IT service organizations that follow the IT Service CMM perform better than those that do not, would probably take several years. From literature we know that software organizations need up to two years to move one level up in the Software CMM. Because the structure of the two models is similar we expect similar figures for organizations that use the IT Service CMM.

These reasons made it impossible to validate the IT Service CMM during the course of the research projects Concrete Kit and Kwintes. However, we have done two case studies in which we applied the model in a process assessment. These case studies are described in the next chapter.

Chapter 10

Assessing IT Service Maturity

In this chapter we describe two case studies in which the IT Service CMM was used as a reference framework for the assessment of two IT service providers. For both assessments we used a questionnaire, based on the key practices of the IT Service CMM. This questionnaire was developed and pilot tested by a MSc student. The goal of both case studies was to apply the IT Service CMM and the related questionnaire in actual process assessments.

We used two different approaches to conduct the assessments. In the first assessment, which was done at the IT management department of one of the project partners, we performed the assessment in a workshop form. The aim was to perform a quick-scan of the maturity of the organization using limited time and resources. The second case study was done at a software support department of one of the other project partners. In this case, the assessment was done at the site of the partner, using a traditional assessment format (e.g. Olson, Humphrey and Kitson 1989).

The questionnaire used differed between the two assessments. In the first assessment a version was used which contained one or two questions for each key practice of the Activities Performed category. For each key practice, the questionnaire asks whether it is performed. If the key practice is to be performed according to a documented procedure, a second question asks whether that is the case. For example, the Configuration Management key practice:

Activity 1 A CM plan is prepared for each service according to a documented procedure.

The matching questions in the questionnaire ask:

1a. Is a CM plan prepared for each service?

1b. Is the CM plan prepared according to a documented procedure?

In the later version of the questionnaire, used in the second case, there is exactly one question per key practice[1]. So the question related to the key practice above would simply be:

1. Is a CM plan prepared for each service according to a documented procedure?

Possible answers to the questions are: 'yes, always', 'not always', 'no, never', 'do not know'.

10.1 Case 1: A quick-scan assessment

The organization assessed in this case study (organization A) is the IT management and exploitation department of the Tax and Customs Computer and Software Centre of the Dutch Tax and Customs Administration. This organization installs, maintains, and operates hardware and software used by the Dutch Tax and Customs Administration. This includes hardware and software operated centrally, and the hardware and software used locally at the different sites of the Tax and Customs Administration.

Systems development and maintenance is done by a sibling department. However, organization A is the front-office towards the customers.

10.1.1 The assessment approach

The goal of this assessment was twofold:

- To test the IT Service CMM on a whole organization, and
- To investigate the feasibility of performing an assessment in a workshop form.

Because performing a process assessment requires quite some effort from the participating organization, we investigated the possibility of performing a quick-scan of the maturity of the organization. The hypothesis was that we could get a quick overview of the maturity of the organization by doing an assessment in a workshop format, using participants with good overall knowledge of the organization.

The assessment was done as follows:

Preparation A number of assessment participants was selected from the internal audit pool of the Software Centre. These participants perform audits of the

[1] Unlike the Software CMM questionnaire (Zubrow, Hayes, Siegel and Goldenson 1994), where the relationship between key practices and questions is not one to one.

processes of organization A. Hence, they are familiar with the processes employed in organization A. Next to these four participants, two employees of organization A itself participated as well. The assessment workshop was led by the author of this thesis. It was planned to discuss all six[2] level two key process areas.

Assessment The assessment workshop took 3 hours. The participants received a short introduction to the IT Service CMM and the goals of the assessment. Each participant received a version of the questionnaire. The assessment covered the Service Planning and Evaluation and Configuration Management key process areas. Each participant filled in the questionnaire, and the questions were discussed one by one. For each question a consensus answer was sought. Because the discussion of the answers took much more time than planned, only two of the six key process areas were discussed. The assessment was rounded off with an oral evaluation.

Follow-up A written report of the assessment was sent to the participants.

10.1.2 Assessment results

During the assessment, the participants answered the questions on the questionnaire one by one. After each participant filled in one question, the answers were discussed and a consensus answer was sought. After a consensus was reached the next question was to be filled in, etc. Though the plan was that people would first fill in the question without talking to each other, to ensure that the opinions of the individual participants would be recorded, this turned out to be difficult. People tended to discuss the question with each other before everybody had written down an answer, thus potentially influencing the answers of other participants.

The individual answers to the questions were generally not identical. People disagreed for several reasons:

- Participants did not know whether the practice asked about in the question was actually performed.

- Participants knew that a practice was performed in some parts of the organization but not in others.

[2] At the time of this assessment, level two of the IT Service CMM had six key process areas. The Service Planning and Evaluation key process area was later split into two key process areas, namely Service Commitment Management and Service Delivery Planning.

- Participants did know about one part of the organization performing the practice, but did not know whether other parts of the organization applied the practice.
- Some practices are applied part of the time. For example, many practices are not applied in so-called emergency situations.

Also, oftentimes practices are applied, but there is no formal procedure describing the practice, or the procedure is outdated.

Key process area	Yes	Not always	Never	Unknown	No agreement	Total
Service Planning and Evaluation	3	13	3	3	0	22
Configuration Management	1	12	2	6	3	25

Table 10.1: Consensus answers

Table 10.1 shows the consensus answers arrived at after discussing the individual answers. The 'yes' column shows how often the participants agreed that a certain practice was always performed. The 'not always' column indicates how many practices are performed some times, or in some parts of the organization. The column 'never' depicts the number of practices that are never performed. The 'unknown' column indicates the number of practices of which none of the participants knew whether they were implemented in the organization or not. The 'no agreements' column shows how often the participants disagreed on the answer, and hence no consensus answer was reached.

If we look at the results per key process area the following picture arises for the Service Planning and Evaluation key process area: organization A does not follow a consistent procedure for identifying and documenting the needs of the customer, documenting the commitments made, and planning the service. In addition, not all aspects of the service delivery as required by the IT Service CMM are planned:

- Costs are only estimated at the level of the complete organization, not at the level of individual services.
- No consistent estimation is done of the expected service workload. No procedure to do so exists.
- Risks are not always estimated and reviewed.

- Service commitments are being reviewed by senior management, but not according to a documented procedure.
- Data about the actual service delivery is being gathered, but the participants are not sure whether this data is actually used to evaluate the planning.
- Sometimes the service is evaluated with the customer, but not on a periodic basis, and not according to a documented procedure.

For the key process area Configuration Management we arrived at the following conclusions from the assessment:

- Configuration management activities are not always planned, and if they are planned, the plans are not always used. The participants do not know why this is the case.
- Not all departments use a configuration management database, and changes to the configuration baseline are not always controlled.
- Change procedures do exist, but are often laborious because tools do not match the procedures or vice versa. This is a major reason why the status of configuration items is not maintained properly.
- Reports are made of the contents of and changes to the configuration databases. However, the participants suspect that these reports remain unused because they do not match the needs of the potential users of those reports.
- Audits of the configuration management databases are done, reviews of the configuration management activities hardly ever.

The assessment was rounded off with an oral evaluation. The participants were positive about the assessment. They appreciated the discussion with their colleagues, and the chance to participate in new developments. The large number of 'not always' answers was noted as a negative point.

10.1.3 Assessment conclusions

The goal of this assessment was twofold: to test the IT Service CMM to assess a complete organization and to test the feasibility of a quick-scan assessment in a workshop setting. We conclude the following: though the workshop form, combined with assessment participants that have a global overview of the whole organization, seemed to be an attractive alternative to the usual on-site assessment approach, this was not the case. Compared to a standard assessment approach, the

workshop form does take much less effort. In addition, the workshop form makes it possible to quickly reach a consensus on the questions in the questionnaire. However, these advantages do not outweigh the biggest disadvantage experienced in this assessment: the participants, though having a global overview, lack detailed insight in the daily operations of the organization. This resulted in a lot of questions answered with 'not always'. So even though the goal was to perform 'just' a quick-scan of the maturity of the organization, the assessment results are insufficient to give a good judgment of the maturity of the organization.

10.2 Case 2: On-site assessment

Organization B is a marketing, sales and support (MS&S) department of one the project partners. Organization B is responsible for selling a software product that supports business modeling, conceptual modeling, and software development. In addition, the organization supports the customers of the product and informs the users of updates and bug fixes.

Organization B consists of a marketing team, sales support team, a service desk, and account management. Organization B is headed by the manager marketing, sales and support. The manager MS&S is the principal of this assessment.

10.2.1 The assessment approach

The goal of this second assessment case study was to:

- support organization B in determining the quality of its service processes and indicate directions for improvement, and
- use and test the IT Service CMM in a real process assessment, following a standard assessment procedure.

The assessment was performed by an assessment team consisting of one senior consultant Software Process Improvement of the company and the author of this thesis. The assessment was done according to the internal Software CMM assessment procedure of the company. We adapted this procedure to the IT Service CMM by replacing the Software CMM key processes by the IT Service CMM key processes and by using the IT Service CMM questionnaire. The approach itself was not changed. During this assessment, we focused on the key process areas Service Commitment Management, Event Management and Configuration Management. These three key process areas were selected for two reasons:

- Organization B does not currently use service level agreements but feels that it will be forced by its customers and/or competition to do so in the near future. By looking at the Service Commitment Management key process area organization B wants to gain insight in the practices it needs to implement to be able to use service level agreements.
- The Event Management and Configuration Management key process areas were selected because one of the primary processes of organization B is the support of customers in using the software product. Moreover, if the organization is to use tailored service level agreements that contain commitments to the customers, the Event Management and Configuration Management processes need to be able to support and enable these commitments.

The assessment was done as follows:

Preparation Before the assessment started, an assessment plan was developed and agreed upon by the manager MS&S. Together with the manager, the assessment team selected a number of employees that would participate directly in the assessment. Six people were selected: the team manager service desk, one service desk employee, the team manager sales support, one sales support employee, the team manager marketing and one account manager. These six people filled in the questionnaires, were interviewed and participated in all key process area discussions.

Assessment The assessment was conducted during three consecutive days at the premises of organization B. The following activities were performed:

Day 1. During a kick-off meeting all employees of organization B were informed of the goals and approach of the assessment. Next, the questionnaires were filled in by the participants selected in advance. The questionnaires were analyzed by the assessment team and used as a basis for the interviews held in the afternoon.

Day 2. The assessment team made a list of positive points and possible improvements. These points were used as input for the three discussion sessions held on day two and three.

In the afternoon two discussion sessions were held during which the Event Management and Configuration Management key process areas were discussed.

Day 3. The last day started with the final key process area discussion (Service Commitment Management). Next, the assessment team prepared the final presentation. During the final presentation, the results of the

assessment were summarized and presented to the employees and the manager of organization B. All action items identified during the assessment were agreed upon, assigned to people, and dates were set. The presentation session was ended with a short oral evaluation of the assessment.

Follow-up A written report of the assessment was delivered to the organization. The organization is currently implementing the actions defined during the assessment.

10.2.2 Assessment results

The analysis of the questionnaires in preparation for the interviews showed that quite some conflicting answers were given. Some of the participants indicated that certain practices were not implemented, others said they were. During the interviews it became clear that these inconsistencies were due to a different interpretation of the questions and due to the fact that people were not aware of certain practices.

Key process area	Commitment	Ability	Activities	Measurement	Verifying
Service Commitment Management	0/2	1/3	1/5	0/1	0/3
Event Management	0/1	1/3	2/7	1/2	0/4
Configuration Management	0/1	1/4	4/9	0/1	0/4

Table 10.2: Number of key practices implemented by organization B. The notation x/y indicates that x out of y required practices were implemented.

Table 10.2 shows the compliance of organization A with the key practices of the three key process areas assessed, based on the questionnaires and the interviews. The main issues derived by the assessment team from the questionnaires and the interviews were:

Service Commitment Management Organization B uses one standard maintenance and support contract for all its customers. This contract does not contain any commitments to the customer. This means that the needs of the customer are not identified, nor are service levels specified. Evaluation of

the service commitments and the delivered service with the customer is not performed. However, the organization has started a new after-sales procedure that contains some evaluation aspects. Because organization B has a large number of customers (about 2700) individual service level agreements and evaluation of the delivered service will be difficult to achieve.

Event Management There is no planning of the event management activities in organization B. This is not surprising since all customers have the same maintenance and support contract and are thus treated similarly. The service desk uses a helpdesk system for the registration of events. This system also supports the service desk employees with the registration of calls from customers. Other parts of the process are not formalized. Though a lot of information is present in the service desk database, for example how often customers call the service desk, this information is not distributed properly to other parts of the organization. Finally, it is known that the information in the service desk database is incorrect to a certain extent. However, there are no activities implemented to prevent or correct this.

Configuration Management Again, because no distinction is made between customers, configuration management is not planned separately for each customer. Two issues are important for organization B: first, the registration of which customer runs which version and which parts of the software, and second, what is contained in the next release of the software. The first point is arranged for sufficiently. The second point, however, is a problem for organization B. The actual development of the software is done by a separate department of the company, outside organization B. Organization B has insufficient insight in the contents of new releases, and this causes problems in the communication with the customers.

During the key process area discussions with all involved employees, positive points and improvement points were discussed. Multi-voting was used to select improvement points to focus on. During the sessions, concrete improvement actions were defined, including the persons responsible and target dates for the implementation.

One example of the improvement points discussed and improvement actions taken is the following: the different departments in organization B have difficulties using and accessing the information that each of the departments has about the customers of the company. This relates to for example Activity 5 of the Event Management key practices. To improve the exchange of information, the information needs of each of the teams will be specified and a procedure for the exchange

What	Who	When
Document the information needs of each of the departments	teamleaders	1 month
Develop a plan to implement the information exchange	teamleaders marketing and sales support	2 months
Implement the plan	teamleader service desk	2 months
Setup verification process	manager MS&S	2 months
First evaluation of the new procedure	manager MS&S	3 months

Table 10.3: Example improvement actions defined for one of the Event Management improvement points

of information will be developed. Table 10.3 shows the actions that were agreed on.

Finally, during the final presentation the results were summarized by the assessment team and presented to the employees and manager of organization B. The actions defined were summarized and the assessment team presented suggestions for further steps after successful implementation of the actions. Some of these suggested further steps were:

- Develop policy statements for the Event Management and Configuration Management processes to make it clear to all people involved what the policy of organization B is, what needs to be done minimally, and how these processes should be performed.
- Develop a configuration management plan in cooperation with the software development department.

10.2.3 Assessment conclusions

The assessment described in this section had two goals:

- Support organization B in determining the quality of its service processes and indicate directions for improvement.
- Use and test the IT Service CMM in a real process assessment.

The main conclusions with respect to the first goal are:

- The Event Management and Configuration Management processes are performed satisfactorily, though the actual performance of these processes depends quite heavily on the employees that execute them. The distribution

of information about these processes between teams within the organization needs improvement. The Configuration Management process needs to be implemented together with the software development department, because this process spans both departments.

- The Service Commitment Management key process area is essentially non-existent at organization B. Each customer receives the same maintenance and support contract, and no commitments are made to the customer. Because there are several developments that will force organization B to use service level agreements in the future, the organization needs to get experience with service level agreements. Also, information is needed about the levels of service the organization is able to guarantee to its customers. Action items are defined to gain some experience with service level agreements and to gain insight into the service levels organization B can maintain.

With respect to the second goal, we conclude that:

- The inclusion of the Event Management key process area in the IT Service CMM is justified by this case study. The management of events is very important for organization B to maintain the satisfaction of its customers. The Service Commitment Management process is not used by organization B. However, organization B does expect to be forced to use firm service commitments in the future by competitors and/or customers. This confirms our opinion that service commitment management is an essential part of the IT Service CMM.

- The scope of a process assessment needs to be based on the service *process.* In this case study, the assessment was done at the marketing, sales and support organization. During the assessment it became clear that the software development department is part of this service process as well. Because the software development department was not involved in the assessment it was difficult to define proper improvement actions for the areas where the software development department needed to be involved as well.

- The IT Service CMM requires organizations to identify the service needs of each of its customers. In the case of this organization, which has 2700 customers, this demand seems unrealistic. The questions is whether the IT Service CMM requirements are too strict in this respect.

- The IT Service CMM does not contain any processes or practices aimed at the link between a software development organization and a software support

organization. This link did turn out to be important in the case of organization B. The question is whether the IT Service CMM needs to be extended to cover this, and if so, how.

10.3 Conclusions

Both assessment case studies described in this chapter were aimed at getting practical experience with applying the IT Service CMM and the accompanying questionnaire in process assessments. Though the first case study was less successful than the second one, we feel that the usage of the IT Service CMM in these situations was appropriate. Especially the key process areas Service Commitment Management and Event Management give insight in the capability of the organizations with respect to the translation of customer service expectations in service level agreements and the management of the communication between customer and service provider. These two case studies strengthen our confidence that these processes have rightfully been included in the IT Service CMM.

Chapter 11

Conclusions

In this chapter we present the conclusions of this thesis. Our research has been concerned with the improvement of software maintenance processes. In chapter 1 we have introduced the context in which this research has been performed, and we presented the research questions and the research design. We distinguished two perspectives on process improvement: measurement-based improvement and maturity-based improvement. An overview of the literature on each of these approaches has been presented in chapter 2.

In part I we discussed measurement-based improvement of software maintenance processes. We presented four case studies in which we investigated the use of measurement programs in four software maintenance environments (chapters 3, 4 and 5). Based on the results of the four case studies we developed a Measurement Capability Maturity Model, which aims to capture the processes needed for mature software measurement. This Measurement CMM is described in chapter 6. As a first step towards validation of the Measurement CMM, we compared our Measurement CMM with other guidelines for implementing measurement programs in chapter 7. In order to facilitate this comparison, we developed a simple four-phase process model of measurement-based process improvement. The comparison revealed that most of the guidelines focus on the measurement side of measurement-based improvement. There is no consensus on, nor description of, activities needed to successfully *use* the results of measurement programs. This led us to investigate 'external' success factors for measurement programs, in addition to the well-known internal success factors. From different uses of measurement programs we deduced four external success factors. In addition, we provided guidelines which could be used by organizations to adhere to these external success factors.

Part II investigated maturity-based improvement of software maintenance processes. In chapter 8, we looked at the differences between services and products

in general, and how these apply to software maintenance. It was shown that the quality of services is judged on two dimensions: functional and technical quality. We presented the gaps model of service quality that explains how differences between expected and perceived quality come about. In addition, we presented an overview of a number of case studies which were aimed at testing part of the Concrete Kit and Kwintes research in service organizations. These theoretical and practical experiences led to the formulation of an IT service maturity model, which is presented in chapter 9. We discussed the objectives of the model, its structure and the key process areas it contains. Finally, chapter 10 presented two case studies in which we assessed two service organizations against the IT Service CMM.

In the remainder of this chapter we revisit the research questions, discuss the answers to them, and give directions for possible future research.

11.1 The research questions revisited

In chapter 1 we discussed six research questions, divided into two research issues, that formed the basis of the research presented in this thesis. With respect to the first issue – the usage of measurement for the improvement of IT services and IT service processes – we posed three questions:

1. How to introduce measurement in an IT service organization? What are the necessary steps to set up a measurement program and in which order should they be performed?
2. What are the prerequisites that need to be satisfied in order to improve the likelihood of success of the measurement program?
3. What is – or what should be – the relation between measurement and the maturity of the IT service organization?

We limited our research by looking specifically at software maintenance as one type of IT service. Furthermore, we limited the possible measurement applications to the planning and estimation of software maintenance effort.

The second research issue was concretized into the following three questions:

1. What arguments can we supply to support the notion of IT service process maturity?
2. How should mature IT service organizations look? Which processes should a mature service provider implement?
3. How can we use the concept of IT service process maturity in practice to support the improvement of IT service providers?

Again, we looked at software maintenance as one possible IT service to reduce the complexity of the research.

11.2 Measurement-based improvement

In the following three paragraphs we discuss the three measurement research questions.

11.2.1 How to implement measurement programs?

We have conducted four measurement program case studies to investigate the first of the three measurement-based improvement research questions. The lessons learned from these four measurement program case studies have been used to develop a maturity model for software measurement processes. This Measurement CMM provides an ordered set of measurement processes that, in our opinion, allows organizations to assess their own measurement maturity and that provides directions for the improvement of an organization's measurement processes.

We have taken a first step towards validation by comparing our measurement maturity models with other guidelines for implementing measurement programs. In order to facilitate this comparison, we developed a simple four-phase process model of measurement-based process improvement. Next, we mapped the activities of several measurement program guidelines, including our Measurement CMM, onto the process model.

From this comparison we drew three conclusions:

- there is quite some consensus on the basic activities needed to successfully implement measurement programs; but,
- at the same time, different frameworks emphasize widely different aspects of measurement program implementation, and,
- there is almost no consensus on, nor description of, activities needed to successfully *use* the results from measurement programs.

11.2.2 Prerequisites for successful measurement programs

The comparison of our Measurement CMM with other related work led us to investigate 'external' success factors for measurement programs, in addition to the well-known internal success factors. From different uses of measurement programs we have deduced four external success factors:

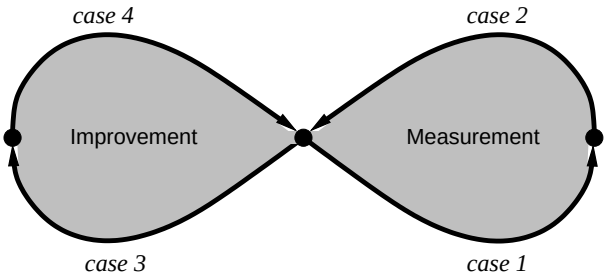


Figure 11.1: Weak points of the four measurement program case studies

1. The various assumptions underlying the measurement program should be made explicit. It should be decided if and when these assumptions are tested.
2. Different outcomes can result from a measurement program. An organization should consider all possible – negative and positive – outcomes and decide how to act on them.
3. The organization should act according to the outcomes of the measurement program, in order to reach the goals set or solve the problems identified.
4. The organization should monitor the changes implemented, in order to verify that these changes indeed constitute an improvement for the organization.

In addition, we have provided guidelines which could be used by organizations to adhere to these external success factors.

The combination of the well-known 'internal' success factors and our four 'external' success factors covers all four phases of the measurement-based improvement process model introduced in chapter 7. Hence, these two sets of success factors give an answer to the second research question: 'what are the prerequisites that need to be satisfied in order to improve the likelihood of success of the measurement program?'

To illustrate this, figure 11.1 shows the main weak point of each of the four measurement program cases. We see that the biggest failure factor of the first case was the lack of a rigorous implementation of the measurement program (an internal success factor). The weakest point of case two was the lack of validation of the maintenance function point model used (also an internal success factor). The third case mainly failed due to a lack of follow-up (an external success factor). Finally, the biggest weakness of case four was the fact that a big assumption made

– process variation is not an issue – was not made explicit, and was not tested (again an external success factor).

11.2.3 The relationship between measurement and process maturity

The last of the three questions concerns the relationship between process maturity and the success of measurement programs. We have found some indications that a more mature organization has a better possibility of a successful measurement program. However, there is no solid evidence. Moreover, we did not formally determine the maturity of the four organizations involved in the measurement program case studies. What we do observe is:

- Measurement program cases three and four were set up very similarly. However, the explanatory power of the data gathered in case three was much bigger than in case four. Since the measurement programs are practically the same, the cause of this effect must lie in the organizations. We suspect that organization three had a more mature software maintenance process than organization four. The process in organization four depended heavily on the individual engineer doing the maintenance work.

- The Ericsson department described in section 7.2 was assessed at level three of the Software CMM in 1995. Its measurement program is part of its effort to reach level four. The measurement program is quite successful in supporting the organizational goals of controlling its software process.

There seems to be a relationship between the maturity of the software process and the success of the measurement program. However, these few datapoints are not compelling evidence. Moreover, the 'correlation' between software process maturity and measurement program success could very well be caused by a third factor, for example the culture of an organization. Whether a causal relationship exists, and if so, what the size of the effect is, remains an open question.

11.3 Maturity-based improvement

In this section we discuss the three maturity-based improvement research questions.

11.3.1 The notion of IT service process maturity

The first of three research questions concerned the concept of IT service process maturity. In chapter 8 we discussed the concepts of services and products, and

we applied work from the service marketing literature to the software engineering domain, specifically to software maintenance. Starting from the argument that the quality of service is judged differently from the quality of products, we provided arguments for the conjecture that, because of these differences, software maintenance organizations need different processes to produce high quality results than software development organizations need. We described several case studies to indicate what types of processes that could be. These processes are aimed at supporting the functional quality of the services delivered, and at narrowing the organizational gaps that cause differences between the service quality as expected by the customer and as perceived by the customer.

We focused especially on software maintenance as one type of IT service in chapter 8. Hence, the arguments given lead to the conclusion that if we view software maintenance from a service perspective, other processes are needed for high maturity software maintenance organizations than those provided by the Software CMM. Or put differently, high process maturity means different things for software development and software maintenance organizations.

We have not by any means proven that the notion of IT service process maturity exists. However, we have argued that, if we accept that process maturity is a useful notion to support process improvement activities, then the processes needed for a high maturity IT service provider are different from the processes needed for a high maturity software developer.

11.3.2 Mature IT service processes

In chapter 9 we described the IT Service Capability Maturity Model. This IT Service CMM was designed to capture the issues described in chapter 8. The IT Service CMM includes the four processes identified in chapter 8 that organizations can apply to help close the four service gaps.

11.3.3 IT service process maturity applied

We took some first steps in applying the IT Service CMM in practice by doing two case studies in which the IT Service CMM was used as a reference framework for process assessment.

11.4 Future work

It is clear that quite some work described in this thesis cannot be considered to be completely finished. In part I we proposed a Measurement CMM to describe the processes needed for successful software measurement. This model has not been

specified as detailed as we would like. Furthermore, in order to show the merits of such a model, we need to validate it further. In chapter 7, we have derived four external measurement program success factors and activities that organizations can use to adhere to these success factors. It remains to be shown that these external success factors are indeed strictly necessary for a successful measurement program, i.e. that if an organization does not adhere these success factors, the measurement program will indeed fail.

In part II we looked at maturity-based improvement for software maintenance organizations. We argued that software maintenance should be seen as a service. Starting from that perspective, we suggested that different processes are needed for high quality software maintenance. This argument is difficult to prove directly. However, we have captured these processes in a IT Service Capability Maturity Model. By demonstrating the feasibility of this model, we can supply extra support for the software maintenance-as-a-service perspective.

To summarize, two major open questions remain:

- We have argued that several external success factors exist for measurement programs. These need to be shown to be *necessary* factors for successful measurement programs.
- We argued that software maintenance should be considered a service. Consequently, this leads to different processes that are *key* for high quality software maintenance. We have captured these processes in a maturity model. The validity of this model remains to be determined.

References

Abran, A. and Maya, M. (1995), A Sizing Measure for Adaptive Maintenance Work Products, *in* 'International Conference on Software Maintenance', IEEE Computer Society, Nice, France, pp. 286–294.

Abran, A. and Robillard, P. N. (1996), 'Function Point Analysis: An Empirical Study of Its Measurement Processes', *IEEE Transactions on Software Engineering* **22**(12), 895–910.

Albrecht, A. (1979), Measuring Applications Development Productivity, *in* 'Proceedings Application Development Symposium', SHARE/GUIDE, pp. 83–92.

Albrecht, A. J. and Gaffney, Jr, J. E. (1983), 'Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation', *IEEE Transactions on Software Engineering* **9**(6), 639–648.

Bach, J. (1994), 'The Immaturity of the CMM', *American Programmer* .

Banker, R. and Kemerer, C. (1989), 'Scale economies in new software development', *IEEE Transactions on Software Engineering* **15**(10), 1199–1205.

Basili, V. (1990), Recent Advances in Software Measurement, *in* 'Proceedings of the 12th International Conference on Software Engineering', pp. 44–49.

Basili, V., Briand, L., Condon, S., Kim, Y.-M., Melo, W. L. and Valett, J. D. (1996), Understanding and Predicting the Process of Software Maintenance Releases, *in* 'Proceedings of the 18th International Conference on Software Engineering', IEEE Computer Society Press, Berlin, Germany, pp. 464–474.

Basili, V. R. and Rombach, H. D. (1988), 'The TAME Project: Towards Improvement-Oriented Software Environments', *IEEE Transactions on Software Engineering* **14**(6), 758–773.

Berry, L. L. and Parasuraman, A. (1991), *Marketing Services - Competing Through Quality*, The Free Press, Macmillan Inc.

Birk, A., Derks, P., Hamann, D., Hirvensalo, J., Oivo, M., Rodenbach, E., van Solingen, R. and Taramaa, J. (1998), Applications of Measurement in Product-Focused Process Improvement: A Comparative Industrial Case Study, *in* 'Proceedings of the Fifth International Software Metrics Symposium', IEEE Computer Society TCSE, IEEE Computer Society Press, Bethesda, Maryland, USA, pp. 105–108.

Birk, A., van Solingen, R. and Järvinen, J. (1998), Business Impact, Benefit, and Cost of Applying GQM in Industry: An In-Depth, Long-Term Investigation at Schlumberger RPS, *in* 'Proceedings of the Fifth International Software Metrics Symposium', IEEE Computer Society TCSE, IEEE Computer Society Press, Bethesda, Maryland, USA, pp. 93–96.

Bøegh, J., Depanfilis, S., Kitchenham, B. and Pasquini, A. (1999), 'A Method for Software Quality Planning, Control, and Evaluation', *IEEE Software* **16**(2), 69–77.

Boehm, B. (1981), *Software Engineering Economics*, Englewood Cliffs, N.J: Prentice-Hall.

Bollinger, T. B. and McGowan, C. (1991), 'A Critical Look at Software Capability Evaluations', *IEEE Software* **8**(4), 25–41.

Bouman, J., Trienekens, J. and van der Zwan, M. (1999), Specification of Service Level Agreements, clarifying concepts on the basis of practical research, *in* 'Proceedings of the Software Technology and Engineering Practice conference', Pittsburgh, Pennsylvania, USA.

Briand, L. C., Differding, C. M. and Rombach, H. D. (1996), 'Practical Guidelines for Measurement-Based Process Improvement', *Software Process – Improvement and Practice* **2**(4), 253–280.

Briand, L., Kim, Y.-M., Melo, W., Seaman, C. and Basili, V. R. (1998), 'Q-MOPP: qualitative evaluation of maintenance organizations, processes and products', *Journal of Software Maintenance: Research and Practice* **10**(4), 249–278.

Budlong, F. and Peterson, J. (1995), Software Metrics Capability Evaluation Guide, Technical report, The Software Technology Support Center (STSC). Version 2.0.

Byrnes, P. and Phillips, M. (1996), Software Capability Evaluation – Version 3.0, Method Description, Technical Report CMU/SEI-96-TR-002, Software Engineering Institute/Carnegie Mellon University.

Central Computer and Telecommunications Agency (1992*a*), *Information Technology Infrastructure Library*, HSMO Books, London, UK.

Central Computer and Telecommunications Agency (1992*b*), *The IT Infrastructure Library - An Introduction*, in .

Comer, P. and Chard, J. (1993), 'A measurement maturity model', *Software Quality Journal* **2**(4), 277–289.

Conte, S., Dunsmore, H. and Shen, V. (1986), *Software Engineering Metrics and Models*, The Benjamin/Cummings Publishing Company, Inc.

Curtis, B., Hefley, W. E. and Miller, S. (1995*a*), Overview of the People Capability Maturity Model, Technical Report CMU/SEI-95-MM-01, Software Engineering Institute/Carnegie Mellon University.

Curtis, B., Hefley, W. E. and Miller, S. (1995*b*), People Capability Maturity Model, Technical Report CMU/SEI-95-MM-02, Software Engineering Institute/Carnegie Mellon University.

Daskalantonakis, M. K. (1994), 'Achieving Higher SEI Levels', *IEEE Software* **11**(4), 17–24.

Daskalantonakis, M. K., Yacobellis, R. H. and Basili, V. R. (1990-1991), 'A Method for Assessing Software Measurement Technology', *Quality Engineering* **3**, 27–40.

Diaz, M. and Sligo, J. (1997), 'How Software Process Improvement Helped Motorola', *IEEE Software* **14**(5), 75–81.

Dion, R. (1993), 'Process Improvement and the Corporate Balance Sheet', *IEEE Software* **10**(4), 28–35.

Drew, D. W. (1992), Tailoring the Software Engineering Institute's (SEI) Capability Maturity Model (CMM) to a Software Sustaining Engineering Organization, *in* 'Proceedings of the Conference on Software Maintenance', IEEE Computer Society, Orlando, Florida, pp. 137–144.

El Emam, K. (1998), The Internal Consistency of the ISO/IEC 15504 Software Process Capability Scale, *in* 'Proceedings of the Fifth International Software

Metrics Symposium', IEEE Computer Society TCSE, IEEE Computer Society Press, Bethesda, Maryland, USA, pp. 72–81.

El Emam, K. (1999), 'Benchmarking Kappa: Interrater Agreement in Software Process Assessments', *Empirical Software Engineering: an International Journal* **4**(2), 113–133.

El Emam, K., Briand, L. and Smith, R. (1996), 'Assessor Agreement in Rating SPICE Processes', *Software Process – Improvement and Practice* **2**(4), 291–306.

El Emam, K., Drouin, J.-N. and Melo, W., eds (1998), *SPICE: The Theory and Practice of Software Process Improvement and Capability Determination*, IEEE Computer Society.

El Emam, K. and Goldenson, D. R. (1996), 'An Empirical Evaluation of the Prospective International SPICE Standard', *Software Process – Improvement and Practice* **2**(2), 123–148.

El Emam, K., Simon, J.-M., Rousseau, S. and Jacquet, E. (1998), Cost Implications of Interrater Agreement for Software Process Assessments, *in* 'Proceedings of the Fifth International Software Metrics Symposium', IEEE Computer Society TCSE, IEEE Computer Society Press, Bethesda, Maryland, USA, pp. 38–51.

Fayad, M. E. and Laitinen, M. (1997), 'Process assessment considered wasteful', *Communications of the ACM* **40**(11), 125–128.

Fenton, N. E. (1991), *Software Metrics, a rigorous approach*, Chapman & Hall.

Fenton, N. E. and Pfleeger, S. L. (1997), *Software Metrics: A Rigorous and Practical Approach*, second edn, Int. Thomson Computer Press.

Fuggetta, A., Lavazza, L., Marasca, S., Cinti, S., Oldano, G. and Orazi, E. (1998), 'Applying GQM in an Industrial Software Factory', *ACM Transactions on Software Engineering and Methodology* **7**(4), 411–448.

Fusaro, P., El Emam, K. and Smith, B. (1998), 'The Internal Consistencies of the 1987 SEI Maturity Questionnaire and the SPICE Capability Dimension', *Empirical Software Engineering: an International Journal* **3**(2), 179–201.

Garcia, S. M. (1997), 'Evolving Improvement Paradigms: Capability Maturity Models and ISO/IEC 15504 (PDTR)', *Software Process – Improvement and Practice* **3**(1), 47–58.

Grady, R. B. (1992), *Practical software metrics for project management and process improvement*, Hewlett-Packard Professional Books, Prentice-Hall, Inc.

Grady, R. B. and Caswell, D. L. (1987), *Software Metrics: Establishing a company-wide program*, Prentice-Hall, Inc., Hewlett-Packard Company.

Gray, E. and Smith, W. (1998), 'On the limitations of software process assessment and the recognition of a required re-orientation for global process improvement', *Software Quality Journal* **7**(1), 21–34.

Grönroos, C. (1990), *Service Management and Marketing - Managing the Moments of Truth in Service Competition*, Lexington Books, Lexington, Mass. pp. 298.

Haase, V., Messnarz, R., Koch, G., Kugler, H. J. and Decrinis, P. (1994), 'Bootstrap: Fine-Tuning Process Assessment', *IEEE Software* **11**(4), 25–35.

Hall, T. and Fenton, N. (1997), 'Implementing Effective Software Metrics Programs', *IEEE Software* **14**(2), 55–65.

Herbsleb, J., Zubrow, D., Goldenson, D., Hayer, W. and Paulk, M. (1997), 'Software Quality and the Capability Maturity Model', *Communications of the ACM* **40**(6), 30–40.

Hetzel, W. (1993), *Making Software Measurement Work: Building an Effective Software Measurement Program*, QED Publishing, Boston, Massachusetts, chapter The Measurement Process, pp. 25–56.

Hevner, A. (1997), 'Phase containment metrics for software quality improvement', *Information and Software Technology* **39**(13), 867–877.

Hollenbach, C., Young, R., Pflugrad, A. and Smith, D. (1997), 'Combining Quality and Software Improvement', *Communications of the ACM* **40**(6), 41–45.

Horst, B. t., Niessink, F. and van Vliet, H. (1999), Implementing a Quantitatively Controlled Software Process, *in* R. Kusters, A. Cowderoy, F. Heemstra and E. van Veenendaal, eds, 'Proceedings of the combined 10th European Software Control and Metrics Conference (ESCOM) and the 2nd SCOPE conference on Software Product Evaluation', Shaker Publishing, Herstmonceux, England, pp. 167–175.

Humphrey, W. S., Snyder, T. R. and Willis, R. R. (1991), 'Software Process Improvement at Hughes Aircraft', *IEEE Software* **8**(4), 11–23.

ISO (1987*a*), *ISO 9000 Quality management and quality assurance standards – Guidelines for selection and use.* First edition.

ISO (1987*b*), *ISO 9001 Quality systems – Model for quality assurance in design/development, production, installation and servicing.* First edition.

ISO (1987*c*), *ISO 9002 Quality systems – Model for quality assurance in production and installation.* First edition.

ISO (1987*d*), *ISO 9003 Quality systems – Model for quality assurance in final inspection and test.* First edition.

ISO (1990), *ISO 10011-1: Guidelines for auditing quality system.*

ISO (1991), *ISO 9000-3 Quality management and quality assurance standards – Part 3: Guidelines for the application of ISO 9001 to the development, supply and maintenance of software*.

ISO/IEC (1995*a*), *ISO/IEC 9126-1: Information Technology - Software quality characteristics and metrics.* Working Draft Version 3.2.

ISO/IEC (1995*b*), *ISO/IEC 9126-2: Information Technology - Software quality characteristics and metrics - External metrics.* Working Draft Version 4.3.

Jeffery, R. and Berry, M. (1993), A Framework for Evaluation and Prediction of Metrics Program Success, *in* 'Proceedings of the First International Software Metrics Symposium', IEEE Computer Society TCSE, IEEE Computer Society Press, pp. 28–39.

Jeffery, R. and Stathis, J. (1996), 'Function Point Sizing: Structure, Validity and Applicability', *Empirical Software Engineering: an International Journal* **1**(1), 11–30.

Jørgensen, M. (1995), 'An Empirical Study of Software Maintenance Tasks', *Journal of Software Maintenance: Research and Practice* **7**, 27–48.

Kemerer, C. (1987), 'An empirical validation of software cost estimation models', *Communications of the ACM* **30**(5), 416–429.

Kemerer, C. and Porter, B. (1992), 'Improving the Reliability of Function Point Measurement: An Empirical Study', *IEEE Transactions on Software Engineering* **18**(11), 1011–1024.

Kitchenham, B. A., Travassos, G. H., von Mayrhauser, A., Niessink, F., Schneidewind, N. F., Singer, J., Takada, S., Vehvilainen, R. and Yang, H. (1999), 'Towards an ontology of software maintenance', *Journal of Software Maintenance: Research and Practice* **11**(6), 365–389.

Kitchenham, B., Pfleeger, S. L. and Fenton, N. (1995), 'Towards a Framework for Software Measurement Validation', *IEEE Transactions on Software Engineering* **21**(12), 929–944.

Kuvaja, P., Similä, J., Krzanik, L., Bicego, A., Koch, G. and Saukkonen, S. (1994), *Software Process Assessment and Improvement: The BOOTSTRAP Approach*, Blackwell Publishers.

Latum, F. v., van Solingen, R., Oivo, M., Hoisl, B., Rombach, D. and Ruhe, G. (1998), 'Adopting GQM-Based Measurement in an Industrial Environment', *IEEE Software* **15**(1), 78–86.

McFeeley, B. (1996), IDEAL^{SM}: A User's Guide for Software Process Improvement, Handbook CMU/SEI-96-HB-001, Software Engineering Institute/Carnegie Mellon University.

McGarry, F., Burke, S. and Decker, B. (1998), Measuring the Impacts Individual Process Maturity Attributes Have on Software Products, *in* 'Proceedings of the Fifth International Software Metrics Symposium', IEEE Computer Society TCSE, IEEE Computer Society Press, Bethesda, Maryland, USA, pp. 52–60.

Montgomery, D. (1985), *Introduction to Statistical Quality Control*, John Wiley & Sons.

Musa, J. D., Iannino, A. and Okumoto, K. (1987), *Software Reliability - Measurement, Prediction, Application*, McGraw-Hill Series in Software Engineering and Technology, McGraw-Hill Book Company.

NEFPUG (1993), Functiepuntanalyse in onderhoud – functionele en technische richtlijnen, Technical report, NEFPUG (Vereniging Nederlandse Functie Punt Gebruikers). Version 0.1.

Niessink, F. (1998), Software Maintenance from a Service Perspective, *in* Pfleeger and Rosenberg (1998), pp. 25–26.

Niessink, F. and van Vliet, H. (1997), Predicting Maintenance Effort with Function Points, *in* M. J. Harrold and G. Visaggio, eds, 'International Conference on Software Maintenance', IEEE Computer Society, Bari, Italy, pp. 32–39.

Niessink, F. and van Vliet, H. (1998*a*), 'Towards Mature IT Services', *Software Process – Improvement and Practice* **4**(2), 55–71.

Niessink, F. and van Vliet, H. (1998*b*), Towards Mature Measurement Programs, *in* P. Nesi and F. Lehner, eds, 'Proceedings of the Second Euromicro Conference on Software Maintenance and Reengineering', IEEE Computer Society, Florence, Italy, pp. 82–88.

Niessink, F. and van Vliet, H. (1998*c*), Two Case Studies in Measuring Software Maintenance Effort, *in* T. M. Khoshgoftaar and K. Bennett, eds, 'International Conference on Software Maintenance', IEEE Computer Society, Bethesda, Maryland, USA, pp. 76–85.

Niessink, F. and van Vliet, H. (1999*a*), A Pastry Cook's View on Software Measurement, *in* R. Dumke and A. Abran, eds, 'Software Measurement - Current Trends in Research and Practice', Deutscher Universitäts Verlag, Wiesbaden, Germany, pp. 109–125.

Niessink, F. and van Vliet, H. (1999*b*), Measurements Should Generate Value, Rather Than Data, *in* 'Proceedings of the Sixth International Software Metrics Symposium', IEEE Computer Society TCSE, IEEE Computer Society Press, Boca Raton, Florida, USA, pp. 31–38.

Niessink, F. and van Vliet, H. (1999*c*), The IT Service Capability Maturity Model, Technical Report IR-463, Division of Mathematics and Computer Science, faculty of Sciences, Vrije Universiteit Amsterdam. Model version L2-1.0.

Niessink, F. and van Vliet, H. (2000), 'Software Maintenance from a Service Perspective', *Journal of Software Maintenance: Research and Practice* **12**(?). To appear.

Offen, R. J. and Jeffery, R. (1997), 'Establishing Software Measurement Programs', *IEEE Software* **14**(2), 45–53.

Olson, T. G., Humphrey, W. S. and Kitson, D. (1989), Conducting SEI-Assisted Software Process Assessments, Technical Report CMU/SEI-89-TR-007, Software Engineering Institute/Carnegie Mellon University.

Onoma, A., Tsai, W., Tsunoda, F., Suganuma, H. and Subramanian, S. (1995), 'Software Maintenance – an Industrial Experience', *Journal of Software Maintenance: Research and Practice* **7**, 333–375.

Ould, M. A. (1996), 'CMM and ISO 9001', *Software Process – Improvement and Practice* **2**(4), 281–289.

Panfilis, S. D., Kitchenham, B. and Morfuni, N. (1997), 'Experiences introducing a measurement program', *Information and Software Technology* **39**(11), 745–754.

Park, R. E. (1992), Software Size Measurement: A Framework for Counting Source Statements, Technical Report CMU/SEI-92-TR-020, Software Engineering Institute/Carnegie Mellon University.

Park, R., Goethert, W. and Florac, W. (1996), Goal-Driven Software Measurement – A Guidebook, Technical Report CMU/SEI-96-HB-002, Software Engineering Institute/Carnegie Mellon University.

Paulk, M. C. (1995), 'How ISO 9001 compares with the CMM', *IEEE Software* **12**(1), 74–83.

Paulk, M. C., Curtis, B., Chrissis, M. B. and Weber, C. V. (1993), Capability Maturity Model for Software, Version 1.1, Technical Report CMU/SEI-93-TR-024, Software Engineering Institute/Carnegie Mellon University.

Paulk, M. C., Garcia, S. M. and Crissis, M. B. (1996), An Architecture for CMM Version 2?, *in* 'Proceedings of the Software Engineering Process Group Conference', Atlantic City, NJ.

Paulk, M. C., Konrad, M. D. and Garcia, S. M. (1995), 'CMM Versus SPICE Architectures', *Software Process Newsletter, IEEE Computer Society Technical Council on Software Engineering* (3), 7–11.

Paulk, M. C., Weber, C. V., Garcia, S., Chrissis, M. B. and Bush, M. (1993), Key Practices of the Capability Maturity Model, Version 1.1, Technical Report CMU/SEI-93-TR-025, Software Engineering Institute/Carnegie Mellon University.

Pfleeger, S. L. (1993), 'Lessons Learned in Building a Corporate Metrics Program', *IEEE Software* **10**(3), 67–74.

Pfleeger, S. L. (1995), 'Maturity, Models and Goals: How to Build a Metrics Plan', *The Journal of Systems and Software* **31**(2), 143–155.

Pfleeger, S. L. and McGowan, C. (1990), 'Software Metrics in the Process Maturity Framework', *The Journal of Systems and Software* **12**(3), 255–261.

Pfleeger, S. L. and Rosenberg, J., eds (1998), *Proceedings of the Fourth Workshop on Empirical Studies of Software Maintenance*, Bethesda, Maryland USA.

Pigoski, T. M. (1996), *Practical Software Maintenance: Best Practices for Managing Your Software Investment*, John Wiley & Sons. 384 pp.

Pitt, L. F., Watson, R. T. and Kavan, C. (1995), 'Service Quality: A Measure of Information Systems Effectiveness', *Management Information Systems Quarterly* **19**(2), 173–187.

Pulford, K., Kuntzmann-Combelles, A. and Shirlaw, S. (1996), *A Quantitative Approach To Software Management – The Ami Handbook*, Addison Wesley.

Rifkin, S. and Cox, C. (1991), Measurement in Practice, Technical Report SEI-91-TR-16, Software Engineering Institute/Carnegie Mellon University.

Rijsenbrij, D., Kemperman, E., van Vliet, J. and Trienekens, J. (1994), 'Concrete-Kit – Concretisering van Kwaliteitsbeheersing en -verbetering: naar een nieuwe generatie IT-tools'. Subsidieaanvraag in het kader van de informatietechnologie-programma's 1994, vastgesteld door de Minister van Economische Zaken.

Rijsenbrij, D., van Veen, L., Beekman, J., Trienekens, J., van Vliet, J. and Looijen, M. (1996), 'KWINTES – KWantificeren van INformatieTEchnologie-infrastructuur Services'. Subsidieaanvraag in het kader van de informatietechnologie-programma's 1996, vastgesteld door de Minister van Economische Zaken.

Rout, T. P. (1995), 'SPICE: A Framework for Software Process Assessment', *Software Process – Improvement and Practice* **1**(Pilot Issue), 57–66.

Ruijs, L., de Jong, W., Niessink, F. and Trienekens, J. (2000), *Op weg naar volwassen ICT dienstverlening: Resultaten van het onderzoeksproject Kwintes*, Academic Service. To be published.

Schmauch, C. H. (1995), *ISO 9000 for Software Developers*, revised edn, ASQC Quality Press.

Schneidewind, N., Kitchenham, B., Niessink, F., Singer, J., von Mayrhauser, A. and Yang, H. (1999), Panel 1: Resolved: "Software Maintenance Is Nothing More Than Another Form of Development", *in* 'International Conference on Software Maintenance', IEEE Computer Society, Oxford, England, pp. 63–64.

SEI (1995), *The Capability Maturity Model: Guidelines for Improving the Software Process*, SEI Series in Software Engineering, Addison-Wesley Publishing Company. Carnegie Mellon University/Software Engineering Institute.

Selby, R., Porter, A., Schmidt, D. and Berney, J. (1991), Metric-driven Analysis and Feedback Systems for Enabling Empirically Guided Software Development, *in* 'Proceedings of the 13th International Conference on Software Engineering', pp. 288–298.

Shepperd, M., Schofield, C. and Kitchenham, B. (1996), Effort Estimation Using Analogy, *in* 'Proceedings of the 18th International Conference on Software Engineering', IEEE Computer Society Press, Berlin, Germany, pp. 170–178.

Simon, J.-M., El Emam, K., Rousseau, S., Jacquet, E. and Babey, F. (1997), 'The Reliability of ISO/IEC PDTR 15504 Assessments', *Software Process – Improvement and Practice* **3**(3), 177–188.

Singer, J. (1998), Practices of Software Maintenance, *in* T. M. Khoshgoftaar and K. Bennett, eds, 'International Conference on Software Maintenance', IEEE Computer Society, Bethesda, Maryland, USA, pp. 139–145.

Solingen, R. v. and Berghout, E. (1999), *The Goal/Question/Metric Method: a practical guide for quality improvement of software development*, McGraw-Hill.

Solingen, R. v., Berghout, E. and Latum, F. v. (1998), 'Interrupts: just a minute never is', *IEEE Software* **15**(5), 97–103.

Solingen, R. v., Lattum, F. v., Oivo, M. and Berkhout, E. (1995), Application of software measurement at Schlumberger RPS – Towards enhancing GQM, *in* 'Proceedings of the Sixth European Software COst Modeling Conference (ESCOM)', Rolduc, The Netherlands.

Stålhane, T., Borgersen, P. and Arnesen, K. (1997), 'In Search of the Customer's Quality View', *The Journal of Systems and Software* **38**(1), 85–93.

Stark, G. E. and Oman, P. W. (1997), 'Software maintenance management strategies: observations from the field', *Journal of Software Maintenance: Research and Practice* **9**(6), 365–378.

Symons, C. (1988), 'Function point analysis: difficulties and improvements', *IEEE Transactions on Software Engineering* **14**(1), 2–11.

Thomson, H. E. and Mayhew, P. (1997), 'Approaches to Software Process Improvement', *Software Process – Improvement and Practice* **3**(1), 3–17.

Trienekens, J., Zwan, M. v. d., Niessink, F. and Vliet, J. v. (1997), *De SLA Specificatiemethode*, Cap Gemini Perform Service Management, Academic Service. (In Dutch).

Trillium (1996), Trillium Model – For Telecom Product Development and Support Process Capability, Model Issue 3.2, Bell Canada.

Vliet, H. v. (2000), *Software Engineering: principles and practice*, second edn, John Wiley & Sons. To be published.

Watson, R. T., Pitt, L. F. and Kavan, C. (1998), 'Measuring Information Systems Service Quality: Lessons from Two Longitudinal Case Studies', *Management Information Systems Quarterly* **22**(1), 61–79.

West, R. (1996), 'Book Review: Improving the Maintainability of Software', *Journal of Software Maintenance: Research and Practice* **8**(5), 345–356.

Zahran, S. (1997), *Software Process Improvement: Practical Guidelines for Business Success*, Addison Wesley.

Zeithaml, V. A. and Bitner, M. J. (1996), *Services Marketing*, McGraw-Hill Series in Marketing, McGraw-Hill.

Zeithaml, V. A., Parasuraman, A. and Berry, L. L. (1990), *Delivering Quality Service: Balancing Customer Perceptions and Expectations*, The Free Press, Macmillan Inc.

Zelkowitz, M. V. and Wallace, D. (1997), 'Experimental validation in software engineering', *Information and Software Technology* **39**(11), 735–743.

Zelkowitz, M. V. and Wallace, D. R. (1998), 'Experimental Models for Validating Technology', *IEEE Computer* **31**(5), 23–31.

Zubrow, D., Hayes, W., Siegel, J. and Goldenson, D. (1994), Maturity Questionnaire, Technical Report CMU/SEI-94-SR-7, Software Engineering Institute/Carnegie Mellon University.

Zuse, H. (1998), *A Framework of Software Measurement*, Walter de Gruyter.

Samenvatting

Perspectieven op het verbeteren van software onderhoud

Dit proefschrift behandelt een gedeelte van het onderzoek gedaan tijdens de projecten Concrete Kit en Kwintes. Deze projecten hadden als doel om:

- kwantitatief, objectief en fundamenteel inzicht te verkrijgen in de kwaliteit van onderhoud en beheer van IT produkten, en om
- methoden, technieken en tools te ontwikkelen die het onderhoud en beheer van IT producten ondersteunen.

In dit proefschrift behandelen we twee van de vier binnen de projecten behandelde onderzoeksgebieden, namelijk (1) het gebruik van metingen en meetprogramma's ter ondersteuning en verbetering van IT beheer en onderhoud – meetgebaseerd verbeteren, en (2) het concept 'volwassenheid van IT dienstverleners' – volwassenheids-gebaseerd verbeteren. Om het onderzoek beter hanteerbaar te maken is gekozen voor de dienst software onderhoud als specifiek aandachtsgebied.

In deel I van dit proefschrift wordt het eerste onderzoeksgebied behandeld – meet-gebaseerd verbeteren. We analyseren een viertal meetprogramma's die zijn geïmplementeerd in vier software onderhoud organisaties. Twee van deze meetprogramma's zijn door de betreffende organisaties zelf ingericht, de twee andere zijn met behulp van afstudeerders opgezet. We zien dat de meetprogramma's in succes variëren van zeer onsuccesvol tot succesvol. We denken dat het succes van een meetprogramma in ieder geval gedeeltelijk is te verklaren aan de hand van de kwaliteit van de meetprocessen die de organisatie heeft ingericht. In hoofdstuk 6 ontwikkelen we een volwassenheidsmodel dat beschrijft welke processen een organisatie zou moeten inrichten om beter te worden in het meten van haar software en software processen. Als eerste stap richting validatie van dat model vergelijken we in hoofdstuk 7 het model met andere literatuur die richtlijnen geeft op het gebied van het implementeren van meetprogramma's. Om deze vergelijking mogelijk

te maken ontwikkelen we een abstract model van het proces van meet-gebaseerd verbeteren. We gebruiken dit abstracte model om de verschillende richtlijnen met elkaar te vergelijken. Hieruit concluderen we dat er consensus bestaat over de basisvereisten voor meetprogramma's, maar dat er buiten die basis geen overeenstemming is, en dat er nauwelijks richtlijnen worden gegeven voor het daadwerkelijk *gebruiken* van de meetgegevens voor het tot stand brengen van verbeteringen in de organisatie. Dit betekent dat er, naast de gebruikelijke 'interne' succesfactoren – wat moet er geregeld zijn om goed te kunnen meten – ook 'externe' succesfactoren voor meetprogramma's zijn – wat moet er geregeld zijn om de meetgegevens toe te passen voor verbetering. Uit de verschillende situaties waarin meetprogramma's worden ingezet leiden we vier externe succesfactoren af. Tevens geven we aan hoe die succesfactoren door organisaties zouden kunnen worden vervuld.

Deel II behandelt het tweede onderzoeksgebied – volwassenheids-gebaseerd verbeteren. We beginnen in hoofdstuk 8 met een vergelijking van diensten en produkten in het algemeen, en hoe dit van toepassing is op software onderhoud en software ontwikkeling. We laten zien hoe klanten de kwaliteit van diensten beoordelen aan de hand van twee dimensies, de functionele en de technische kwaliteit. We gebruiken het 'gaps-model' van service kwaliteit om te verklaren hoe het verschil tussen verwachtingen en perceptie van de ontvangen dienst kan ontstaan. Daarnaast beschrijven we een aantal case studies tijdens welke deelresultaten van de Concrete Kit en Kwintes projecten werden getest. Deze theoretische en praktische overwegingen hebben geleid tot het formuleren van een volwassenheidsgroei model voor IT dienstverleners, IT Service Capability Maturity Model (IT Service CMM) genaamd. In hoofdstuk 9 zijn het doel van dit model, de structuur en de processen beschreven. In hoofdstuk 10 beschrijven we een tweetal case studies tijdens welke het IT Service CMM is toegepast bij het beoordelen en verbeteren van de processen van twee IT dienstverleners.

Het laatste hoofdstuk van dit proefschrift behandelt de in hoofdstuk 1 opgestelde onderzoeksvragen, relateert de resultaten zoals beschreven in de overige hoofdstukken aan de onderzoeksvragen en geeft richtingen voor mogelijk toekomstig onderzoek.

SIKS Dissertatiereeks

1998-1 Johan van den Akker (CWI)
DEGAS - An Active, Temporal Database of Autonomous Objects
promotor: prof.dr. M.L. Kersten (CWI/UvA)
co-promotor: dr. A.P.J.M. Siebes (CWI)
promotie: 30 maart 1998

1998-2 Floris Wiesman (UM)
Information Retrieval by Graphically Browsing Meta-Information
promotores: prof.dr.ir. A. Hasman (UM)
prof.dr. H.J. van den Herik (UM/RUL)
prof.dr.ir. J.L.G. Dietz (TUD)
promotie: 7 mei 1998

1998-3 Ans Steuten (TUD)
A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective
promotores: prof.dr.ir. J.L.G. Dietz (TUD)
prof.dr. P.C. Hengeveld (UvA)
promotie: 22 juni 1998

1998-4 Dennis Breuker (UM)
Memory versus Search in Games
promotor: prof.dr. H.J. van den Herik (UM/RUL)
promotie: 16 oktober 1998

1998-5 E.W.Oskamp (RUL)
Computerondersteuning bij Straftoemeting
promotores: prof.mr. H. Franken
prof.dr. H.J. van den Herik
promotie: 13 mei 1998

1999-1 Mark Sloof (VU)
Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products
promotor: prof.dr. J. Treur
co-promotor: dr.ir. M. Willems
promotie: 11 mei 1999

1999-2 Rob Potharst (EUR)
Classification using decision trees and neural nets
promotor: prof.dr. A. de Bruin
co-promotor: dr. J.C. Bioch
promotie: 4 juni 1999

1999-3 Don Beal (Queen Mary and Westfield College)
The Nature of Minimax Search
promotor: prof.dr. H.J. van den Herik
promotie: 11 juni 1999

1999-4 Jacques Penders (KPN Research)
The practical Art of Moving Physical Objects
promotor: prof.dr. H.J. van den Herik
co-promotor: dr. P.J. Braspenning
promotie: 11 juni 1999

1999-5 Aldo de Moor (KUB)
Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems
promotor: prof.dr. R.A. Meersman
co-promotor: dr. H. Weigand
promotie: 1 oktober 1999

1999-6 Niek J.E. Wijngaards (VU)
Re-design of compositional systems
promotor: prof.dr. J. Treur
co-promotor: dr. F.M.T. Brazier
promotie: 30 september 1999

1999-7 David Spelt (UT)
Verification support for object database design
promotor: prof.dr. P.M.G. Apers
assistent promotor: dr. H. Balsters
promotie: 10 september 1999

1999-8 Jacques H.J. Lenting (UM)
Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation
promotor: prof.dr. H.J. van den Herik
co-promotor: dr. P.J. Braspenning
promotie: 3 december 1999

2000-1 Frank Niessink (VU)
Perspectives on Improving Software Maintenance
promotor: prof.dr. J.C. van Vliet
promotie: 28 maart 2000