Product focused software process improvement : SPI in the embedded software domain

Product Focused
Software Process Improvement

SPI in the embedded software domain

Rini van Solingen

Eindhoven University of Technology, The Netherlands **ISBN 90-386-0613-3**

CIP-DATA LIBRARY EINDHOVEN UNIVERSITY OF TECHNOLOGY

Solingen, Rini van

Product Focused Software Process Improvement: SPI in the embedded software domain

by Rini van Solingen – Eindhoven:

Technische Universiteit Eindhoven, 2000

Proefschrift

ISBN 90-386-0613-3

NUGI 684

Keywords: Software Quality / Software Process Improvement / SPI / Embedded Software / Product focused SPI / Goal Question Metric / GQM

Printed by Eindhoven University Press

English editing by: Miranda Aldham-Breary

Product Focused Software Process Improvement

SPI in the Embedded Software Domain

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van
de Rector Magnificus, prof.dr. M. Rem, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen op
donderdag 2 maart 2000 om 16.00 uur

door

Dirk Marinus van Solingen

geboren te Middelburg

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr. T.M.A. Bemelmans

en

prof.dr.ir. A.C. Brombacher

Copromotor:

dr.ir. J.J.M. Trienekens

Preface

The difficulties of creating high quality embedded products are addressed in this thesis. Embedded products are integrated into daily life at all kinds of levels and in all kinds of situations. Examples of embedded products include flight control systems, air bags, ATMs, mobile phones, televisions, fuel pumps, and infusion monitors. The quality of these products is an important topic, as quality has direct consequences for people's lives, financial situation and happiness. Although most people support the importance of quality products, our ability to create or guarantee high quality embedded products is not very well established. Many products are being developed, but their level of quality is often more due to luck than judgement and sound engineering.

An approach is introduced in this thesis to manage product quality. Explicit, product driven, customisations of the development process are recommended in tandem with learning the effects on product quality of these changes. A conceptual model of this approach is presented and guidelines that can be used to apply this approach in practice are given. The thesis is rounded of with case-studies, detailing the application of the recommended approach in an industrial setting.

With this thesis the work in this field is not finished. On the contrary; one of the conclusions presented in this thesis is that it is currently unclear what the effects on product quality are of most of the methods, techniques, tools, etc. that are being applied nowadays to make embedded software products. With this thesis a modest first step is taken towards a situation in which creating embedded product quality becomes really a matter of 'engineering': a matter of taking specific actions and measuring the results of these actions. Thus I hope to contribute, be it a small step, towards a situation in which it is possible to design a specific development process for each embedded product that results in a guaranteed level of product quality. I sincerely hope that the work presented in this thesis will be adopted by other people both in industry and academia, so that the work towards this long term objective will be continued.

Acknowledgements

This thesis was created over a period of four years, and is based on work that was definitely not carried out by myself alone. I, therefore, want to use the opportunity to thank a number of people for their contributions and support. Firstly, my Ph.D. project committee: Prof. Dr. Theo Bemelmans, Prof. Dr. Ir. Aarnout Brombacher, Prof. Dr. Rob Kusters, and Dr. Ir. Jos Trienekens, without whom this work would never have been started, carried out, or finished.

Secondly, I would like to thank my supervisor at Schlumberger RPS/Tokheim: Erik Rodenbach, who provided me with a context in which I could freely test all the ideas, concepts and guidelines developed throughout the whole period. I want to thank him for this opportunity, his supervision and support, and also thank him for the fun we had sharing an office. In addition, I thank all my Schlumberger RPS/Tokheim colleagues who participated in the work presented in this thesis. It is impossible to thank everyone individually, however, in particular I want to thank: Frank Simons, Wim van der Bijl, Henry van den Boogaert, Tjeerd Böttcher, Frans Heesters, Marc Lessage, Johan Couwenberg, Frans '$narfø' van Beers and Erich Sigrist.

Three persons were mainly responsible for motivating me to start this Ph.D. project: Egon Berghout, Michiel van Genuchten and Frank van Latum. I am grateful for their support, insights, knowledge transfer and co-operation during the past years. I hope that we will continue this co-operation in the future. Egon receives a very special thanks. Not only for the very fruitful and productive co-operation of the past years, but also for his work during the editing process of the final draft of this thesis, which he did for me while I was taking a three month round the world holiday.

I want to thank all my colleagues involved in the (international) research and co-operation projects in which I participated and with whom I could share ideas. I am very grateful for their co-operation, they are: Andreas Birk, Pieter Derks, Janne Järvinnen, Frank van Latum, Prof. Markku Oivo, Günther Ruhe and Hans Wijnands. Furthermore I want to thank Adriana Bicego, Adrian Cowderoy, Christiane Gresse, Dirk Hamann, Jorma Hirvensalo, Barabara Hoisl, John Jenkins, Munish Khurana, Seija Komi-Sirvio, Pasi Kuvaja, Dietmar Pfahl, Prof. Dieter Rombach, Mattias Vierima and all those people I have not mentioned from the PROFES, SPIRITS, ESSI/CEMP and SPACE-UFO projects.

I thank all the master of science students, from Eindhoven and Delft University of Technology, who supported this research by bringing their graduation work in line with

my Ph.D. project: Roy de Jonge, Erik Kooiman, Hans Leliveld, Shyam Soerjoesing, Paul Stalenhoef, Arnim van Uijtregt, Stephan van Uijtregt and Niels van Veldhoven.

Thanks to my unforgettable university (ITQM) and room mates: Teade Punter, Erik van Veenendaal, Mark van der Zwan and Sjaak 'MP3' Bouman. Furthermore thanks to all my colleagues in the 'Information and Technology' section, especially to Frans 'lacheman' Mouws, Bas Vermeer, Roel van de Berg, Monique Jansen, Mark Eeuwe, Ramon Caanen, Tom Dolan, Remco Helms and Frank Berkers.

Special thanks are due to Ineke Withagen for helping me out with the final editing of the manuscript, and to Miranda Aldham-Breary for removing the Dutch from my English, pointing out all the 'personifications'. This thesis takes, however, full responsibility for any that remain.

Finally, thanks to all my friends and family. They supported me all along the journey, paid attention to all my (often boring) stories, and provided an important stimulus for me to finish this work. Their support has been a very important foundation of this end result, although they may not be aware of this.

A very special thanks goes to my parents, who equipped me with the notion that a solid education is a firm basis for a bright future. I want to express my gratitude by dedicating this thesis to them.

Last but definitely not least, I thank my wife Patricia for her endless support and confidence, and for regularly reminding me that I work to live, and not the other way around.

Many thanks to all of you!

Rini van Solingen

Eindhoven, The Netherlands, December 14, 1999

Table of Contents

1. Introduction and Problem Definition

The relationship between software development processes and embedded product quality is dealt with in this thesis. Software development processes are often changed in an attempt to increase product quality, costs and throughput time (duration); however, the exact impacts of the specific process changes on the quality of the product are then unclear. The aim of this thesis is to provide a solution towards product focused software process improvement, in such a way that the quality of an embedded product can be controlled through specific improvements of the software development process.

1.1 Embedded Products

Life today is heavily dependent on software. Examples of software applications include: word processors, spreadsheets, e-mail and Internet applications. When using such applications it is quite clear that one is working with software; however, there is also a large amount of software incorporated in (electronic) products and these are widely present in today's world. Such products include mechanical, hydraulic and electronic machinery with a processor and embedded memory chips. These chips contain certain control instructions, which is termed 'software'. Software for such products is commonly known as 'embedded software' and the whole product is termed an 'embedded product'.

Examples of embedded products include: cellular phones, televisions, microwave ovens, petrol-pumps, cars, coffee machines, clocks, medical equipment and payment terminals. Embedded products range from single products to mass produced items, from 1 dollar products to 1 million dollar products, from single user to thousand user products, from product life times of 3 months to several decades, from single input and output to multiple input and output products. Embedded products are used throughout society by many different users, for many different purposes and in many different domains.

An embedded product is defined in this thesis as a physical entity consisting of hardware and software that performs a function that results in specific output data, based on specific input data. This product fulfils a dedicated function for which it is of no interest to the user whether certain functionality is implemented in software or in hardware. Although the term 'embedded system' is also often used, the term is not used in this thesis explicitly, because 'systems' are also considered to contain people, experiences, procedures, etc., which are not addressed by this thesis when talking about an embedded product.

The characteristics of embedded products are [Downes and Goldsack 1982][Kündig 1986][Taramaa et al. 1997]:

- Long operation is required; operation of more than ten years is often standard.
- Device autonomy, the product fulfils an unchangeable autonomous function.
- Reliability of the software is crucial. A first version is not allowed to contain faults that cause fatal failures, since replacing software in thousands of already sold products is costly, if not impossible.
- Delivery of embedded products is time critical. Deadlines are important.
- Cost of the product is important for mass markets. Small product cost reductions may result in huge cost savings due to economy of scale.
- Low-level and high-level software implementation technologies co-exist.
- Many embedded products are subject to extreme environmental conditions, such as heat, humidity, temperature changes, etc.
- The amount of software incorporated in embedded products grows fast.
- The product consists of parallel functions and processors, combined with many input and output functions, all in real-time application.
- Maintainability and extendibility through new functions, technologies and interfaces are often required.
- Interfaces and communication with other systems or embedded products are extremely important.
- There is a close connection but strict separation between software and hardware inside the product. Users, however, have no need to distinguish this separation: users consider an embedded product to be one single system.

It is not necessary that a product complies to all these characteristics before it can be classified as an embedded product. Certain products can fall into the grey area, for which it is not fully clear whether they are an embedded or non-embedded product. Take for example a supermarket cashing system, which consists of software that runs on a personal computer. Some will argue that this is not an embedded system, because software can be added or replaced using the personal computer. Others will point to the dedicated use of this computer as a cashing device, which is not changeable by the user, and therefore classify it as an embedded product. Carrying out such discussions to reach agreement on whether a certain product is embedded or not, is not the purpose of this thesis. It is not the aim of this thesis to define a cross-boarder between embedded and non-embedded products, the intention is to resolve some problems that are clearly relevant in the embedded product domain. So, if there is a need to improve product quality then this thesis will be useful, and if the product at hand complies to (some of) the embedded product characteristics than the product can be considered to be an embedded product.

1.1.1 Embedded software

'Embedded software is software, which determines the functionality of microprocessors and other programmable devices that are used to control electronic, electrical and electromechanical equipment and sub-systems. The programmable devices are often 'invisible' to the user' [TickIT 1995].

The role of embedded software in products and services is increasing tremendously. Software design is becoming *the* most effort-consuming task during the development of embedded products. Take for example a television set: the effort spent to develop a new generation of televisions has been shown to consist for more than 70% of software development resources [Rooijmans et al. 1996]. Or take for example cellular GSM phones: currently over 1 Mbyte of software is included in a GSM phone (Figure 1-1). This amount of software shows an increase by a factor 10 for cellular phones over the last 12 years [Karjalainen et al. 1996]. Note that this is a factor 10 for each 1000 days. This is both a *large* and *fast* increase in the amount of embedded software.

Generation	System type	Example system	Software size
1984: 1st	Analogue	Nordic mobile phone system (NMT)	some Kbytes
1988: 2nd	Analogue	NMT	tens of Kbytes
1992: 1st	Digital	Global system for mobile telecommunications (GSM)	hundreds of Kbytes
1996: 2nd	Digital	GSM	about one million bytes

Figure 1-1: Role of embedded software in mobile phones [Karjalainen et al. 1996]

A powerful benefit of software is that it creates the possibility to provide products with functionality that was not feasible before. Technically, almost all the features could be built purely in hardware, however the costs for such features was simply too high, software changes this; additional features that can be implemented using software do not increase the production costs of an embedded product. The costs of the hardware remains the same, but the total amount of functionality that can be sold increases, and software (once developed) costs nearly nothing.

A benefit of embedded software is also that hardware problems or hardware defects can be resolved using software solutions. Design mistakes in hardware are expensive to resolve, as designs must be updated, production settings must be changed or dedicated chips must be designed and produced. Software is, therefore, often used to fix hardware problems.

The increase in software application implies a rigorous change in the development of embedded products. There is a shift going on from mainly hardware product development to mainly software product development. Embedded product development is not just hardware development with some added software. Embedded product development

becomes mainly software development with some hardware development going on in parallel. This change, impacts organisations that develop embedded products highly. Past knowledge on embedded product development becomes obsolete, while knowledge on software development is lacking. Although many techniques are available to develop software, their applicability in the embedded product domain is still unclear. The past excellence of industry in developing high quality products is likely to deteriorate, because the way of working will change drastically. The embedded product industry must learn to manage the new situation by increasing their focus on software development and finding out how to be successful in this area. Guaranteeing high quality products is becoming more and more difficult as the application of embedded software increases.

The amount of functionality and complexity of software in embedded products is growing rapidly. Apparently this is not a gradual change, but a paradigm shift [Gal and Genuchten 1996] [Sol 1995]: the electronics industry is becoming a software industry. Suppliers that have many decades experience in their traditional market domain, such as automobiles, consumer electronics, or petroleum retail, are increasingly becoming ‘software suppliers’. Most suppliers, who until recently developed dedicated products for a dedicated market, are confronted with new players in their market that have no history in that domain. Embedded products bring about a move towards open systems [Gal and Genuchten 1996], in which software can be sold independently from the hardware. In the future it might be likely to buy a television by buying the hardware from one supplier, the audio control software from a second, and the Internet and video software from a third.

1.2 Embedded Product Quality

The quality of embedded products is a relevant topic as more embedded software is incorporated in life-vital applications: energy supply, transport, telecommunications, health care, etc. The quality of embedded products is based on the quality of the hardware and the quality of the software; however, good hardware quality and good software quality is no guarantee for overall quality of the embedded product. The whole is more than the sum of its separate parts.

In the above paragraphs it has been indicated that embedded product quality is a relevant topic for research because society is highly depending on the quality of embedded products. Take for example pacemakers, cars, aeroplanes and powerplants as examples of embedded products and the relevance of quality embedded products becomes clear. Society and peoples lives often depend highly on good quality embedded products. Furthermore, the quality of these products is at risk, because time and costs are easier to measure and manage, and this can cause product quality to be at risk.

The emphasis on embedded product quality is often refocused onto the development process of these products. The reason for this is that quality is created during a development process. Quality is not something that happens by accident, nor can it be brought in afterwards [Gilb 1994] [Humphrey 1989]. Specific actions need to be taken during the development process to create quality embedded products. The focus on creating quality embedded products is therefore often on managing the development process in an appropriate way.

Managing the development of quality embedded products is both a relevant and difficult issue. Some reasons for this are:

- Embedded product development is a complex discipline. It is a combination of several engineering disciplines: mechanics, hydraulics, electronics and informatics, but also ergonomics, marketing and economics. Combining disciplines means that experts from different fields must cooperate and this may cause communication difficulties and mistakes with consequent lower quality.
- Product quality is difficult to measure. Few objective embedded product quality metrics exists. In some cases, quality is defined as the number of defects found in the product, which is just one indicator of product quality. Even though measuring embedded product defects is already a difficult task in practice, product quality is much more than just defects, for which appropriate metrics and norms are not available and therefore not used.
- Product quality is difficult to define. Many subjective opinions are involved in deciding what is good quality or not. Product quality is multi dimensional: it means different things for different people. Capturing all these views on product quality into one single definition is often impossible.
- Traditional process quality approaches do not work. Such repeatable approaches mainly come from manufacturing where ‘statistical process control’ is used to manage product quality; however, embedded product development, and especially embedded software development, does not follow a repeatable process. The process of developing an embedded product is different every time. Direct links between process and product quality have only occasionally been proved; this relationship is mainly an assumption.
- Rapid technological innovation makes it impossible to learn technologies thoroughly. Once a new technology has been applied for the first time and its characteristics have been learned, new technologies are adopted, and the knowledge of the previous technology may become obsolete. This does not mean that adopting new technologies is wrong. It simply identifies that it is difficult to manage product quality when the characteristics of a technology are unknown.

- Product costs and delivery times are often subject to discussion and negotiation, while quality is included as a pre-requisite; however, cutting costs and shortening development cycles also impact process actions that are linked to product quality. The impact of costs and time on quality however, are unclear, which complicates the trade-off discussion.

In current software engineering theory and practice solutions to embedded product quality are sought by focusing on the development process. This is a legitimate way of working, since the development process creates quality; however, the detailed impact of the process on product quality is unknown.

1.3 Problem definition and research objective

The problem addressed in this thesis is the quality issue of embedded software. This issue is related to defining embedded product quality, developing such products, and measuring whether a quality product is being developed or not.

In this thesis the relationship between the development process and product quality is investigated. Investigating the way in which the relationship between the process view and the product view exist, was expected to lead to a better managerial concept for embedded product quality.

The objective of this research was to develop a conceptual model for process improvement for embedded product development. Starting from this conceptual model a set of guidelines will be proposed that support improving embedded product quality. Using this conceptual model and the guidelines, the software development processes can be configured and tuned in such a way that this process will contribute most efficiently and effectively to the product quality objectives.

1.4 Conclusions and thesis outline

Embedded products are integrated into daily life on all kinds of levels and in all kinds of situations. The quality of these products is an important topic, as it has direct consequences for people’s lives, financial situation and happiness. Although the importance of good quality embedded products is supported by most people, creating or guaranteeing quality is not well established. It is the intention that this thesis will contribute to this problem by developing an approach to manage product quality through explicitly making product-driven customisations to the development process.

Figure 1-2 depicts the outline of this thesis. The thesis is divided into five parts:

- Part 1: Problem identification and research approach
- Part 2: Process and product quality improvement
- Part 3: Concepts for product focused SPI
- Part 4: Practical application of product focused SPI
- Part 5: Conclusions

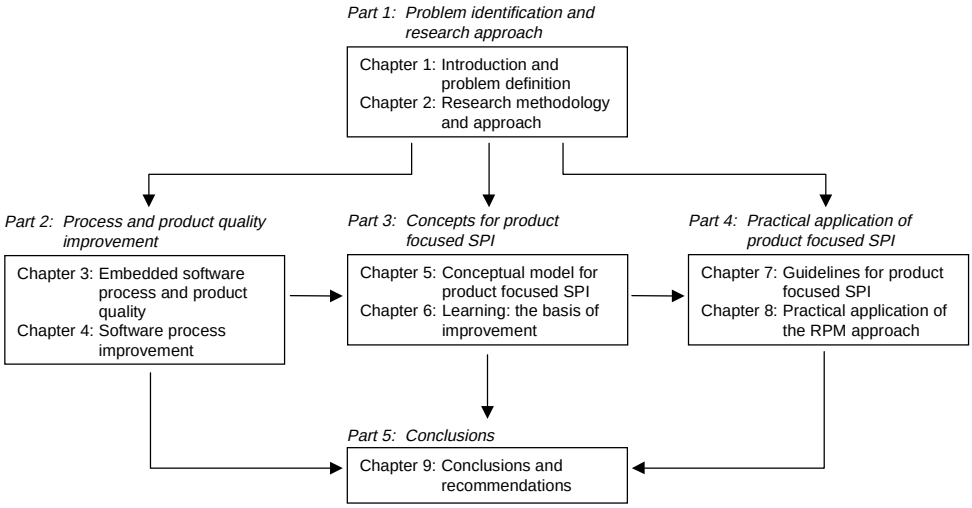


Figure 1-2: Outline of this thesis

In the first part of this thesis, an initial exploration of the problems involved with creating high quality embedded products is provided. Furthermore, the approach used to structure this research is presented, and the decisions made to establish validity of the research results are discussed. The problem addressed in this thesis is based on signals from industry and insights from literature. It identifies the difficulty of creating embedded software and embedded product quality and identifies the need to explicitly link process quality to product quality (Chapter 1). The approach of this research is classified as 'applied research' with the aim to interfere with industrial practice to solve the signalled problems. A set of case-studies is used, for the validation of the designed solution, which is expected to result in externally and internally valid results (Chapter 2).

Quality improvement of embedded software products and development processes is the theme of the second part. The current state of practice and literature on embedded software process and product quality, and software process improvement approaches is presented in this second part. Based on an analysis of this current situation the problem definition is refined and specified in detail. Chapter 3 starts with an overview of the structural problems with software quality and their causes, followed by an overview of embedded software quality literature. This overview is subdivided into a process orientation, product orientation and measurement of these orientations. The reason to

focus on improving development processes to create product quality is clarified. This is the reason that the current state of approaches for software process improvement (SPI) is also presented, (Chapter 4). Based on the available approaches for SPI and the requirements for the embedded product domain, an analysis of strengths and weaknesses of the existing approaches is made. A selection from these weaknesses is made that will be solved within this thesis, and a set of criteria is defined to which an approach for 'product focused SPI' should comply.

In the third part of this thesis such an approach is developed. A conceptual model for product focused SPI is designed that complies to the criteria set in chapter 4. First of all, a set of expansions to existing SPI approaches is presented that match the criteria for product focused SPI, and these expansions are used to design a conceptual model that is elaborated in detail regarding available literature and practical experiences (Chapter 5). Furthermore, it is concluded that product focused SPI requires strong learning effects and therefore learning theory is explored. Based on findings in learning theory the conceptual model is expanded and learning enablers for product focused SPI are presented with detailed practical examples (Chapter 6).

The fourth part of this thesis contains the application of product focused SPI in practice. Therefore, a set of practical guidelines is provided for applying the conceptual model, and detailed experiences are presented from the case-studies in two international companies. In chapter 7, detailed guidelines are provided for each of the three working areas of the conceptual model. These guidelines and the conceptual model have been used in four industrial case-studies of which the detailed findings and experiences are described, together with an analysis of the benefits and costs (Chapter 8).

The final part of this thesis contains a chapter on conclusions and recommendations (Chapter 9). Conclusions are presented on product focused SPI in general, the conceptual model and the guidelines. Furthermore, some recommendations are provided for further research on the topic presented in this thesis, and an epilogue is provided that looks back on the research as a whole.

How to read this book largely depends on the personal interests of the reader; however, four possible paths are recommended (Figure 1-2). When interested in the current state of product and process quality for embedded product development it is recommended to read parts 1, 2 and 5. When interested in the concepts of product focused SPI and learning oriented process improvement, it is recommended to read parts 1, 3 and 5. Furthermore, if the reader is a practitioner that wants to *apply* product focused SPI it is recommended to read parts 1, 4 and 5, and to at least scan part 3. Finally, for those people closely interested in the whole research, it is recommended to read part 1 to 5 sequentially.

2. Research Methodology and Approach

A methodological overview of this thesis is provided in this chapter. A presentation is given of the methodological justification of this research, and the approach undertaken. Furthermore, the scope of this research is presented and the main assumptions on which it is based.

2.1 Methodology

In management science and related disciplines two main streams of research are distinguished [Verschuren and Doorewaard 1995] [Renkema 1996][1]: 'theoretical research' and 'applied research'. The first stream is mainly observing phenomena, and attempting to find generic theory. Solving theoretical problems are the main outcomes of this type of research. The second stream is interfering in practice and attempting to solve practical problems by designing theoretically sound solutions. Both streams have the objective to develop valid theories, however the first stream does this via 'describing', and the second via 'designing'.

2.1.1 The role of case-studies within applied research

This research is of the second type: applied research. An approach is designed in this research with the intention to solve some specific problems in practice. The approach designed in this thesis is used to take specific action in industrial practice and to observe whether this interference solves the problems. Some people might argue that such an approach does not result in scientific knowledge, because it only takes specific action to solve specific problems. In this research, however, a generic problem is addressed in a certain domain, and the research is designed as such that the proposed solution is valid within that domain. It therefore results in 'design knowledge'. The main methodological instrument for the development of such design knowledge is the 'reflective cycle' [Aken 1994].

[1] Different terms are often used for these two types of research. Theoretical research is also referred to as empirical-analytic, fundamental, descriptive, purely scientific, or explanatory research. Applied research is also referred to as design, action, or intervention research.

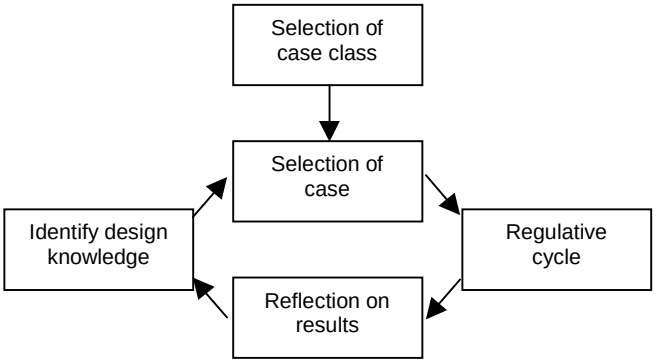


Figure 2-1: The reflective cycle of research [Aken 1994]

The reflective cycle is an add on to the regulative cycle [Strien 1986]. This regulative cycle is used to perform a study that consists of three characteristic phases [Renkema 1996]: a problem definition and diagnostic phase, a design and change phase, and an evaluative phase. During the regulative phase a study is carried out in a practical context, during which action is taken to solve a specific problem, and the effects of the action are observed and evaluated. During the reflective cycle, knowledge is identified that can be abstracted from the regulative cycle [Aken 1994].

Case-studies are a sound research technique during the regulative cycle. Case-studies are most useful to perform research to answer ‘how’ and ‘why’ questions, which do not require control over behavioural events, and which focus on contemporary events [Yin 1994]. In other words, when performing research in a practical context and clarifying ‘how’ and ‘why’ phenomena occur and interrelate, case-studies are preferred. Case-studies are mainly performed for two purposes:

- discovery: identifying phenomena in practice and observing causal relationships, events, etc. [Lammers 1997] [Hutjes and Buuren 1992]
- validation: checking whether certain theories are correct [Yin 1994] [Zwaan 1998]

Most literature on research methodology mainly suggests using case-studies for the first purpose; however, the value of the second purpose is often underestimated: case-studies can be used to validate the theoretical correctness of an approach [Yin 1994] [Zwaan 1998]. The validity of case-study results will be discussed in more detail, because in this research case-studies were applied as the main validation technique.

2.1.2 Validity of case-study results

Validity is the term used to test the 'quality' of research results. Two types of validity are relevant for this methodological discussion [Kidder and Judd 1986] [Yin 1994]:

- 'internal validity', establishing causal relationships, whereby certain conditions are shown to lead to other conditions, to distinguish them from false relationships
- 'external validity', establishing the domain to which the findings of a study can be generalised

Internal and external validity are related. Internal validity addresses observing causal relationships under certain conditions, while external validity addresses the domain within which these causal relationships are correct. In the previous section it was stated that case-studies are internally valid: a case-study is an excellent mechanism for the regulative cycle and for studying causal relationships within a specific context. External validity is, however, not so obvious because the validity of the knowledge is limited to the context in which such a causal relationship was observed. It is difficult to predict from the results in one context what the results will be in another. In other words, it is not clear to which extend case-study results are generic, and what needs to be done to make case-study results as generic as possible. This section discusses the generalisability of case-studies, by comparing them to experiments: a research technique that is considered to be highly externally valid.

External validity of causal relationships is ideally shown by using experiments [Verschuren and Doorewaard 1995] [Yin 1994]. In an experiment two equal contexts are created of which one is held constant and in the other an independent variable is changed. The impact of this change on the dependent variables is observed and used to accept or reject the stated hypotheses. The main reasons for selecting experiments as validation instrument is its sound comparison possibilities. Results in a stable context are compared with results in an other context in which variables are manipulated. The correctness of the causal theory is investigated by means of 'falsification': it is attempted to reject a theory [Popper 1968]. If the theory can not be rejected it can remain; therefore, validation of theories is based on creating a context in which validation of a theory is possible. Doing this with experiments requires a rigorous control over the studied contexts [Zwaan 1998]. To check the correctness of theory in a practical context is therefore often considered to be impossible, due to the unsuitability of practice for experimentation. A context for falsification can, however, also be constructed within case-studies, by setting them up in a 'quasi-experimental' manner.

When carrying out case-studies, 'quasi-experimental' conditions can often be created under which comparison is feasible [Cook and Campbell 1979] [Yin 1994] [Zwaan 1998]:

- nested case-studies make it possible that several sub-cases within one case-study are distinguished that can be compared
- longitudinal case-studies make it possible to compare different similar situations over time
- multiple case-studies make it possible to compare when the different cases show similarities on the most relevant conditions

In such quasi-experimental conditions in practice it is possible to validate causal relationships under externally valid conditions. One main prerequisite is that a set of selection criteria is defined. The conditions that are crucial for the validation study should be known and the case-study should comply to them [Aken 1994] [Yin 1994] [Zwaan 1998]. When following this approach, it is possible to create a 'context for falsification': a set of case-studies based on which it is possible to reject a certain theory. When such a context for falsification is established it becomes therefore possible to accept or reject a theory based on its practical application.

For the validation of theory with multiple case-studies, two types of case selection are distinguished that influence the boarders of external validity [Yin 1994]:

- Cases with literal replication. In such cases similar results are predicted. The same results occur within similar case-studies, thus increasing the likelihood that these results also occur in other (similar) cases.
- Cases with theoretical replication. In such cases contradicting results are allowed to occur, but under explicitly stated reasons and predicted results. This constructs generalisability for a wider scope of cases, thus expanding the domain in which the results are externally valid.

The last type, theoretical replication, is considered to be the most economical and fruitful approach for validation via case-studies [Zwaan 1998] [Yin 1994]. When selecting case-studies an attempt must be made to try to cover the problem domain as best as possible, and to make individual differences that can be expected in the case-study results explicit. It is therefore not the objective to select case-studies that have the most in common, but to select the case-studies that have the most differences within the problem domain. The differences in these case-studies can influence the different results, but as long as these different results have been predicted, the approach can still be validated.

2.1.3 Validation within the case-studies of this research

The above stated conditions for an externally valid case-study research design were applied in the research presented in this thesis. The selection criteria for case-studies within this research were:

1. The organisation produces embedded products.
2. The organisation has not outsourced the development of the embedded software.
3. Product quality is considered to be an important product objective.
4. The organisation is able to change its software development processes.
5. The software developers are able to spend a portion of their efforts on process improvement.

An attempt was made to select case-studies that were scattered over the embedded product domain, to make the results of this research broadly valid. The more diverse the case-studies are, the better the external validity of the results becomes.

Case-studies were carried out in two organisations: Tokheim and Dräger. Both companies produce complex embedded products but for two totally different markets: petroleum retail, and medical equipment. Within Dräger one development project has been supported, and within Tokheim three development projects. The case-study in Tokheim is a combination of a nested, longitudinal and multiple case-study design. Three different projects were guided within one organisation, making it a nested case-study. These projects were supported sequentially during a period of four years, making it a longitudinal case-study. Furthermore, these three projects covered all the three product lines of this company making it a multiple case-study design, with the intention to allow theoretical replication. Thus, the Tokheim cases provide an excellent mix to compare case-study results to each other. The Dräger case-study is used to increase external validity by applying the designed approached in a totally different environment within the embedded product domain.

In this research, the four mentioned case-studies in two companies were used to validate a conceptual model and a set of guidelines. These four case-studies were assumed to be a sufficient mix of different projects within the embedded product domain, and were used as a context for falsification. If the application of the approach presented in this thesis does not fail in these case-studies, or if certain inadequacies occurred, but they had been predicted based on characteristics of these case-studies, then it could be concluded that the designed approach was externally valid.

2.2 Research Approach

The approach in which this research was undertaken is presented in this section. Figure 2-2 shows this order.

Analysis	
1.	Inventarisation of existing literature, experiences in Tokheim and orienting cases
2.	Analysis of problems and current theory
3.	Identification of some structural problems in existing SPI approaches
Design	
4.	(Theoretical) construction of a conceptual product-focused SPI model to remedy these errors
5.	Inventarisation and analysis of learning theory: identification of key issues for product focused SPI in a development type environment
6.	Update of the conceptual product-focused SPI model to include the key issues identified in learning theory
7.	Developing practical guidelines for product-focused process improvement based on the conceptual model and experiences in Tokheim and Dräger
Validation	
8.	Validating the practical guidelines through usage of this method in industrial case-studies
9.	Analysing the benefits and the costs involved with applying the conceptual model in practice

Figure 2-2: Sequence of steps taken in the research

Firstly, time was spent to analyse existing literature on software process improvement and embedded product quality. Furthermore, some small exploratory cases were carried out in Tokheim and experiences were collected. The results of this inventarisation were analysed, and a set of problems with existing SPI (Software Process Improvement) approaches was identified for usage in the embedded product domain. Secondly, a solution was designed to these problems by constructing a conceptual model for product focused SPI. This conceptual model was enhanced based on findings in learning theory to increase the learning effects of the conceptual model. Furthermore, a set of guidelines was designed to support the practical application of the conceptual model in the embedded product industry. Finally, this conceptual model and the guidelines were validated in the case-studies. In addition an analysis was made of the benefits and the costs for product focused SPI when using the approach presented in this thesis.

The analysis part of this research is presented in chapter 3 and 4. The first design of the conceptual model illustrated with some practical cases is presented in chapter 5. This first design is enhanced in chapter 6 with learning theory, and operationalised with practical guidelines in chapter 7. The validation of the conceptual model and the guidelines are described in chapter 8 together with the cost/benefit analysis.

2.3 Scope

This research has a specific scope, which will be described in this section. Each boundary of the research is presented separately.

2.3.1 Focus on embedded products

The first limitation is that this research was limited to embedded product development. Although the concept of product quality improvement by improvement of development processes is also common outside this domain, the focus of the thesis was embedded products. This has three main reasons. Firstly, the embedded product domain is a domain in which the improvement of product quality via a focus on the process is specifically relevant. Secondly, the embedded product domain is a domain in which different quality demands exist for products, but the methods do not distinguish different processes for different product quality demands. Finally, the organisations that were available for case-studies were all in the embedded product domain.

2.3.2 Focus on improvement of embedded software development processes

This research was limited to the improvement of the development processes of the software for the embedded products. Other processes also influence embedded product quality. For example, the development process of the hardware, but also the manufacturing processes and the service processes influence the quality perception on the product; however, this research was only focused on the development processes of the software.

This is done for two reasons. Firstly, the quality of the embedded product is often largely determined by the quality of the software, as the software largely determines the functionality of the product, and the development efforts are often largely spent on the software developments [Rooijmans et al. 1994] [Karjalainen et al. 1996] [Taramaa et al. 1996]. Secondly, the improvement of embedded software development processes is receiving increasing attention [Trienekens 1994]; however, the impacts of these process improvements on the product are rarely considered. Therefore, the focus of the thesis is specifically the software processes for embedded products.

2.3.3 Focus on integration of measurement into SPI

Measurement should be carried out together with SPI approaches [Humphrey 1989] [Hetzel 1995] [Fenton and Pfleeger 1996]. This is, however, often not done in practice. Even though this will be further elaborated in chapter 3 and 4, it must be stated that one of the explicit starting points of this research was to look for possibilities to increase the integration of software measurement into SPI.

2.3.4 Operationalisation of product quality to project level

Addressing product quality and process improvement is often an activity that focuses at the organisational levels. Existing SPI approaches intend to increase the software processes of the whole organisation; however, the quality of product is largely determined in the project that develops it. Therefore, this thesis was focused on the operationalisation of product quality and process improvement approaches to the project level. It was the intention to make it possible to carry out project specific process improvement programmes that focus to the project specific product quality goals.

2.3.5 Not a 'quality culture' thesis

The focus of this thesis was on providing practical guidelines on *'what'* to do to improve product quality. As such, the focus of the thesis is not on topics such as establishing a quality culture, motivating people, management commitment, implementing Total Quality Management, etc. This thesis is intended to provide guidelines for practice to be used within a practical conceptual model for product focused SPI.

2.3.6 No solution to resistance to change in general

Similar to the previous topic, resistance to change is not addressed in general. Much research has been performed and literature is available on how to change organisations and how to make sure that the people in those organisations support such changes. This thesis does not provide material in that domain; however, this thesis does provide practical guidelines to carry out product focused SPI that have been tested in practice and that did not cause significant resistance. As such, this thesis is related to change management, but it is not the main topic.

2.3.7 Not a CMM thesis

Finally, this is not a thesis on the Capability Maturity Model (CMM) [Paulk et al. 1993]. CMM is a widespread method for software process improvement that is used in the embedded product domain. This thesis does not attempt to validate or expand the CMM as process improvement framework. This thesis uses the CMM as input to the research and sees the CMM as an assessment based SPI approach, as discussed in chapter 4. Experiences with the CMM and its strengths and weaknesses are important inputs to this thesis. This thesis, however, constructs its own conceptual model for product focused SPI in which the CMM has its place, but in which shortcomings are also overcome.

2.4 Assumptions

This thesis is written based on four main assumptions:

- There is no 'silver bullet' for embedded product quality. It is not possible to prescribe *the* right process towards product quality: every organisation should attempt to find its own situated process for each specific situation. The 'best' process for a product will always be a situated one, depending on a multitude of factors such as customer needs, market, developer experience, process maturity, etc. This assumption is a contingency-based approach [Galbraith 1977] as starting point for designing a solution to the problem addressed in this thesis.
- It is possible to control embedded product quality by taking specific actions in the development process. In this thesis it is assumed that creating product quality is a matter of selecting the right development process for that specific situation. It is assumed that product quality can be created by using a specific process. Identifying what this specific process should be for each situation, is addressed in this thesis.
- Product quality can only be guaranteed if it is specified beforehand. Without the explicit specification of the product quality demands it is assumed to be impossible to create a product that is perceived as quality by its users in a predictable manner. Specification of product quality is therefore a prerequisite to product focused SPI.
- Other factors beside process actions that may influence product quality, such as people factors, political factors, environmental factors, are assumed *not* to interfere with the approach under construction. Such factors have an important impact to product quality, however this thesis focuses to the impact of the process.

2.5 Conclusion

In this chapter an overview has been provided on the research methodology and research approach. The generalisability of case-study results was discussed, and the conclusion is that the selected mix of case-studies in this research has the potential to be externally valid. Having three multiple, longitudinal and nested cases in one organisation fully covering all three product types can result in internally and externally valid research results. A final external validity check in a different organisation for a different product in the embedded software domain, completes this research and completes the drive for external validity.

Furthermore, the steps taken in the analysis, design and validation phases of the research were presented in this chapter, together with the scope and underlying assumptions. Based on this methodological clarification the presentation of the research results can start, beginning in the next chapter with an overview of the existing literature of embedded software process and product quality.

3. Embedded Software Process and Product Quality

In this chapter the starting points of this thesis on process and product quality are introduced. Both the product orientation and the process orientation for embedded software quality are addressed, and furthermore the role of measurement for embedded software quality is clarified.

3.1 Introduction

Quality has received much attention during the last decades. Several approaches for quality have been published and used with success [Crosby 1979] [Deming 1986] [Juran 1988]. A closer look on the essential characteristics of embedded software is needed, because this thesis focuses on the quality of embedded software and its relationship with the development process. Before these characteristics are considered, a short overview is provided on the history of embedded software and its application in embedded products.

3.1.1 History of software and software engineering

Today's world has accepted software as a normal product, and it is used to the application of software into daily life; however, software differs in many aspects from other products. This can best be explained by stepping back in time and looking at the history of software engineering.

Although there were some attempts in the past to create a purely mechanic computer, the major breakthrough came from the application of electronic computation. Electronic circuits were constructed, containing only hardware that had one dedicated purpose. Later, electronic circuits were developed with some flexibility in functioning, by which certain tuning options of the hardware became possible. It enabled designers to define 'settings' for changing the function of the specific hardware. These tuning possibilities expanded largely, until a machine was built for which the computation could be tuned completely. As a result, the tuning of this machine became a job in itself: computer programming.

A computer program (the software) was stored on different kinds of media, such as punch cards, tape, or digital videodisc. In the early days, each computer had its own way of programming. This evolved into standard ways of programming computers and evolved into standard hardware components and interfaces: computers became open systems [Gal and Genuchten 1996].

Together with the standardisation of the hardware and of programming software, major application of software (and hardware) in society became possible. This evolved into the current state of practice, which has a large degree of computer and software application.

One should remember from this short historical overview, that software is more or less a means to tune hardware. As a result of the large tuning possibilities and flexibility of hardware, software engineering became a discipline in itself, and the results are considered as a separate product. Let's take a look at the specific characteristics of software, from this perspective.

3.1.2 Characteristics of software

Most problems with (embedded) software quality originate from some of its basic characteristics [Fenton and Pfleeger 1996] [Genuchten 1991] [Gillies 1992] [Basili and Rombach 1988] [Heemstra 1989]:

- Software is abstract. Software is not a physical entity: it is not possible to see, feel, taste, hear or smell software. Human senses fail when attempting to detect software. If a print of a software program is made on paper that can be seen and touched, merely a static representation of that software is seen and not the software itself. Only in operation software performs the dynamic functions it is developed for, and only then its attributes can be appreciated.
- Software is complex. Software code can contain over hundred types of statements, with over a thousand variables, in a sequence of over one million lines. Branches in the software make it possible that there are millions of paths to go through a program. The large number of program lines and exponentially growing number of program paths, make software complex.
- Software is flexible. Compared to other products that do have a physical existence, it seems to be easier to change software products. Although this is the largest strength of software, it is also a weakness. Changes to software can seriously hamper the architecture of the software, and can make the software totally different from its initial intention.
- Software properties are difficult to measure. Originating in the absence of a physical entity, measurement difficulties arise. Software has no weight, volume, or colour. Visual inspections, which are common for other products, become difficult when you can only see a representation of software. The measurement difficulty for software is extremely relevant because measurement is needed to identify quality.

3.1.3 Characteristics of software engineering

The above characteristics of software largely influence the way in which software is developed. Development of software is termed 'software engineering'. Specific aspects of

the software engineering process that are related to software process and product quality are [Basili and Rombach 1988] [Humphrey 1989] [Shaw 1990] [Genuchten 1991] [Bemelmans 1998] [Basili 1993] [Oivo 1994] [Basili et al. 1994a] [Glass 1995] [Fenton and Pfleeger 1996]:

- Software engineering is a creative, intelligent type of work, which highly depends on well skilled intelligent people. Assigning well-equipped and educated people to a software project is the most critical success factor reported from industry.
- Software engineering is a relative young discipline, which has existed for only a couple of decades. As a consequence there is limited knowledge and experience of 'how' to develop quality software. Only a few 'scientific laws' are available for software engineering, which indicate that it is a weak 'engineering discipline'.
- Due to the complexity and abstractness of software, software engineering uses processes that very often can not be clearly traced to the product. A product is developed by means of several phases, actions and intermediate products (documents and software); however, the individual contribution of these process parts to the final product is often unclear.

These characteristics of software products and software engineering are the basis for many process and product quality problems for embedded software development. Examples are planning difficulties, unknown or bad product quality, projects that fail, milestones that are reached months or years too late, etc.

Industry is often confronted with the above listed characteristics and problems. Developers are confronted repeatedly with the same questions such as:

- What does 'good quality' mean? For whom? When?
- How can it be ensured that the different requirements of different users are all incorporated in the product to ensure that all users find the product of good quality?
- How to create quality? How to control it during development? What to do to resolve quality problems?
- What can be done to ensure that product problems are detected before they are shipped to the users? Or, how can one ensure that product problems are found as soon as possible?
- How to evaluate whether embedded software is of good quality? How to measure quality?
- How can it be ensured that the continuous change of the software does not decrease quality?

3.1.4 Definition of quality

Answering these questions can not be done without a framework to be used to describe what 'quality' actually means. Quality means different things to different people. To define

quality might seem easy, but it is not. In an industrial context it is necessary to define quality goals and attributes, and to take appropriate process and measurement actions to create quality [Trienekens 1994]. Garvin classifies five different definitions of quality [Garvin 1984]:

- Transcendent based view. A context specific interpretation of quality is defined which is difficult to measure. It expresses that people are capable of recognising what they perceive as quality and what not.
- Product based view. It is assumed that product quality can be created by defining several objective measurable product attributes. Specification of software requirements in a measurable way, and controlling quality via these measurements, is expected to result in quality.
- User based view. In this definition the specific stated and implied needs of the user are addressed. This resembles Juran's definition of quality: 'fitness for use' [Juran 1988].
- Manufacturing based view. Through management of the process and eliminating flaws and defects, product quality is expected to be controlled. It assumes that when a product is made exactly according to specification ('conformance to requirements' [Crosby 1979]), the resulting product is of good quality.
- Value based view. In this definition quality is related to other factors such as time, effort and cost. This definition focuses on balancing quality to these other factors: a certain quality for a certain price, with a certain life-time, and developed within a certain time.

These definitions are all used in parallel by the different stakeholders for embedded products. What should be remembered from these definitions is that selecting just one definition of quality will not work. For example, a transcendent based definition does not support a development team of embedded products in creating quality products. On the other hand, fulfilling all manufacturing requirements for a product and sufficient scoring on the product quality metrics, does not guarantee that users will experience the product as good quality. In order to create quality embedded products, all five types of should be considered.

For embedded software development two orientations on quality can be recognised in practice. The first is a product orientation, which focuses on specifying and evaluating the attributes of product quality (product based view), but also addresses the specification of quality by the actual users (user based view). The second is a process orientation, which focuses on continuous improvement of the development processes under the assumption that this will improve the product as well (manufacturing based view). Both orientations when used in practice balance against costs and time issues (value based view). This product orientation and process orientation will be elaborated further in this chapter.

3.1.5 Co-ordination of input, process and output

The fact that both product and process orientations are being used to manage product quality can be clarified by theory on organisational control. Mintzberg defines five co-ordinating mechanisms that explain the fundamental ways in which organisations co-ordinate their work. These five mechanisms are [Mintzberg 1983]:

- mutual adjustment, which achieves the co-ordination of work by the process of (informal) communication
- direct supervision, which achieves co-ordination by having one person in charge of the work of others, issuing instructions and monitoring actions
- standardisation of work processes, preferable when the content of the work is well specified
- standardisation of outputs, preferable when the results of the work can be specified beforehand
- standardisation of skills *and knowledge* (inputs), which is done when the other standardisations are not possible

According to Mintzberg, these five co-ordinating mechanisms fall into an explicit order: '*As organisational work becomes more complicated, the favoured means of co-ordination seems to shift from mutual adjustment to direct supervision to standardisation, preferably of work processes, otherwise of outputs, or else of skills, finally reverting back to mutual adjustment*' [Mintzberg 1983].

Development of software can be controlled in the same way as organisational control. Co-ordination based on mutual adjustment in a chaotic environment will be changed towards an environment that standardises its work processes. One of the main actions by managers to assure a successful project is to hire good people, with certain skills and knowledge. Working towards standardisation of work processes and output products is, however, a logical next step, which clarifies the current focus in the software industry to improve product quality by orienting on processes and products.

3.2 Product orientation towards software product quality

The product orientation towards software product quality is based on refining quality into attributes and sub-attributes, using hierarchical product quality models [Gillies 1992]. Product quality standards exist, which refine quality in characteristics and/or sub-characteristics. Quality characteristics are used as attributes to describe a software product. Some quality models in which the relationships between the quality characteristics are determined are described in the literature. Examples can be found in [Cavano and McCall 1978], [Boehm 1981], [ISO 9126 1991] and [Quint 1991].

3.2.1 ISO 9126 standard for software product quality

For dividing product quality into attributes, the ISO 9126 quality model is used in this thesis [ISO 9126 1998]. This is an international standard that addresses user-needs of a product explicitly. In ISO 9126 six quality characteristics are distinguished: Functionality, Reliability, Usability, Efficiency, Maintainability and Portability. Each of these quality characteristics is further refined into sub-characteristics. For example the quality characteristic 'maintainability' is divided into five quality sub characteristics: analysability, changeability, stability, testability and compliance. A common vocabulary is provided by ISO 9126 to express quality of *user needs*. An overview of the ISO 9126 standard is presented in Figure 3-1.

Functionality - the capability of the software to provide functions which meet stated and implied needs when the software is used under specified conditions. • Suitability -the capability of the software to provide an appropriate set of functions for specified tasks and user objectives. • Accuracy - the capability of the software to provide right or agreed results or effects. • Interoperability - the capability of the software to interact with one or more specified systems. • Security - the capability of the software to prevent unintended access and resist deliberate attacks intended to gain unauthorised access to confidential information, or to make unauthorised modifications to information or to the program so as to provide the attacker with some advantage or so as to deny service to legitimate users. • Compliance - the capability of the software product to adhere to standards, conventions, or regulations in laws and similar prescriptions
Reliability - the capability of the software to maintain the level of performance of the system when used under specified conditions • Maturity - the capability of the software to avoid failure as a result of faults in the software. • Fault tolerance - the capability of the software to maintain a specified level of performance in cases of software faults or of infringement of its specified interface. • Recoverability - the capability of the software to re-establish its level of performance and recover the data directly affected in the case of a failure. • Compliance - the capability of the software product to adhere to standards, conventions, or regulations relating to reliability
Usability - the capability of the software to be understood, learned, used and liked by the user, when used under specified conditions. • Understandability - the capability of the software product to enable the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use. • Learnability - the capability of the software product to enable the user to learn its application. • Operability - the capability of the software product to enable the user to operate and control it. • Attractiveness - the capability of the software product to be liked by the user. • Compliance - the capability of the software product to adhere to standards, conventions, or regulations relating to usability

Efficiency - the capability of the software to provide the required performance, relative to the amount of resources used, under stated conditions. • Time behaviour - the capability of the software to provide appropriate response and processing times and throughput rates when performing its function, under stated conditions. • Resource utilisation - the capability of the software to use appropriate resources in an appropriate time when the software performs its function under stated conditions. • Compliance - the capability of the software product to adhere to standards, conventions, or regulations relating to efficiency
Maintainability - the capability of the software to be modified. • Analysability - the capability of the software product to be diagnosed for deficiencies or causes of failures in the software, or for the parts to be modified to be identified. • Changeability - the capability of the software product to enable a specified modification to be implemented. • Stability - the capability of the software to minimise unexpected effects from modifications of the software. • Testability - the capability of the software product to enable modified software to be validated. • Compliance - the capability of the software product to adhere to standards, conventions, or regulations relating to maintainability
Portability - the capability of software to be transferred from one environment to another. • Adaptability -the capability of the software to be modified for different specified environments without applying actions or means other than those provided for this purpose for the software considered. • Installability - the capability of the software to be installed in a specified environment. • Co-existence - the capability of the software to co-exist with other independent software in a common environment sharing common resources. • Replaceability - the capability of the software to be used in place of other specified software in the environment of that software. • Compliance - the capability of the software product to adhere to standards, conventions, or regulations relating to portability.

Figure 3-1: ISO 9126 standard for quality characteristics and sub-characteristics [ISO 9126 1998]

3.2.2 Product quality evaluation

The ISO 9126 standard can be used for product quality evaluation. Evaluations of a software product must be objective - based upon observation, not opinion. They should be reproducible. Evaluation of the same product to the same evaluation specification by different evaluators should produce results that can be accepted as identical and repeatable. To do so, procedures for project control and judgement are necessary; an evaluation process is required. Such an evaluation process was defined during the 'SCOPE' project [Robert 1994]. This project was the basis for the international standard ISO 14598, which is visualised in Figure 3-2. The standard also distinguishes three perspectives on evaluation: developer, acquirer and evaluator.

	Analysis	Specification	Design	Execution
Process for Developers 14598-3	definition of quality requirements and analysis of their feasibility	quantification of quality requirements	planning of evaluation during development	monitoring of quality and control during development
Process for Acquirers 14598-4	establishing purpose and scope of evaluation	defining the external metrics and corresponding measurements to be performed	planning, scheduling and documentation of evaluation	evaluation shall be performed, documented and analysed
Process for Evaluators 14598-5	describing the objectives of the evaluation	defining the scope of the evaluation and the measurements	documenting the procedures to be used by the evaluator	obtaining results from performing actions to measure and verify the software

Figure 3-2: ISO 14598 definition of evaluation process [ISO 14598 1996]

Products with different application risks must also be evaluated differently. For example a mobile phone has a lower application risk than the security system of a nuclear power plant. Four levels are distinguished by decreasing risk: A to D. Figure 3-3 presents a proposal of evaluation techniques for these four levels and quality characteristics [Rae et al. 1995].

	Level A	Level B	Level C	Level D
Functionality	formal proof	component testing	review (checklists)	functional testing
Reliability	formal proof	reliability growth model	fault tolerance analysis	programming language facilities
Usability	user mental model	laboratory testing	conformity to interface standards	user interface inspection
Efficiency	performance profiling analysis	algorithmic complexity	benchmark testing	execution time measurement
Maintainability	traceability evaluation	analysis of development process	static analysis	inspection of documents (checklists)
Portability	program design evaluation	environment constraints evaluation	conformity to programming rules	analysis of installation

Figure 3-3: Evaluation techniques for various levels and quality characteristics [Rae et al. 1995]

The thoroughness of an evaluation is reflected in the evaluation techniques used. Different evaluation levels result in different levels of confidence in the quality of a software product. Different embedded products with different application risks should also be evaluated differently.

3.2.3 Embedded product architecture and product quality

One specific area of product evaluation is the evaluation of the architecture to determine the quality of a product. This is based on the notion that [Clements et al. 1995] [Clements and Northtrop 1996] [Bass and Kazman 1999]:

- An architecture permits or precludes the achievement of a system's targeted quality attributes. Making a 'good' architecture is the first order approach to achieving product quality attributes.
- It is possible to predict certain qualities about a system by studying its architecture.
- The architecture is the main object for communication and understanding the system and its development.
- The architecture is the embodiment of the earliest design decisions.

Two types of architectures are relevant in the context of embedded systems. Firstly, there is the product architecture, which contains both the hardware and the software. For the construction of the product architecture decisions are taken on including certain features or qualities or not, and decisions on whether certain features or qualities are provided by the hardware or by the software. Secondly, there is the software architecture, which is the structure of the components of a program or system, their interrelationships, principles and guidelines governing their design and evolution over time [Garlan and Perry 1995]. For the software architecture decisions are taken on the general structure and subdivision of the software. Architectural decisions have a large influence on product quality since they provide the structure of a product or software that enables or disables the possibility to address certain product quality attributes.

'*It is important to understand, however, that an architecture alone cannot guarantee….the quality of a system. Decisions at all stages of the life cycle – from high level design to coding and implementation – affect system quality. Therefore, quality is not completely a function of an architectural design. A good architecture is necessary, but not sufficient, to ensure quality*' [Clements and Northtrop 1996].

In this thesis, architectural design of both the product and the software will not be addressed separately. Designing an architecture in which the product quality is sufficiently addressed is assumed to be a default step in the software development process. The importance of a good architecture is, however, underlined. During the collection of product quality requirements and the development of the product, architectural issues deserve special attention.

3.2.4 Problems with product oriented approaches towards software quality

The starting point of product oriented approaches is the specification of quality of the product. An objectively measurable specification of product quality is a prerequisite for the product oriented approaches; however, this specification of product quality is not always available, nor is it easy to specify.

It is not clear what this product exactly is. Is it the software code? With or without documentation or other means of support? Is it the software programme in operation? Or is evaluation of the product architecture sufficient? These questions are not easy to answer. From the user point of view, it is at least necessary to look at the software in operation, because in that case the software performs the function it is intended for.

One other issue for product oriented software quality creation is the notion that product quality can not be brought in at the end. Once a product is finished its quality is determined. Bad quality can not be easily changed into good quality by carrying out some adjustments. Quality must be created along the path of product development. Intermediate checks and improvements are necessary. This refers to the necessity of a process-oriented approach towards software quality.

3.3 Process orientation towards software product quality

In contrast to the product orientation to software quality, there is the process orientation. This is based on the assumption that a 'quality process' results in a 'quality product'. Improvements to the product should therefore be created by the process. ‘*The process is the key to understanding and solving product problems. Product problems must be related back to the process - how the software was developed, it is necessary to learn what factors are important in the process*‘ [Yeh 1993]. It must be noted, however, that a single focus on the development process can not guarantee the quality of the end product.

The process orientation towards product quality improvement originates from the manufacturing industry, where the production process is managed through application of statistical process control. Quality methods such as the Shewart-Deming cycle Plan/Do/Check/Act [Deming 1986] are based on this process orientation. It is however, questionable to what extend such a process focus is applicable to software processes, since these are different from production processes. One other argument to support this doubt is that there are almost no norms to compare software processes to. Since software engineering is such a young discipline, it is unknown what the best way is to build certain software, in a certain context, for a certain domain.

By improving and controlling the quality of the software development process, the quality of the product is expected to become more constant, with higher, better predictability and to be better controllable. '*Just as manufacturers look for ways to assure the quality of the products they produce, software engineers should find methods to assure that their products are of acceptable quality*' [Pfleeger 1991]. The most well known standards for evaluating the quality of the software development process are the ISO 9000-3 quality standard [ISO 9000-3 1997], the Capability Maturity Model (CMM) [Paulk et al. 1993] and the SPICE standard (ISO 15504) [ISO 15504 1998].

Within the process orientation techniques are used such as process modelling and process assessments. Assessments are used to evaluate the status of an organisation's processes to a normative model. *An assessment is a review of a software organisation to provide a clear and factual understanding of the organisation's state of software practice* [Humphrey 1989]. The result of an assessment is an analysis of the current processes in the organisation and the extent to which they comply to industry 'best practice'. The current capabilities of the organisation are listed in this analysis, including a proposal for further improvements of the processes. Based on the assessment, a process improvement plan is developed that specifies which processes have to be improved and the actions to be taken to implement these improvements.

3.3.1 Problems with process oriented approaches for software quality

Even though industry has reported many successes with the process orientation, the question is still open as to what the impact is of each specific part of the process on product quality. Firstly, the experiences from practice claim product quality improvement by increasing the 'maturity' of the organisation [Goldenson and Herbsleb 1995]; however, each capability consists of several specific improvement actions, for which it is not clear what the single contribution to product quality is. Secondly, the benefits from literature mainly focus on product defects. Although product defects are part of quality, they are only one single aspect. Other aspects of product quality improvement are rarely investigated.

The focus on the process should be complemented with a focus on the product. '*One of the best ways to evaluate a software organisation is to examine the quality of its products. Product quality is a key measure of the software process. It provides a clear record of development progress, a basis for setting objectives, and a framework for current action'* [Humphrey 1989].

Summarising, improving the processes to create product quality is the focus of the process orientation. This is done because quality is created during this process and not after it is finished. This is a logical starting point to focus on the process; however, the impacts of process improvements on product quality should be clear in that case to control product

quality. It is clearly not a matter of choosing between a product or process orientation; both types need to be applied together.

3.4 Measurement of process and product quality

'Measurement' is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules [Fenton and Pfleeger 1996]. Measurement can be used on both a software development process and a software product. It can be used to measure quality characteristics of a certain software product, or to measure effects of a certain software process. *Software measurement* is the continuous process of defining, collecting, and analysing data on the software development process and its products in order to *understand, control* and *optimise* that process and its products.

Five types of measurement scales are commonly distinguished: nominal, ordinal, interval, ratio and absolute [Fenton and Pfleeger 1996]. These scale types (Figure 3-4) have increasing levels of richness, which means that the scales become more sophisticated and allow stronger statistical analysis methods.

Measurement has an important role in 'real' engineering disciplines [Shaw 1990]. '*Professor Mary Shaw of Carnegie Mellon University points out that mature engineering fields codify proved solutions in handbooks so that even novices can consistently handle routine designs, freeing more talented practitioners for advanced projects. No such handbook yet exists for software, so mistakes are repeated on project after project, year after year*' [Gibbs 1994]. An empirical factor is added to the research on software engineering by using measurement, New methods, techniques and tools are measured to understand how good they are, which problems they solve, which they do not, and the basic needs to apply them. '*There has always been this assumption that if I give you a method, it is right just because I told you so. People are developing all kinds of things, and it's really quite frightening how bad some of them are*': Victor Basili in [Gibbs 1994]. Basili supports that measurement provides an excellent mechanism to learn what works and what does not. He promotes the existence of software engineering laboratories that investigate specific aspects through empirical evaluation, after which upscaling to industrial application can be started; unfortunately, few such software engineering research labs exists. The only well-known laboratory is the NASA-SEL [McGarry et al. 1994], which was started and is directed by Basili himself.

Scale	Explanation
Nominal scale	Nominal measurement consists of assigning items to groups or categories. No quantitative information is conveyed and no ordering of the items is implied. Variables measured on a nominal scale are often referred to as categorical or qualitative variables. Some examples: • classifying origin of defects into classes like design error, not tested fully, external reasons. • classifying failure detection phases into classes like field, integration, and testing.
Ordinal scale	The ordinal scale is similar to the nominal scale, but has the additional characteristic that the categories can also be ranked in that one class is better, or higher than another class. The ranges between two classes have no meaning for the ordinal scale. Some examples: • classifying failures into severity classes like fatal, major and minor. • classifying complexity into classes like high, medium and low.
Interval scale	The interval scale is an ordinal scale with equal differences between the classes. Interval scales do not have a true zero point, and therefore it is not possible to make statements about how many times higher one score is than another. A common example is the Celsius scale for indicating temperature, where a 10 degree difference has the same meaning anywhere along the scale; however, because the zero point is arbitrary, it is not possible to say that a temperature of 30 degrees is twice as warm as a temperature of 15 degrees.
Ratio scale	Ratio variables are very similar to interval variables; in addition to all the properties of interval variables, they feature an identifiable absolute zero point, allowing statements such as x is two times more than y. A much used example to describe the distinction between interval and ratio scales is temperature. In contrast to the Celsius scale it is allowed to say on the Kelvin temperature scale that a temperature of 200 degrees is twice as high as one of 100 degrees.
Absolute scale	The measurement for an absolute scale is simply made by counting the number of elements that needs to be measured. The absolute scale resembles the ratio scale strongly. The difference is that for the absolute scale the scale is unique (i.e. there is only one scale allowed for the measurement). The absolute scale is the most restrictive scale.

Figure 3-4: Five types of measurement scales [Fenton and Pfleeger 1996]

A valuable tool is provided by measurement to understand the effects of specific process actions that are implemented to improve a software development process. Examples of results are [Möller and Paulisch 1993][Pfleeger 1991]:

- increased understanding of software development processes
- increased control of the software development process
- increased capacity to improve the software development process
- more accurate estimates of software project costs and schedule
- more objective evaluations of changes in technique, tool, or methods
- more accurate estimates of the effects of changes on project cost and schedule
- decreased development costs due to increased productivity and efficiency
- decrease of project cycle time due to increased productivity and efficiency
- improved customer satisfaction and confidence due to higher product quality

Software measurement data is interpreted by people to provide information that can be used for three different purposes: understanding, control and improvement [Basili and

Rombach 1988]. In the first place, the data makes the current development process visible and the characteristics of the software products. This visibility is required to reduce complexity and increase *understanding* of the process and products. Once basic understanding has been established, the collected and analysed data can be used to *control* the process and the products, by defining corrective and preventive actions. Furthermore, based on analysis, the collected measurement data can be used to assess the process, and therefore act as an indicator of development process problem areas, from which process *optimisation* can be established. This hierarchy of measurement is illustrated in Figure 3-5.

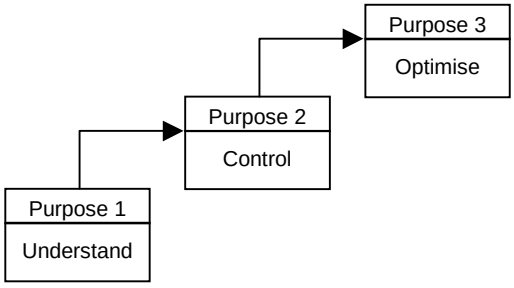


Figure 3-5: Applying measurement for 3 purposes

Two main problem areas exist for the measurement of software development: the product area and the process area. Firstly, a software product is abstract, complex and difficult to measure. Since software has no physical existence, measurement problems arise. Secondly software engineering is a young discipline with a lack of scientific laws, in which measurement is not a common routine. On the contrary: for measurement of software special attention is required, and when used in the wrong way measurement may causes resistance on behalf of the developers. Summarising, both the product and the process orientation for software measurement have special demands and problems. These will be further elaborated in the next sections on product and process measurement.

3.4.1 Measurement of software products

Product measurements can be divided over external and internal product measurements [Humphrey 1989] [Fenton and Pfleeger 1996]. External product measurement measures the external (dynamic) behaviour of the product. Examples are function response times, number of screens, time to add a feature, mean time to failure, etc. Internal product measurement measures the internal (static) structure of the software product. Examples are measurements of lines of code, cyclomatic complexity, test defects found, number of comment lines, etc. Both types of measurement are used in practice. External measurement is more interesting from a user point of view, while internal measurement is more interesting from a developer point of view.

The relationship between internal product measurements and their external product characteristics, however, is largely unknown. A clear example is described in the paper of [Gentleman 1994]. This paper compares internal and external software product measurement with the measurement of audio speakers. For these speakers hundreds of internal technical metrics existed; however, these metrics only become of interest, once their values could be correlated to what people experienced as being a 'good sound'. The relation between internal and external product quality metrics is relevant for software engineering, but has not been established yet.

Available methods for external product measurement, can be used, for example, to evaluate conformance of a product to user needs. These approaches assume that user-needs are known, and made explicit. Examples of such methods are: Scope [ISO 14598 1996], Space-Ufo [Trienekens et al. 1997] and the ISO 9126 part III [ISO 9126 1998]. The methods are based on the specification of product quality and evaluating the end-product to this specification. The more objective and concrete these specifications are, the better it can be evaluated. Specification of product quality in measurable terms is therefore recommended [Gilb 1994].

In the available methods for internal product measurement, it is mainly the static attributes of a product that are measured. These measurements can be used to compare internal product measurements to (company specific) norms and standards to achieve a certain level of quality. Software measurement literature contains many examples of internal software metrics (see for example: [Shepperd 1993] [Shepperd and Ince 1993] [Cook and Roesch 1994] [Zuse 1991] [Albrecht and Gaffney 1983] and [Halstead 1977]). These metrics are also easy to measure due to the availability of automated calculation tools for these metrics.

The main benefit of both internal and external product measurement is that objective numbers are provided to characterise a software product. The same stands for process measurements. Even though the limitations of metrics are still questionable, they are in some way *objective*. Remembering that this level of objectivity is crucial for co-ordination using standardising processes or products, their benefits for control of product quality are clear. Current practice has insufficiently adopted this kind of measurement working to achieve quality products, which is a big shortcoming in the current state of practice.

3.4.2 Measurement of software processes

A partial solution to the unknown relationship between process and product quality is provided by software process measurement, as it supports in the discovery of the impact of process actions on product quality in a specific context. Through measurement and analysis of the development process and its success in achieving the intended product qualities, an evaluation instrument becomes available. As explained before, data collection

and analysis of software engineering processes is rarely executed in current practice. Many authors have insisted on expanded application of measurement during software development, however industry still has not adopted this sufficiently [Hatton 1995].

By monitoring the performance of the software development process, it is possible to provide an overview on actual results of that process, and to take corrective action based on these results. Measurement is an excellent mechanism for *control* of a software process. Several methods are available that describe how to carry out process measurement. The Goal/Question/Metric approach [Basili and Weiss 1984] was used for this research and will therefore be explained in detail.

3.4.3 Goal/Question/Metric measurement

In the GQM method a systematic approach is represented for tailoring and integrating goals to models of the software processes, products and quality perspectives of interest, based upon the specific needs of the project and the organisation [Basili et al. 1994b]. The result of the application of the GQM method is the specification of a measurement system targeting a particular set of issues and a set of rules for the interpretation of the measurement data.

By using GQM a certain goal is defined, this goal is refined into questions, and metrics are defined that should provide the information to answer these questions. By answering the questions, the measured data can be analysed to identify if the goals are attained. Thus, by using GQM, metrics are defined from a top-down perspective and analysed and interpreted bottom-up, as shown in Figure 3-6.

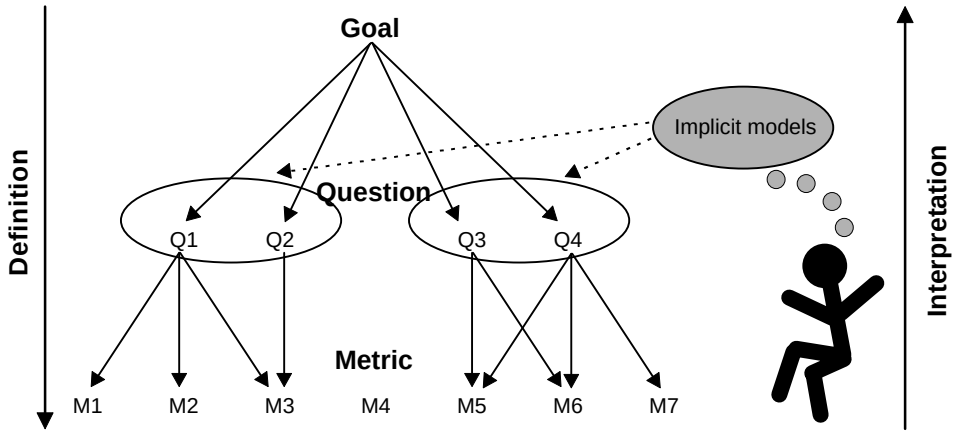


Figure 3-6: The GQM Paradigm [Basili and Weiss 1984]

The GQM model is started top-down with the definition of an explicit measurement goal. This goal is refined into several questions that break down the issue into its major

components. Each questions is then refined into metrics that should provide information to answer those questions. Measurement data is interpreted bottom-up. As the metrics are defined with an explicit goal in mind, the information provided by the metrics should be interpreted and analysed with respect to this goal to conclude whether or not it is attained.

GQM trees of goals, questions and metrics are built on knowledge of the experts in the organisation: the developers [Basili and Rombach 1988]. Knowledge acquisition techniques are used to capture the implicit models of the developers built during years of experience. Those implicit models give valuable input to the measurement programme and will often be more important than the available explicit process models.

Even though the GQM approach is mostly considered to be a process measurement approach, it can also be used to measure product quality [Solingen and Berghout 1999].

3.5 Conclusions

It has been stated in this chapter that the creation of embedded software quality is difficult; the characteristics of software and the current state of software engineering mainly cause this. Control of embedded software product quality is established by two orientations: a product orientation and a process orientation on software quality.

Managing embedded software quality can be done by standardisation of work products and standardisation of work processes. Standardisation of product quality can be done through the use of standard product quality terminology, and specifying product quality in measurable terms. Such standard terminology supports in dividing quality into attributes and sub-attributes, making the abstract concept of quality better manageable. Furthermore, the definition of the product depends on its use. User views on product quality are reflected in this product quality specification. If possible, product quality is specified in measurable terms.

Standardisation of work processes can be done through modelling and assessment of established software development processes. These approaches support in carrying out a standard process and documenting this process. The main reason for a focus on the process is the notion that product quality is created by that process, and that quality can not be created by making some adjustment to the product when it is already available.

Both the software development process and the software product can be measured. Increased control and early feedback on results are the outcome of measurement; therefore it is recommended to use measurement, both of product quality and of process performance. The Goal/Question/Metric approach provides a goal-oriented way of measurement that can be used in the context of this thesis.

For the development of embedded products, there is a focus on product quality, because the product is being sold, not the process; however, once a product is finished its quality is already determined and it is difficult to improve it. This clarifies why often emphasis is put on improving the embedded software development process, because this impacts product quality. The assumption is that better processes lead to better products. To support this approach, the embedded software industry uses software process improvement (SPI) methods to implement continuous improvement of their processes. SPI is the topic of the next chapter.

4. Software Process Improvement

Software Process Improvement (SPI) is described in this chapter. An overview of SPI methodologies is provided, and the strengths and weaknesses of these methodologies are discussed. Furthermore, a set of criteria is identified in this chapter to which a methodology for product focused SPI for embedded product development should comply.

4.1 Introduction

Software Process Improvement (SPI) is the set of activities with which an organisation attempts to reach better performances on product cost, time-to-market and product quality, by improving the software development process. Changes are made to the process based on ‘best practices’: experiences of other, not necessarily similar organisations. Within SPI methodologies there is a focus on the software development process, because it is based on the assumption that an improved development process positively impacts product quality, productivity, product cost and time-to-market. Note that the changes to the development process do not guarantee that these intended impacts will indeed occur.

The identification of the criteria to which a SPI methodology for embedded software development should comply is the objective of this chapter. These criteria are identified based on the strengths and weaknesses of existing SPI methodologies.

4.2 Software Process Improvement Methodologies

The four SPI methodologies that are most referred to in literature are: CMM, ISO 9000-3, SPICE and BOOTSTRAP. Those methodologies will be described in more detail.

4.2.1 Capability Maturity Model (CMM)

Based on best practices from industry, the Software Engineering Institute (SEI) developed the Capability Maturity Model (CMM). The model was originally developed to assess the software development process of third party suppliers, of the US Department of Defence; however, this model also helped those suppliers to improve their process. Organisations are supported with the CMM to improve the maturity of their software process through an evolutionary path, of five maturity levels (Figure 4-1), from 'ad hoc and chaotic' to 'mature and disciplined' management. As organisations become more ‘mature’, risks are expected to decrease and productivity and quality are expected to increase.

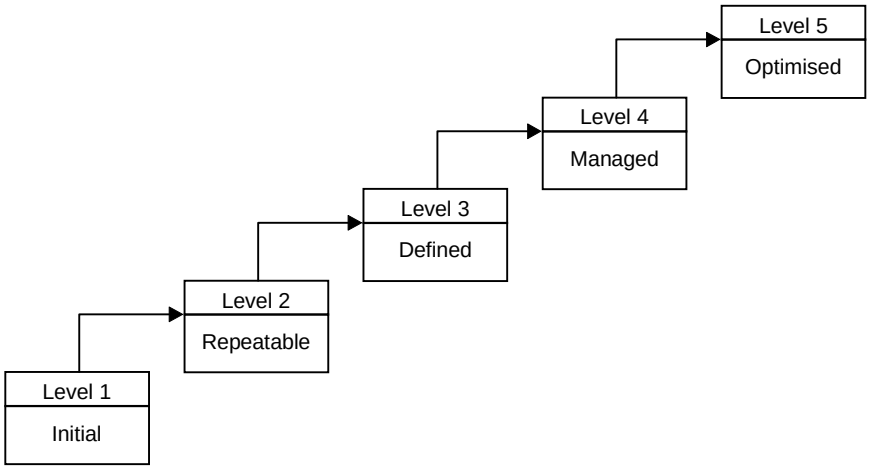


Figure 4-1: Capability Maturity Model (CMM) [Humphrey 1989] [Paulk et al. 1993]

Each CMM maturity level contains a set of Key Process Areas (KPA's) which describe the main capabilities for that level. The KPA's of the CMM are listed in Figure 4-2.

Level	**Key Process Areas (KPA)**
1. Initial The software process is characterised as ad hoc, occasionally or chaotic. Few processes are defined, and success depends on individual effort and heroics.	
2. Repeatable Basis 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.	• Requirements management • Software project planning • Software project tracking & oversight • Software subcontract management • Software quality assurance • Software configuration management
3. Defined The software process for both management and engineering activities is documented, standardised, and integrated into a standard software process for the organisation. All projects use an approved, tailored version of the organisation's standard software process for developing and maintaining software.	• Organisation process focus • Organisation process definition • Training program • Integrated software management • Software product engineering • Inter-group co-ordination • Peer reviews
4. Managed Detailed measurements of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled.	• Quantitative process management • Software quality management
5. Optimising Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.	• Defect prevention • Technology change management • Process change management

Figure 4-2: The five levels and KPA's of the CMM [Paulk et al. 1993]

4.2.2 ISO 9000

An other well-known methodology for SPI is the ISO 9000 approach. The assumption behind the ISO 9000 standards is that a well-managed organisation with a defined engineering process is more likely to produce products that consistently meet the purchaser's requirements, within schedule and budget, than a poorly managed organisation that lacks an engineering process. ISO 9001 describes 'Quality systems - model for quality assurance in design/development, production, installation and servicing'. A representation of the structure of the ISO 9000 standards is given in Figure 4-3.

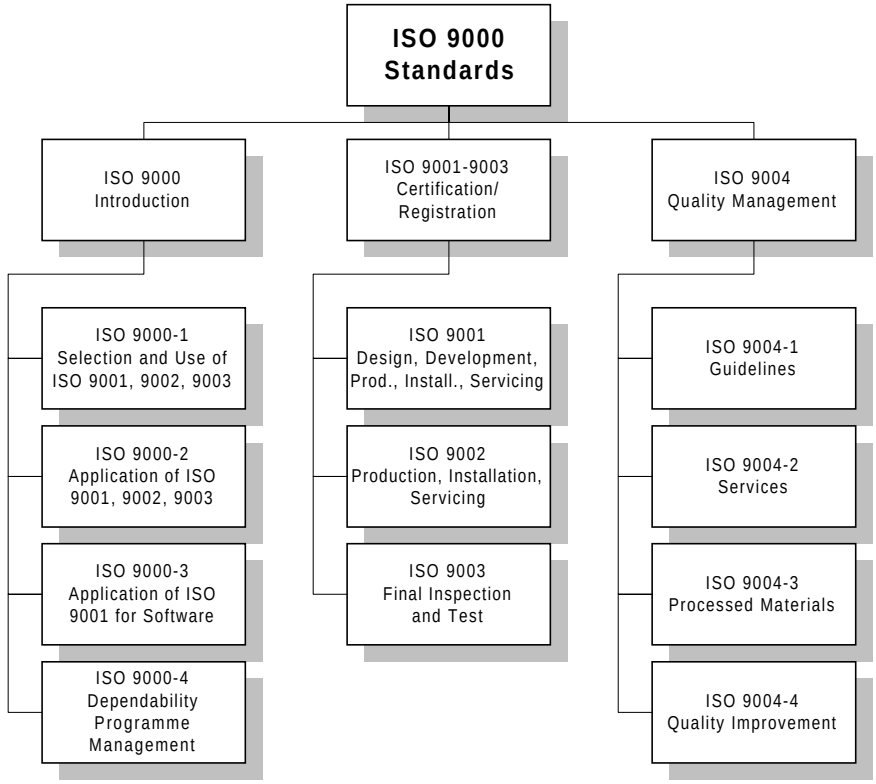


Figure 4-3: The structure of the ISO 9000 standards

ISO 9001 is a set of quality system requirements that consists of twenty clauses that represent requirements for quality assurance in design, development, production, installation and servicing, that defines which aspects of a quality system have to be available within an organisation. Details on how these aspects should be implemented and institutionalised are not provided by ISO 9001. As ISO 9001 was written to be used in all kinds of industries, ISO 9000-3 was added specifically for software-development: 'Guidelines for the application of ISO 9001 to the development, supply and maintenance of software'. ISO 9000-3 provides guidelines for applying ISO 9001 to the specification,

development, supply and maintenance of software [ISO 9000-3 1997]. The requirements for ISO 9000-3 certification are divided over twenty requirements classes (Figure 4-4).

Nr.	Guideline classes
4.1	Management responsibilities
4.2	Quality system requirements
4.3	Contract review requirements
4.4	Product design requirements
4.5	Document and data control
4.6	Purchasing requirements
4.7	Customer-supplied products
4.8	Product identification and tracing
4.9	Process control requirements
4.10	Product inspection and testing
4.11	Control of inspection equipment
4.12	Inspection and test status of products
4.13	Control of non-conforming products
4.14	Corrective and preventive action
4.15	Handling, storage and delivery
4.16	Control of quality records
4.17	Internal quality audit requirements
4.18	Training requirements
4.19	Servicing requirements
4.20	Statistical techniques

Figure 4-4: ISO 9000-3 Guideline classes [ISO 9000-3 1997]

Receiving ISO 9000-3 certification assures customers that in an audited company all its processes and work instructions documented conform the ISO requirements, and that these processes and work instructions are being followed on a continuous basis. ISO certification does not give any guarantee for product quality, it only indicates that the procedures are used to a certain extent.

4.2.3 BOOTSTRAP

The BOOTSTRAP method [Kuvaja and Bicego 1994] [Bicego et al. 1998] is the result of an European project under the auspices of the European Strategic Programme for Research in Information Technology (ESPRIT). It provides an alternative for organisations that are interested in improving their software development process and attaining ISO 9001 certification, as it combines and enhances the methods provided by the CMM and the ISO 9000 quality standards.

The basis of the BOOTSTRAP methodology is established by CMM. Like the CMM, an assessment is based on five maturity levels, but the BOOTSTRAP method uses a different scale to measure an organisations' or projects' overall strengths and weaknesses. The ISO 9000 quality standards (ISO 9001 and ISO 9000-3) are incorporated in the methodology because they provide guidelines for a company-wide quality system. The CMM does not

include such guidelines. Furthermore, many European companies use ISO 9000 as a primary quality standard. BOOTSTRAP can be used by organisations to determine readiness for ISO 9001 certification.

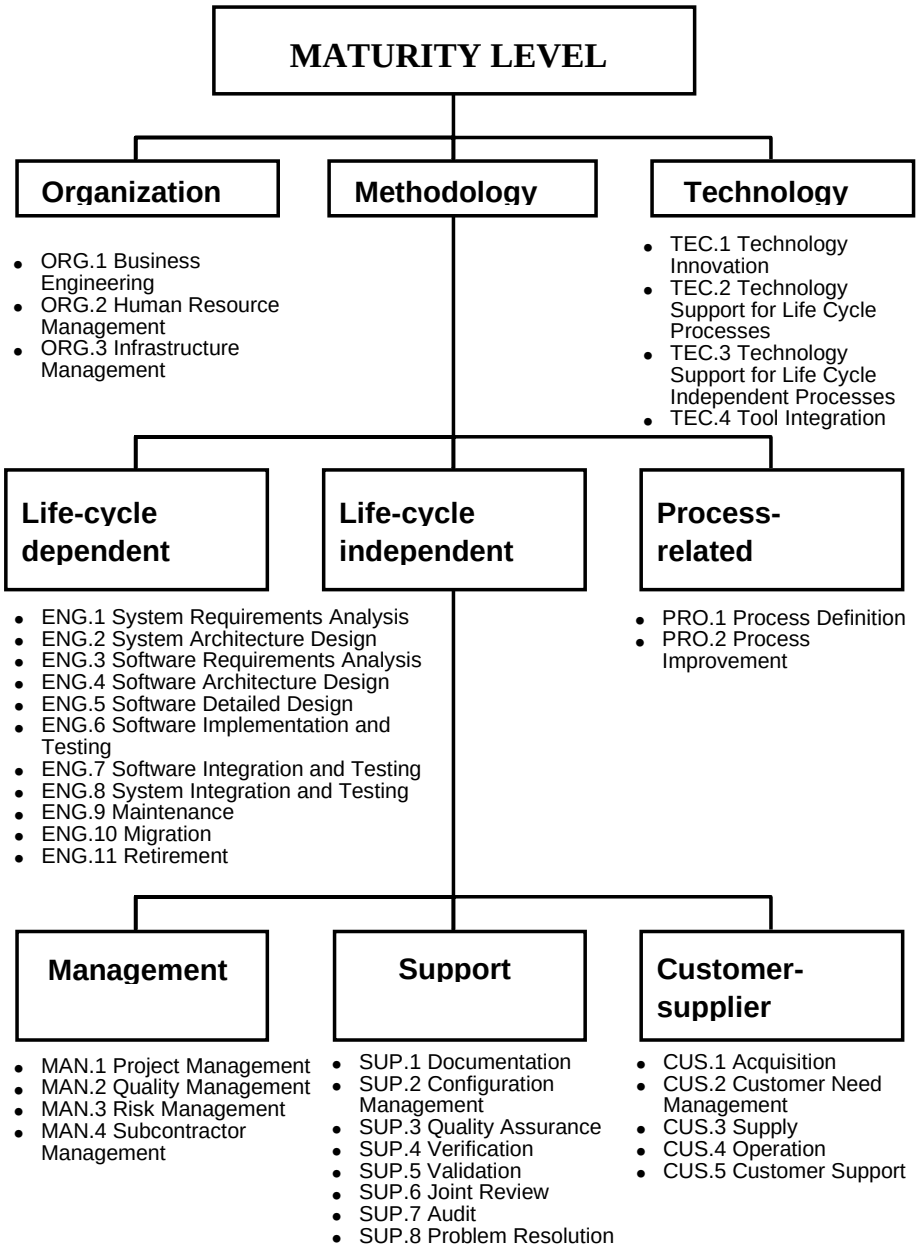


Figure 4-5: BOOTSTRAP methodology [Kuvaja and Bicego 1994]

BOOTSTRAP distinguishes three areas that identify the maturity of an organisation: technology, methodology and organisation. Methodology is sub-divided into a life-cycle

dependent, life-cycle independent and process related area, of which the life-cycle independent area is further divided into management, support and customer-supplier. For each area in this BOOTSTRAP tree, a number of 'processes' are defined. Each process has a number of 'key-practices' that need to be addressed for that process. Furthermore, each process has a 'capability dimension', which identifies the current status of that process on a scale from 0 to 5. Unlike the CMM, quartiles between these levels are distinguished, which make it possible to assess on organisation on for example level 2.5, expressing that level 2 is established and that 50% of the level 3 capabilities are in place.

4.2.4 SPICE

SPICE (Software Process Improvement & Capability dEtermination) is a major international initiative to develop a Standard for Software Process Assessment [ISO 15504 1998]. ISO 15504 (Figure 4-6) is used as a reference framework for software process capability determination. It is based on other popular approaches, mainly on BOOTSTRAP, CMM and ISO 9001.

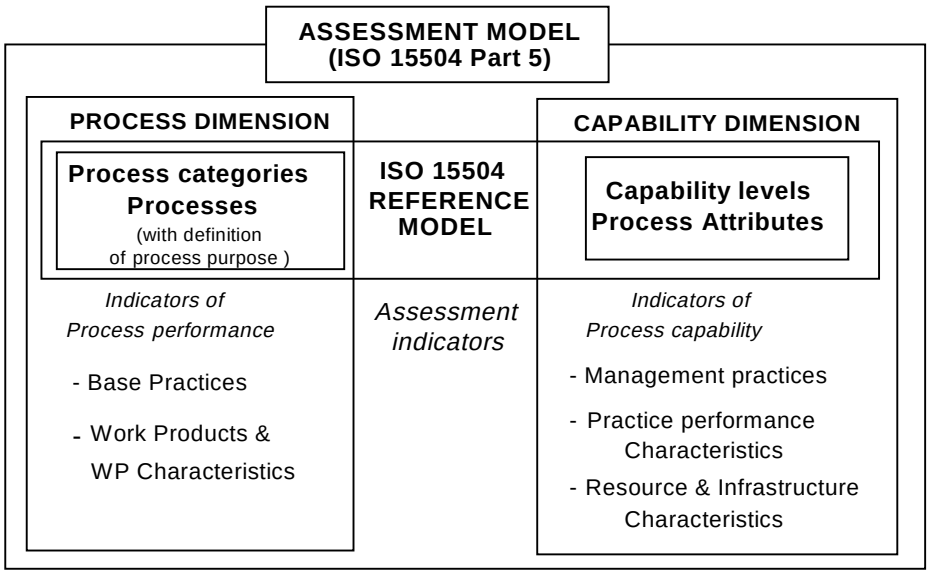


Figure 4-6. The ISO 15504 framework

Changes with respect to the CMM are [Sassenburg et al. 1996]:

- a broader scope: processes that are indirectly related to the software development processes are also considered
- a different architecture: levels are distinguished in all Key Process Areas, whereas specific Key Process Areas within the CMM are only of importance within a certain level
- an integration of other SPI-models, such as ISO 9000, TickIT and Trillium

4.3 Experiences with using SPI in practice

In the foregoing section four well-known methodologies for SPI were presented. These methodologies are being used in practice, and results are being reported.

Several software companies have reported large benefits from adopting SPI methodologies [Daskalantonakis 1994] [Dion 1993] [Rooijmans et al. 1996] [Humphrey et al. 1991] [Wohlwend and Rosenbaum 1994] [Kuvaja and Bicego 1994]. The benefits reported are positive cultural change, increased productivity, shorter time-to-market, better communication, increased product quality, and improved customer satisfaction. It should be noted that many companies only report successes and not the failures, and that results are sometimes biased on how benefits have been measured.

Some embedded product companies report large benefits. Hughes Aircraft claims to save $2 million per year using SPI [Humphrey et al. 1991]. Furthermore, the work conditions and the work ethic are improved. Schlumberger claims that the amount of projects delivered in time has increased from 51% to 94%, and that engineer productivity has doubled [Wohlwend and Rosenbaum 1994]. This increase was realised by using SPI over a period of three years. Raytheon notes a doubling productivity and a return of $7.7 on every dollar invested into SPI [Dion 1993]. The amount of rework as a percentage of the total costs of a project decreased from 40% to 10% in four years. The absolute costs associated with fixing defects remained approximately constant. The prevention costs decreased to the same extend as the total project costs. As a result the total project costs decreased about 30%. Goldenson and Herbsleb have shown that SPI application has a cost-benefit ratio that varies from 1:4 to 1:9 [Goldenson and Herbsleb 1995].

The Software Engineering Institute (SEI) has made an analysis of Software Process Assessments results [Hayes and Zubrow 1995] from 48 organisations undertaking 2 or more CMM assessments. The analysis focuses on the time required to increase process maturity, as well as the most prevalent process issues faced by the 48 organisations. On average, organisations moving from CMM level 1 to level 2 did so in approximately 30 months, between the first assessment that found them at level 1 and the subsequent assessment that found them to be at CMM level 2. Organisations moving from CMM level 2 to level 3, on the other hand, did so within an average of 25 months. The most important finding of this particular analysis is that there is much more variability in the amount of time organisations take to move from level 1 to level 2 than there is in the time it takes to move from level 2 to level 3.

These positive results are also criticised, regarding the validity of these findings, especially the performance measurements are criticised, and it is noted that the answers the managers gave to questions of the SEI inquiries were much more positive than the answers of the software engineers [Hetzel 1995].

4.4 Strengths and weaknesses of SPI methodologies

The available methods for SPI have been presented together with an overview of results of applying these methods in practice. As the development of a conceptual model for SPI, specifically aimed at creating embedded product quality was the objective of this thesis, it was necessary to analyse the currently available methods on their fitness for the embedded product domain. This analysis of the strengths and weaknesses of the existing SPI approaches is presented in this section.

Strengths	Weaknesses
S-1.Based on best practices	W-1.Product quality not addressed
S-2.Provides a vision	W-2.Lack of measurement
S-3.Management tool for improvement	W-3.No cost/benefit analysis included
S-4.Changes are prescribed	W-4.Too generic
S-5.Explicit priority to quality	W-5.No project level support: mainly for large organisations
	W-6.Continuation difficult
	W-7.Dependency on individual managers
	W-8.Phasing not logical
	W-9.Improvement takes long
	W-10.Risk for bureaucracy

Figure 4-7: Strengths and weaknesses of existing SPI methodologies

The analysis of strengths and weaknesses of SPI methodologies will be done on a general level. The main reason for this is that it is not the goal to compare SPI methodologies or to decide which one is 'best'. In this thesis the aim was to create a conceptual model for embedded product focused SPI. To create such a conceptual model it should be clear what the strengths and weaknesses of the existing methodologies are for the embedded product domain. Based on such an overview, it becomes possible to point out the main criteria for methodologies for product focused SPI that suit the embedded product domain.

4.4.1 S-1: Based on best practices

Current SPI methodologies show large improvements in organisational performance [Goldenson and Herbsleb 1995]. The recent increase in professionalism in software engineering in practice has been facilitated through the use of SPI methodologies. Although the scientific grounds for these methods have been criticised [Bach 1994], practice has shown them to be useful. Many positive experiences and results with SPI methodologies are reported (section 4.3).

4.4.2 S-2: Provides a vision

A vision for practice is provided by SPI methodologies on 'how' professional software engineering should be done [Humphrey 1989]. Changes to software development processes are proposed in a specific sequence. This prescribed sequence makes SPI easy to apply, because it provides organisations with guidelines on the activities to carry out. The main goal of existing SPI methodologies is that they are implicitly aimed at establishing *the* right process. SPI methodologies have a strong assumption that the ideal process is known and only needs to be established in an organisation; however, software engineering is still an immature discipline for which valid knowledge is lacking [Shaw 1990] [Basili 1993]. This indicates that the available SPI methodologies include a paradox: they provide a vision, but this vision is based on missing knowledge.

4.4.3 S-3: Management tool for improvement

Stimulating improvement is one of the tasks of management [Humphrey 1989]; however, the tools for management to steer are limited. A clear path to follow is prescribed by SPI methodologies such as the CMM, and a tool for management is therefore included. Companies set objectives such as 'level 3 within 3 years', or 'ISO 9000-3 obliged for all our suppliers'.

4.4.4 S-4: Changes are prescribed

High-level quality models such as Plan-Do-Check-Act [Deming 1986] mainly describe an organisational context for improvement, instead of describing the things to do. Explicit criteria to comply to are set by SPI methodologies [Paulk et al. 1993] [Kuvaja and Bicego 1994]. These criteria are based on best practices, and therefore clearly indicate what an organisation has to do, to comply to a specific maturity level.

4.4.5 S-5: Explicit priority to quality

Using SPI methodologies gives explicit attention to process quality [Humphrey 1989]. Practice often supports the importance of quality, but rarely takes explicit action to create quality. Process quality is addressed explicitly in SPI methodologies and the priorities for process quality are being increased.

4.4.6 W-1: Product quality not addressed

The process is the main focus of current SPI methodologies. Product quality is not explicitly addressed [Trienekens 1994] [Bach 1994]. The explicit specification of product quality is, however, a pre-requisite for product focused process improvement, as explained in chapter 3. This lack of addressing product quality explicitly is a major weakness of

existing SPI methodologies. Especially in the embedded product domain, where the product is sold, and not the process. Existing SPI methodologies are based on the assumption that a quality process will lead to a quality product [Humphrey 1989]; however, the individual contributions of parts of the process, such as specific phases, methods, techniques or tools, to product quality are not considered. Such explicit relationships between process actions and product quality attributes should be explicitly included.

4.4.7 W-2: Lack of measurement

Measurement is an excellent tool to guide process improvement; however, existing SPI methodologies insufficiently apply measurement as a tool to guide improvements [Bach 1994] [Hetzel 1995]. Software engineering should fully apply measurement to obtain knowledge on its merits [Basili et al. 1986] [Basili 1993]. Measurement should be used in product focused SPI programmes to evaluate product impacts of process changes, monitor process change progress, and to learn current statuses and effects. As stated before, SPI methodologies operate under the assumption that a specific change has certain impacts. Such changes are often called ‘improvements’ already before the improvements have been realised. Evaluating whether a change really was an improvement should, however, be also measured, which is not included in the available SPI methodologies. This lack of measurement in existing SPI methodologies is considered to be a major weakness.

4.4.8 W-3: No cost/benefit analysis included

Benefits and costs of improvement are not made explicit. Without an integrated cost/benefit analysis in SPI methodologies, the use of these methods in practice is likely to disappear. Without evidence that the benefits of SPI are higher than the cost, organisations are likely to abandon SPI usage. Analysis of costs and benefits should, therefore, be an integrated part of any SPI methodology. Such cost/benefit analyses are not included in existing SPI methodologies, which is not a surprise because there is already a lack of measurement in the first place. Even though, analysis of costs and benefits is promoted in existing SPI methodologies, the collection of data on costs and benefits is not explicitly included.

4.4.9 W-4: Too generic

One normative way to improve software development for any organisation is prescribed by existing SPI methodologies. Existing SPI methodologies are based on ‘best practices’: success stories from industry with the application of certain capabilities. No distinction is made over different types of organisations, markets, products, cultures, previous experiences, etc. The flexibility and knowledge to tune to the application context is not

incorporated in current SPI methodologies. This is necessary, because it is unlikely that the best development process of, for example, mobile phones, hart monitors, cashing systems, and televisions is completely identical. It is probably even so that the best development process is also different for different generations of such products.

4.4.10 W-5: No project level support: mainly for large organisations

Successes with existing SPI methodologies are mainly reported by large organisations. Project level guidance to improve development processes is not the main focus of the existing methods: they focus on the whole organisation. For product focused SPI, however, support should be given to project teams because they develop the product, independent of whether they are part of a small or large organisation. This absence of project level support and main focus on large organisations is considered to be a weakness of existing SPI methodologies.

4.4.11 W-6: Continuation difficult

Continuation of SPI is difficult and experiences show that it also disappears. Although, SPI is still widely used in industry, some early adopters have already abandoned it, the reasons for this are unknown. Success stories are published in the literature, but failures rarely. One of the reasons that SPI projects die in the long run might be that it is just a separate project from the many other projects that are ongoing. SPI programmes are not linked to development projects, but mainly focus on the whole organisation in which software is being developed. The improvement programme centres around the yearly cycle of assessments from which many follow-up actions are defined. After the assessors have left the company, most actions remain pending until the next assessment is coming up. This separate nature of SPI programmes frequently causes them to be temporary and not continuous. Another issue that might influence continuation is the dependency on management commitment.

4.4.12 W-7: Dependency on individual managers

As SPI programmes are separate projects within organisations, the priority given to these programmes depends highly on the support of the individual managers [Humphrey et al. 1991] [Goldenson and Herbsleb 1995]. Since improvement of the processes and product is not part of every single development project, but more a parallel task, it is dependent on the support of individual managers. As management of organisations change, sometimes even very often, there is a big chance that SPI usage also disappears.

4.4.13 W-8: Phasing not logical

There appears to be a non-logical construction in SPI methodologies such as the CMM. This weakness is that improvement of practices is not done immediately after introduction. 'Optimisation' is postponed to the highest levels. This distracts organisations from optimising their current ways of working, and has the danger that growth on the maturity scale becomes the main objective [Bach 1994]. New and current process actions should indeed be understood and implemented correctly, however improvement and optimisation should not be postponed for several years.

4.4.14 W-9: Improvement takes long

The success of SPI activities needs to be made visible. Improvements should therefore be realised in short cycles. Existing SPI methodologies require years before results become visible [Hayes and Zubrow 1995] [Dion 1993]. Attaining goals of a project or organisation is an excellent means to show success. Progress is expressed in existing SPI methodologies, by noting an increase in maturity level [Bach 1994] [Hayes and Zubrow 1995]. Feedback on improvement should however be related to the projects and business performance, and not to an abstract structure of ‘levels’.

4.4.15 W-10: Risk for bureaucracy

Another weakness is that due to the total emphasis on process control, a bureaucracy can be installed. Creating procedures and manuals, and checking on their usage can become an objective, instead of a means. This risk of bureaucracy is remarkable: the path from a flexible but chaotic way of working (level 1) to a structured flexible way of working (level 5), is established through a situation that has the risk to become overly bureaucratic.

4.5 Towards product focused SPI

Based on the above listed set of problems, a set of criteria was defined to which a conceptual model should comply, to address the specific needs of embedded product focused SPI.

Criterion	Description
C-1	Product focused SPI specifies product quality explicitly
C-2	Product focused SPI relates process improvements explicitly to product quality
C-3	Product focused SPI uses knowledge and experiences from ‘best practices’
C-4	Product focused SPI supports individual projects with process improvement
C-5	Product focused SPI measures both the process and the product
C-6	Product focused SPI measures costs and benefits of the SPI activities

Figure 4-8: Criteria for product focused SPI for embedded software

4.5.1 C-1: Product focused SPI specifies product quality explicitly

It essential to specify product quality for product focused SPI. Without such a specification it will be impossible to focus on process improvement the product quality attributes of interest. It supports in the explicit addressing of product quality (S-3, S-5, W-1), supports in making project specific improvements (S-2, W-4, W-5), and shortens the time in which improvements become visible (W-9).

4.5.2 C-2: Product focused SPI relates process improvements explicitly to product quality

Relationships between process actions and product quality should be made explicit. This supports in selecting the right process improvements for the right product quality attributes (S-5, W-1, W-5, W-8). It supports in making a specific improvement programme for specific projects (S-2, S-4, W-4, W-5, W-10).

4.5.3 C-3: Product focused SPI uses knowledge and experiences from 'best practices'

Best practices from other projects within the same organisation or best practices from other organisations should be used to identify suitable improvements. Process actions that result in a certain level of product quality in one project can also be used in other projects (S-1, S-4, W-5).

4.5.4 C-4: Product focused SPI supports individual projects with process improvement

Creating product quality is often a project specific task; therefore, a product focused SPI methodology should be capable of supporting individual projects with SPI (W-4, W-5, W-7, W-10). This enables project management to control product quality explicitly (S-3, S-5, W-1). Furthermore, it stimulates continuation of SPI, at least within the project, and makes results visible within a reasonable time (W-6, W-9).

4.5.5 C-5: Product focused SPI measures both the process and the product

Application of measurement is a prerequisite for product focused SPI, because it provides feedback on effects of process actions on product quality (W-2). Measurement enables objective control of both the software process and product quality (S-3), and can be used in individual projects (W-5, W-9).

4.5.6 C-6: Product focused SPI measures costs and benefits of the SPI activities

Changes to a process can be regarded as 'improvements' once they result in measurable benefits. These benefits and the costs to achieve these benefits should be an explicit task in a SPI methodology, because it supports management with feedback on effects of investments in SPI (S-3, W-6). Carrying out a cost/benefit analysis using measurement will provide an objective overview on the value of SPI (W-2, W-3, W-6).

4.6 Conclusions

Software process improvement is a movement in the software industry, which has many positive effects. Experiences in industry show beneficial results in product quality increase, reduction of costs and shortening of cycle times. In this chapter, the strengths and weaknesses have been identified of applying these SPI methodologies for the improvement of embedded software processes. The most important points of critique to existing SPI methodologies are that product quality is not explicitly specified, measurement is not sufficiently integrated with process improvement, the specific needs of an organisation or project are not sufficiently adopted, and the cost/benefits of SPI are not measured.

In this chapter a set of criteria was specified to which methodologies for product focused SPI should comply. These criteria will be used in the next chapter to design a conceptual model for product focused SPI.

5. Conceptual model for product focused SPI

The conceptual model for product focused software process improvement (SPI) is constructed in this chapter. Firstly, four required expansions to existing SPI approaches are presented that comply to the criteria for product focused SPI defined in chapter 4. Secondly, three working areas are identified that need to be addressed in the conceptual model under construction. An elaboration is provided that for each of these working areas:

- works towards a textual definition for each working area
- provides some sample practical experiences for each working area
- presents existing literature relevant to each working area

Finally, a conceptual model is presented for product focused SPI, and based on this model the chapter concludes that learning theories need to be explored.

5.1 Necessary expansions for product focused SPI

It was concluded in the previous chapter that a method for product focused SPI should comply to a set of six criteria. Based on these criteria and the available approaches, a conceptual model for product focused SPI will be constructed. The demands set in the previous chapter indicate four major expansions of current SPI approaches:

- specifying product quality
- modelling of product-process relationships
- configuring a project specific process model depending on the product quality specification
- measuring product and process quality

The rationale that clarifies how these four expansions comply to the chapter 4 criteria is included in section 5.1.5. These four major expansions of current approaches will be discussed in more detail. A conceptual model for product focused SPI approaches was designed based on these major enhancements of existing SPI approaches,.

5.1.1 Specifying product quality

In order to create product quality, it must be addressed explicitly. Current SPI approaches address product quality often only implicitly, which will be solved by the approach for product focused SPI. The notion was adopted that quality is not something that can be brought in after a product has been created. It must, therefore, be clear from the start of a

development project, what the product quality objectives are. Defining product quality objectives can be done from several viewpoints. For example, from the viewpoint of an end-user, service engineer, marketing manager, or software engineer. In chapter 3 it was concluded that all such viewpoints need to be considered; however, special attention was given in this research to specifying the customer wishes for quality of an embedded product. Specification of user-based product quality is a prerequisite in an approach for product focused SPI. User needs for a product should be captured and made explicit in a product quality specification.

From now on, in this thesis the term '*Requirements engineering*' will be used for the specification of product quality. Requirements engineering will be further elaborated in section 5.3. It will be made clear that requirements engineering is more than just a specification of quality: it covers a negotiation process in which a trade-off is made between several issues such as product quality, product functionality, cost, duration, and the possible development process.

5.1.2 Configuring project specific processes depending on the product quality specification

In chapter 4 it was concluded that each development project needs its own specific process in order to create an effective and efficient product. This product-specific process contains a set of process actions that contribute to product quality. In order to let the process be most effective and efficient, this set of process actions should be selected based on the product quality requirements.

Once a set of relevant process actions has been selected a project specific process model can be constructed. Such a model represents the phases, activities and deliverables of a specific development project.

From now on, in this thesis the term *'Process engineering'*, will be used to represent the configuring of a situated development process that guarantees the required product quality. In order to facilitate the selection of process actions, an overview should be available on the impact of individual process actions on product quality. For this purpose process-product relationship models are used, which are part of process engineering.

5.1.3 Modelling of product-process relationships

This thesis is based on the assumption that product quality can be managed by taking specific actions in the software development process. In order to identify which process actions are required in a specific situation, the effects of a process action on product quality has to be made explicit. The individual effects of these specific process actions on product quality attributes should therefore be modelled. Such models reflect the expected

impact of a process action on attributes of product quality, such as reliability, maintainability or usability. These effects can be positive or negative. Examples of process actions that have a positive impact on reliability are the use of static code analysers, test techniques, or inspections. Process actions can have several impacts. Take for example the use of a graphical user interface, which on one hand is intended to have a positive effect on usability, but which on the other hand needs more resources and therefore has a negative impact on efficiency.

5.1.4 Measuring product and process quality

In chapters 3 and 4 it was concluded that 'measurement' is a prerequisite for successful process improvement. Measurement should be used for two purposes:

- evaluating conformance of a product to the product quality specification
- evaluating process-product relationships

The previous chapters also concluded however that this usage of measurement together with process improvement is rarely practised. The conceptual model under construction should therefore include measurement explicitly and apply it for these two purposes.

From now on, this thesis will use the term '*Measurement programme engineering*' for the evaluation of process-product relationships, and evaluation of end product quality.

5.1.5 Conformance of these four expansions to the product focused SPI criteria

In this section it is clarified why these four expansions solve the problems with existing SPI methods listed in chapter 4. This is done by presenting the relationship between the criteria for product focused SPI of chapter 4 and the proposed expansions.

In chapter 4 criteria were defined to which an approach for product focused SPI needs to comply. These criteria are depicted in Figure 5-1, together with an overview of the relationships between these criteria and the proposed expansions.

These criteria have been designed to overcome the fundamental problems of existing SPI approaches for improving embedded product quality. The criteria are addressed using the four proposed expansions. It is therefore possible to construct a conceptual model for product focused SPI based on these four expansions. Referring to the underlying problems and criteria is then no longer required. The relationship of each criterion to the four proposed expansions is discussed briefly below.

Criterion 1: Specification of product quality needs is included. This criterion is covered because the first expansion is the specification of product quality (section 5.1.1) as an explicit step for the development of a product. This is called requirements engineering.

Criterion 2: The relationship between process and product quality is addressed explicitly. The second proposed expansion is the modelling of product-process relationships (section 5.1.3). These models make these relationships explicit, and they can be used explicitly to control product quality. These models are used by both requirements engineering and process engineering during the negotiations on what quality will be built, and which process will be used. Criteria 2 is related to both working areas.

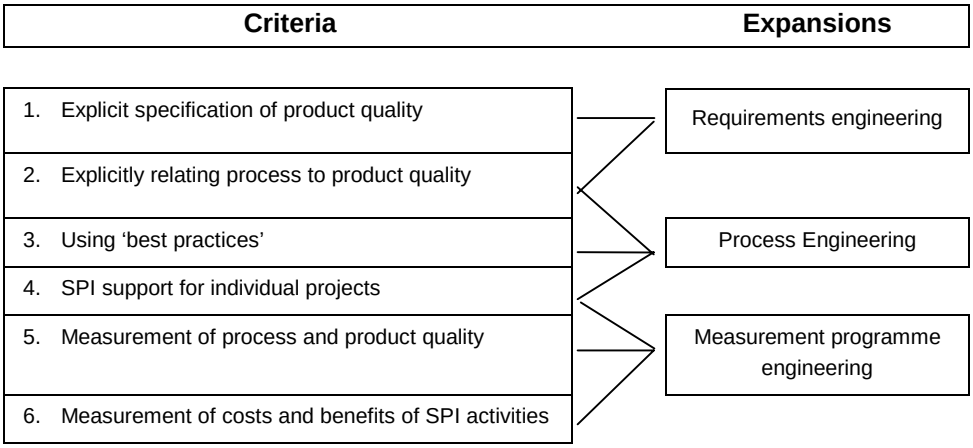


Figure 5-1: Relation between proposed expansions and criteria for product focused SPI

Criterion 3: Knowledge from 'best practices' is included. This criterion is covered with the second expansion: the modelling of product-process relationships (section 5.1.3). Best practices are captured by modelling these relationships and packaging experiences with applying these models. These process-product relationship models will be continuously updated with experiences on these relationships in practice. These models are part of process engineering.

Criterion 4: SPI for projects is possible. This criterion is covered because the third expansion is the configuration of project specific processes depending on a product quality specification (section 5.1.2). Due to the configuration of such a project specific development process, software process improvements can be specifically implemented for a project. Process engineering designs this project specific process. Furthermore, a measurement programme is put in place to support each individual project in measuring its performance and providing specific information to improve (section 5.1.3).

Criterion 5: Measurement of process and product quality is included. This criterion is covered because the fourth expansion is the measurement of product and process quality (section 5.1.3). This is called measurement programme engineering.

Criterion 6: Cost and benefits of the SPI activities are measured. This criterion can be covered by the fourth expansion: measuring product and process quality (section 5.1.3). Benefits in product or process quality improvement are measured by this expansion, but

costs are not explicitly stated; however, when measuring process and product quality the main costs are effort spent, which is often included in the measurement of processes. Costs and benefits are measured during measurement programme engineering

5.2 Towards a conceptual model

Based on existing SPI approaches and the expansions proposed in section 5.1, a conceptual model for product focused SPI was developed. In this model three working areas for product focused SPI must be included: requirements engineering, process engineering and measurement programme engineering.

Each working area addresses a range of activities that needs to be executed. Several methods, techniques or tools can be used to support each working area. The three working areas of product focused SPI, depicted in Figure 5-2, also seem to be interrelated. These interrelationships are currently not defined and will be tackled in section 5.6 of this chapter. In order to work towards a full model, in which these interrelationships are also defined, a first elaboration of the three working areas is necessary.

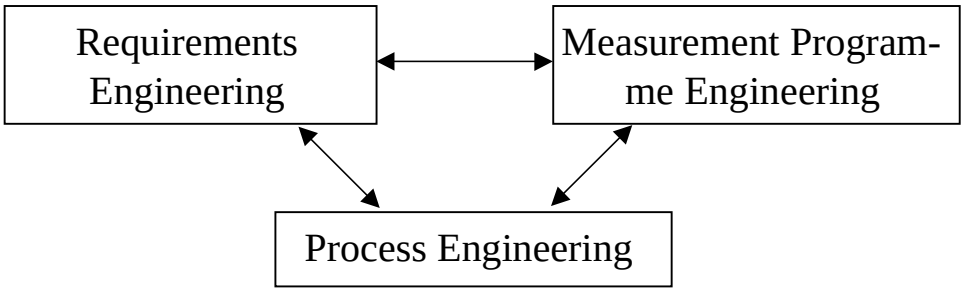


Figure 5-2: Requirements, process and measurement programme engineering

In the following sections an elaboration is included of the three working areas. Each section is started with a definition of the working area from literature. Based on literature and practical experiences, a renewed definition is created for each working area. Based on this ‘thesis-specific’ definition, the existing software engineering literature is explored for relevant issues regarding these three working areas. The findings of this literature exploration are listed and will be used in chapter 7 of this thesis to construct practical guidelines for product focused SPI.

5.3 Requirements Engineering

The traditional definition in literature defines requirements engineering as the process in which complete, consistent, unambiguous specifications have to be developed, which can serve as a basis for common agreement among all parties concerned and that describe what a product will do [Boehm 1981].

A distinction can be made between ‘what’ a product should do, and ‘how’ it should be implemented. Although it is sometimes difficult to make this distinction, traditional literature states that requirements engineering has to focus on the ‘what’ aspect [Boehm 1981] [Faulk 1996]. This makes a distinction between the ‘logical’ and the ‘technical’ specification, and states that requirements engineering should focus on the logical demands for a product. The technical specification and architectural design of how things are implemented is of later concern.

The logical requirements can be subdivided into functional requirements and quality, non-functional, requirements. Functional requirements specify the operation of a system, and quality requirements specify the non-functional aspects of a system such as performance, maintainability, or reliability. Such quality requirements receive, however, little attention while being equally important as functional requirements [Yeh and Ng 1990]. These two types of requirements can be distinguished, but their specification is one activity: functional and quality requirements are closely related. A certain function can be developed with different levels of reliability, security, maintainability, etc. Specification of functional and quality requirements should therefore not be seen as separated tasks. Focusing on functional requirements only and not considering the quality requirements at the same time is a mistake often made in practice [Bemelmans 1998].

Furthermore, requirements engineering results in a ‘common agreement among all parties concerned’ [Boehm 1981]. This underlines the negotiation character of requirements engineering. Specifying requirements deals with a trade-off between functional and quality requirements on one hand and development effort, costs and duration on the other hand [Bemelmans 1998]. Supporting this negotiation process and setting priorities to certain product quality attributes is presented in [Heemstra et al. 1995].

The end result of requirements engineering is a product requirements specification document [Leite 1987]. Four requirements engineering *subtasks* are distinguished [Thayer and Dorfman 1997]:

- elicitation: the process through which customers and developers discover, review, articulate and understand users’ needs
- analysis: the process of analysing users’ needs to arrive at a definition of requirements
- specification: the development of a document that clearly and precisely records each of the requirements of a product
- verification: the process of ensuring that the requirements specification complies to the product needs

So, requirements engineering is not just the development of a product requirements specification, it is the whole range of activities in which requirements are elicited, analysed, specified and verified.

5.3.1 Relevance of requirements engineering

The reason that requirements engineering is one of the working areas for product focused SPI, is because requirements engineering has a large impact on product quality [Thayer and Dorfman 1997]. The accuracy with which a development process is able to create a quality product is dependent on the accuracy of the product quality requirements [Davis et al. 1988]. Neglecting requirements may result in an insufficient product, no matter how elegant the design and its implementation are [Stevens and Paltu 1994]. Alford and Lawson state that the major cause of insufficient product quality, is in non-stated requirements [Alford and Lawson 1979]. These references support that requirements engineering for product quality requirements deserves to be a working area for product focused SPI.

Along side the fact that performing good requirements engineering prevents project failure, is the fact that it is also cost effective. Boehm states that the later mistakes are found the more expensive they are: mistakes found in the requirements phase cost only 1-2% per defect compared to defects found in the maintenance phase [Boehm 1981].

The importance of requirements engineering is also underlined by SPI approaches. The Capability Maturity Model (CMM) presented in chapter 4, addresses a key process area to requirements management as one of the first improvement areas [Paulk et al. 1993]. This means that one of the prerequisites for successful software development is, according to the CMM, the management of the requirements engineering process.

5.3.2 Practical experiences with requirements engineering

During the course of the research several industrial cases were carried out on requirements engineering. One of them will be used to illustrate some of the aspects involved in product quality requirements engineering in practice. The details of this case and the method used to perform the requirements engineering work have been published [Kusters et al. 1997] and will therefore not be included.

This example deals with a requirement stated by the purchase manager of an oil-company, regarding a cashing system:

'Each new version of the cashing system should first be installed in a pilot installation where it should be operational for at least three months before the final role-out of more cashing systems to other fuel stations can proceed.'

In this requirement a demand on the release process of a product is described, to evaluate the *quality* of the product.

What also becomes clear from this requirement is that it is not a product quality requirement in the sense that it specifies performance of a product. On the contrary, this is a demand from a stakeholder on a process. This requirement does address the specific expectations on the product by that person; however, product requirements are needed to communicate with the product developers. Without product requirements, the developers do not know what to build. If they know that a field-test will be held, it is still not clear to them what they will check during that field-test. So, quality requirements should also be stated regarding a product.

The underlying product quality requirement of this process requires can be abstracted based on one specific role of the person in question [Kusters et al. 1997]:

Role A: Buying a cashing system that is operational, contains the necessary functionality and works correctly

This role shows the underlying product quality requirements behind the demand for a field-test:

- the product should be available
- all functionality needed for operation on a fuel-station are included in the cashing system
- the product should only make correct transactions

The first requirement: 'availability' is a requirement on product reliability according to ISO 9126. This is a more general terminology for the requirement: developers understand what 'reliability' for a cashing system means.

This availability requirement is however not so stringent as stated. The cashing system is allowed to be out of operation, as long as it is not longer than 5 minutes and this should not happen more than once a week. With this information the requirements become objectively measurable: Maximum time to failure should be 7 days, and the maximum time to recover should be 5 minutes. If requirements are formulated in such a way, developers can start making design decisions to comply to these requirements.

What became clear from this case was that user based quality could be implemented by stating the product quality needs of the product stakeholders. Preferably in such a way that the stakeholder could express what was needed, while on the other hand, developers understood what they had to build. Specifying requirements in measurable terms (if possible) supports this.

During the case it also became clear that requirements elicitation, analysis and specification is a difficult task. Practical guidelines are required on how to carry out this

process in practice. During this case, mainly open interviews were used, but other means are probably possible.

Some problems became clear for product quality requirements engineering:

- Creating a 'good' set of product quality requirements is difficult. On the one hand these requirements are needed to express stakeholder perceptions, on the other hand these requirements should guide developers in their implementation work. This set of requirements should also be consistent, complete, unambiguous, etc. [IEEE 1994] [Davis et al. 1993]. Shortly, it is a complex and difficult task.
- It is not always practically feasible to fulfil stated requirements from stakeholders. Some requirements stated were, for example, considered to be technically unfeasible, or too expensive. This shows that analysis of the stated wishes is necessary, which is a delicate task, because once stakeholders have stated a wish they expect it to be granted.
- Once the product quality requirements are specified, it is still not clear how to fulfil them. Selecting specific actions in the process that contribute to the stated quality requirements should still be carried out, and it is not always clear how a certain product quality requirement needs to be fulfilled.

5.3.3 Refined definition of requirements engineering

This elaboration was started with a definition of requirements engineering from the literature; however, in this thesis requirements engineering is only partly handled in the way defined in the literature. In this section, a refined definition of requirements engineering is introduced based on the way the topic is handled in this thesis.

Firstly, product *quality* requirements are the focus of this thesis. This does however not mean that it is propagated to deal with functional and quality (non-functional) requirements separately in practice. It is recommended to address both functional and quality requirements at the same time; however, when current practice deals with requirements engineering it mainly focuses on the specification of functional requirements while ignoring quality requirements. In this thesis emphasis is therefore placed on quality requirements specifically and it is not intended in this thesis to specify functional behaviour of embedded products. The available approaches to specify functional requirements are assumed to be sufficient. As such product quality requirements engineering is an important source of information for the design of a product architecture that addresses both functional and non-functional requirements.

Secondly, in this thesis product quality requirements engineering is regarded as a support for product focused SPI. The main objective of requirements engineering in the context of this thesis is that it provides a product quality specification.

Thirdly, for elicitation of requirements there is a need to focus on product stakeholders. This is done, because chapter 1 indicated that user-based quality is important in the context of this thesis. Specification of product requirements is a sound step towards product quality, but these requirements should also include the user perception of quality. All stakeholders should therefore, have the possibility to make their product wishes explicit, in order to be considered for a product quality requirements specification.

Fourthly, these product quality wishes will be described in a measurable way to provide the product developers with an objective and unambiguous specification of product quality requirements [Gilb 1994].

Finally, these considerations lead to a refined definition of requirements engineering. Within this thesis requirements engineering is defined as *the process of collecting the wishes of all product stakeholders and transforming these wishes into a complete, consistent, unambiguous, and measurable product quality specification*.

5.3.4 Focused exploration of literature on requirements engineering

Much literature is available on requirements engineering. In this section only those findings from the literature will be presented that are in accordance with the previous refined definition of requirements engineering, and that will be used in the context of this thesis to construct practical guidelines for product focused SPI.

The findings from literature are presented along with the requirements engineering subtasks presented above. These findings are [ISO 9126 1991] [Gilb 1994] [Rumbaugh 1994] [Kotonya and Sommerville 1996] [Finkelstein et al. 1992] [Leite 1987] [Nissen et al. 1996] [Goguen and Linde 1993] [Siddiqi and Shekaran 1996] [Davis and Olsen 1985]:

- On product quality requirements elicitation:
  - An important aspect of requirements engineering is the existence of different stakeholders, each having a different viewpoint and therefore different requirements.
  - Inventarising the product stakeholders and their interrelationships is a first step for requirements elicitation. After that each stakeholder need to be involved to capture his or her wishes on product quality.
  - Techniques applicable for product quality requirements elicitation are: questioning, i.e. questionnaires and interviews, referencing, i.e. looking at previous similar products, analysing, i.e. refining from object and context models, and iterating, i.e. pilots and prototyping.
- On product quality requirements analysis:
  - Product quality requirements can be conflicting over stakeholders.
- On product quality requirements specification:

  - Once a selection is made from the product quality wishes they need to be transformed into a language understood by product developers.
  - It is most objective and therefore most practical to specify product quality requirements in measurable terms. Without measurable objectives it is difficult to fulfil quality requirements.
- On product quality requirements verification:
  - To support evaluation each requirement stated should be accompanied with a set of metrics and target values to those metrics.

5.3.5 Summary on requirements engineering

Requirements engineering is defined within this thesis as the process of collecting the wishes of all product stakeholders and transforming these wishes into a complete, consistent, unambiguous, and measurable product quality specification. Explicitly addressing stakeholders is a prerequisite of successful requirements engineering, since many stakeholders have their own demands regarding a product and its quality. The impact of requirements engineering on product quality is high, since without a clear and measurable specification of product quality it becomes difficult to guarantee that the development process will create a quality product. Requirements engineering is therefore definitely a working area for product focused SPI.

5.4 Process Engineering

Process engineering is often resembled with a concept which is discussed in literature under the term ‘Method Engineering’ [Hanani and Shoval 1986] [Wrycza 1990] [Brinkemper 1996] [Slooten 1995] [Punter and Lemmen 1996]. According to this literature, process engineering is the engineering discipline to design, construct and adapt methods, techniques and tools for the development of a specific software product [based on Brinkemper 1996]

Many standard development methods for software are available. Examples are Yourdon, ISAC, NIAM, SDM and SSADM. Each of these methods is based on generic development models such as the waterfall model [Royce 1970], the spiral model [Boehm 1987], or prototyping model [Gomaa 1990]. Such development methods consist of several actions that are taken to achieve explicit results. Examples of such actions are: requirements reviews, architectural design, sub-system implementation, unit testing, module inspection, etc.

These methods or parts of these methods, are differently used in different contexts. The (re)design of these methods and the sequence in which they are used is process engineering. Process engineering consists of [Brinkemper 1996]:

- characterisation of the project, during which the project is described according to a list of contingency factors, including the project objectives
- selection of process actions that suit the project, during which process actions are selected based on the project characterisation
- specification of the development process, during which the unrelated process actions are assembled into a project specific development process
- implementation of a process in the project, during which the designed development process is used within the project

So, process engineering contains a whole range of activities involved in designing a specific development process based on the specific needs of a specific project.

5.4.1 Relevance of process engineering

Process engineering is necessary, because, the best suitable development process for a specific project needs to be designed for its specific situation [Slooten 1995]. A generic development process for any type of product does not exist [Malouin and Landry 1983] [Benyon and Skidmore 1987]; dogmatic application of generic development processes is ineffective and inefficient [Rees 1982]. To have an effective and efficient development process, each development method therefore needs to be tailored to the organisation, project, and product needs and characteristics [Basili and Rombach 1988].

Process engineering is based on the notion that, development methods have to be continuously customised to the objectives [Davis et al. 1988]; however, such customisations are often done implicitly [Slooten 1995]. Process engineering strives to make these process adaptations explicit, thereby admitting that process customisation is a basic step in each development project.

5.4.2 Practical experiences with process engineering

During the course of the research several industrial cases were carried out on process engineering. One of them will be used to illustrate some of the aspects involved in process engineering. The details of this case have been published [Solingen and Uijtregt 1997].

In this case a focus is put on a product that had a severe reliability requirement. In the development process it was therefore decided to trial installations at pilot fuel-stations to test the reliability of the product, and detect any failures that were still present in the product. The main reason that this process action was selected, was that the organisation did not know the impact of other process actions, which meant that only one option for a final product reliability test remained: use in a real-life fuel station situation. Field tests are a means to retrieve objective information on the quality of a product, which often largely

differs from the quality predictions based on prediction methods [see e.g. Brombacher 1992 and Brombacher et al. 1996].

A field test is *a specific process action in which behaviour of a product is observed, when it is exposed to field conditions* [Solingen and Uijtregt 1997]. The field testing phase is the part of the development process during which a product is actually used at a customer site, while being closely observed. The output of a field test is a set of data on product performance during practical use.

During this field test specific product performance was monitored using internal problem data storage. This means that the product detects and reports problems by itself. One example from this field test is depicted in Figure 5-3. In this figure it can be seen that the product signals four defects every morning just before shop opening.

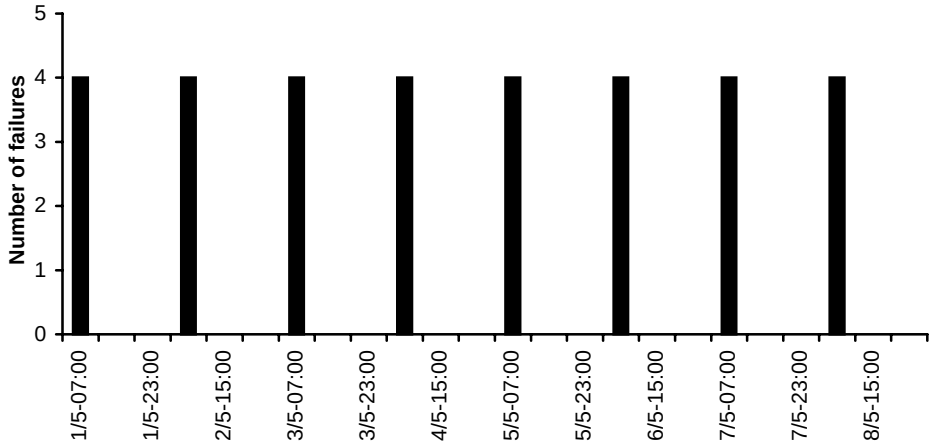


Figure 5-3: Occurrence of card-transaction failures

In Figure 5-3 the number of occurrences of a specific card transaction failure are shown. It appeared that there was a fault remaining in the start-up procedure of the card-server. This became clear, because this failure occurred four times every morning when the product contacted the national card-transaction network. Triggered by the data shown in Figure 5-3, the problem was fixed.

The effects of field tests on product quality could be evaluated in practice. During two field tests of a cashing system, measurements were conducted to find effects of field tests. Relevant data collection was defined in accordance to the measurement programme engineering guidelines presented in chapter 7. This resulted in several measurement results on the field test. It also became clear that the measurement outcomes from these two field-tests were by no means a guarantee that any following field-tests would yield similar

results. Many context factors impact the effects of a field test, which makes its results difficult to predict.

Furthermore, it appeared at the end of the two field-tests that the field-test process action was the only process action in that organisation on which such knowledge was now available. On no other process actions taken in the development projects, did baselines exist for their performances, nor for their impact on quality.

What became clear from this case was that the selection of process actions for a specific project is a difficult task. One of the main reasons is that the organisation was not aware of what the impacts were on product quality of the different steps in their process, which makes the tuning of that process difficult. The absence of valid process-product relationships makes it difficult to 'prove' the effects of a development process. Furthermore, the case concluded that the impacts of process actions on product quality can not be guaranteed. The impacts of a certain process action might be totally different in different situations.

5.4.3 Refined definition of process engineering

This elaboration started with a definition of process engineering taken from the literature; however, this thesis handles process engineering only partly in the way as defined in this definition. In this section, a refined definition of process engineering will be introduced, based on the way this topic is handled in this thesis.

Firstly, a limitation is put on product *quality*. Process engineering is therefore limited to those process actions that contribute to product quality.

Secondly, process engineering is carried out as a support for product focused SPI. The main objective of process engineering in this context is to design a project specific development process, which is likely to develop a product that complies to the product quality requirements. Process engineering therefore focuses on the identification and selection of those process actions that contribute to the required product quality, and the specification of these process actions in a development process model.

Thirdly, during execution of the specified development process, this process is measured. Measurement is carried out to evaluate whether the expected effects of process actions actually occur. If so, the effects of process actions are confirmed. If not, corrective action can be taken. The development process that is designed during process engineering, therefore needs to be 'measurable'.

Within the remainder of this thesis, process engineering is considered to be *the design of a measurable development process for the development of a specific product that complies*

to the product quality specification. A development process contains a set of process actions with explicit expected effects on product quality.

5.4.4 Focused exploration of literature on process engineering

Much literature is available on process engineering. In this section only those parts of the literature are presented that are in accordance with the previous refined definition of process engineering, and that will be used in the context of this thesis to construct practical guidelines for product focused SPI.

Before going to the findings of literature, one should note that to carry out process engineering correctly, the individual impacts of specific process actions should be known. In recent research it has been concluded that there is only a small number of publications that specifies the impacts of specific process actions on product quality [Soerjoesing 1998]. The main stream publications only contain a description of a technique, method or tool. This unavailability of validated process-product relationships confirms that software engineering is still an immature engineering discipline [Gibbs 1994] that lacks knowledge on its merits.

Findings from literature on process engineering are [Paulisch and Carleton 1994] [Humphrey 1989] [Davis et al. 1988] [Hamann et al. 1998a] [Hamann et al. 1998b] [Soerjoesing 1999] [Punter and Lemmen 1996] [Basili et al. 1994a] [Clements et al. 1995]:

- On characterisation of a project:
  - Make the project objectives explicit in a measurable way.
  - Use the product architecture as basis for communicating the system, its development and earliest design decisions.
- On selection of process actions:
  - Use process assessments to identify a baseline overview of the development process that normally is followed before designing a project specific process. Such a baseline can be used to indicate which process actions that contribute to product quality are usually used, and at which time, to which extend, by whom, etc.
  - Process modelling is a prerequisite to make the development process steps explicit, and should therefore be carried out for the ‘normal’ process. This process model also makes the development process baseline explicit.
  - To carry out process engineering in a right way, knowledge is needed on the individual effects of process actions on product quality.
  - Create models of the relationships between process actions and product quality. In this thesis such models are termed process-product relationship models.

- On the specification of the development process:
  - Make an explicit model of the project specific process.
  - Consider a process to be a set of phases that consist of a set of interrelated steps that each apply one or more techniques and result in one or more products.
  - Several modelling techniques are available to make such process models.
- On implementation of the process:
  - Activities need to be undertaken to implement the designed process model, actually in the development project.
  - The sequence of steps in process model should therefore be included in the project planning, procedures and reports.

The details involved in the implementation work are indeed relevant; however, it is outside the scope of this thesis. For example the importance of designing a good product architecture and the impact of this architecture on product quality has been addressed in chapter 3. The product architecture is the object used for communicating system issues and discussing what can be done or cannot be done by the system under development. How this architecture is developed best is, however, out of the scope of this thesis. In the context of this thesis, the delivery of a project specific development process model is sufficient. This thesis assumes that the actual development process is carried out as described.

5.4.5 Summary on process engineering

Summarising, process engineering is defined in this thesis as the design of a measurable development process for the development of a specific product that complies to the product quality specification.

It was concluded that the current state of knowledge on software engineering lacks validated models of the relationships between process actions and product quality. Such product-process relationship models (PPRM) should therefore be created and validated. If a relationship is confirmed in one situation there is no guarantee that that relationship also exists in a different situation, and therefore it needs to be monitored if the expected effects of process actions (re-)occur. This monitoring by means of measurement will be further described in the next section on measurement programme engineering.

5.5 Measurement Programme Engineering

Measurement programme engineering is described in the literature as the design and implementation of a set of process, product and resource metrics, to achieve predefined objectives within an organisation [based on Fenton and Pfleeger 1996].

The design and implementation of a measurement programme carried out in practice contains three main phases [Solingen and Berghout 1999]:

- Definition of the measurement goals, questions, metrics and hypotheses. During this phase the actual design of the measurement programme is carried out.
- Data collection of the measurement data according to the plan. During this phase the data collection procedures are designed and implemented in the organisation.
- Interpretation of the measurement data by the software development team. During this phase feedback is given to the project team, and the collected data is analysed to evaluate hypotheses, answer questions, and attain the measurement goals. Conclusions on the measurements and knowledge packages are stored for future application.

So, measurement programme engineering contains a whole range of activities involved in designing and carrying out measurement in practice.

5.5.1 Relevance of measurement programme engineering

Measurements are objective and can therefore be used as a mechanism for feedback and evaluation, such as for model building, abstracting knowledge, validating process actions, creating management information, evaluating projects, etc. [Basili et al. 1994b] [Grady 1994] [Briand et al. 1996] [Fenton and Pfleeger 1996] [Stark et al. 1994]. Although, subjective measurements are also often taken, these are more general and better comparable than the subjective expression of a person's perception on a specific topic. The main benefits of software measurement are [Hall and Fenton 1994]: objectivity in process management, improved product quality, increased engineer productivity, improved organisational communication, and improved engineer work pride.

The main purpose of measurement programme engineering in the conceptual model is to provide feedback to the project. First of all, measurement enables the provision of information on conformance of the (intermediate) product to the product quality specification. With this information, progress of the project can be traced and corrective action can be taken. Secondly, measurement enables the evaluation of the extend in which process actions result in the intended effects. Especially for new process actions with which less experience is available, measurement is a powerful tool. It is also possible to provide information on performance indicators of process actions that much experience is available on, in order to support better control of, or even optimisation of, such process actions.

5.5.2 Practical experience with measurement programme engineering

During the course of the research several case-studies were carried out on measurement programme engineering. One of them will be used to illustrate some of the aspects involved in measurement programme engineering in practice.

In this case-study a project is described during which product reliability was measured. The measurement programme was active for a development project that developed a low-end cash register. The results consisted of more than the measurement data alone. They also contained conclusions drawn by the project team during feedback sessions, and corrective actions that were defined based on the measurement data.

It was decided to have product reliability as measurement goal. This was done because 'reliability' was the most important quality requirement for the product, and the practical relevance of reliability measurement was clear to the project team. A selection of conclusions drawn and actions taken by the project members based on the measurement results is presented, and illustrated along the product failure trend in Figure 5-4.

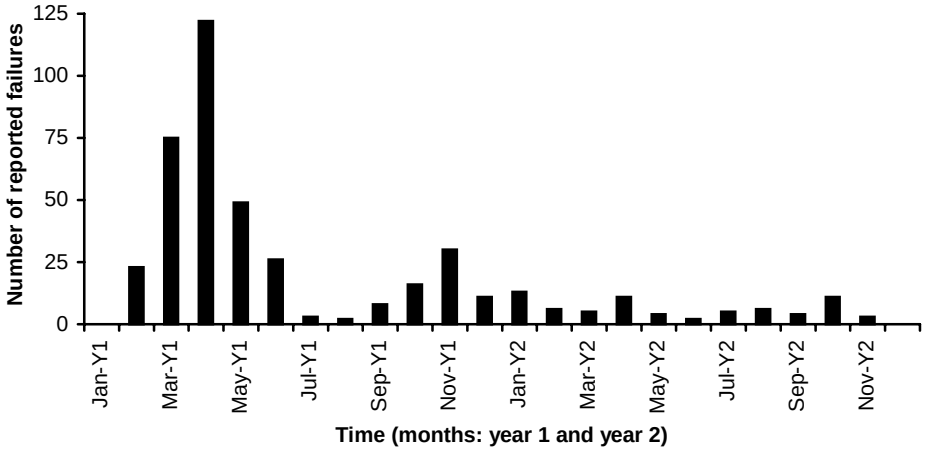


Figure 5-4: Number of failure reports on product under development

In Figure 5-4 the amount of failures is shown that were reported on the product over time. Each failure is recorded only once: at the moment it is detected for the first time. In Figure 5-4 a peak is present in April and November of year 1. The number of failure reports on the product approaches zero at the end of year 2. During this period changes were made to the product to solve detected problems, and to expand the functionality of the product. As such this figure does not present a reliability trend of a product in operation, but presents the amount of known problems in the product. Each known problem is expected to result in regularly failures, thus influencing product reliability.

The evaluation of conformance to the reliability requirements of the product is supported by this chart, because it provides an objective overview of the failures of the product.

Beside the fact that this chart shows an aspect of product reliability, it also supports in the evaluation of effects of process actions. For example in April of year 1, a Novice User Test was executed and therefore the number of failure reports was high. One of the conclusions was that such a Novice User Test is useful and contributes to product reliability. Of course other prerequisites were stated for such tests, but based on other measurements and experiences.

Measurements such as in Figure 5-4, should always be fed back to the project team for interpretation to prevent drawing wrong conclusions. For example, the low number of failures during July and August of year 1 reflect the summer holidays during which no detection activities were executed on the project. A person that did not know these process details might have concluded, wrongly, that the product was becoming reliable.

The project team also learned that drawing conclusions on the current reliability of the product based only on Figure 5-4 is not possible, because information is not included on the amount of effort spent on finding failures. For example, in May of year 1, twice the amount of failures occurred than in June of that year. Conclusions on reliability can not be drawn, because it is not clear how much effort was spent to find these failures. If other measurements had shown that in both months the amount of effort spent on testing was the same, better conclusions on reliability could have been drawn.

Finding out such relationships is an important aspect of measurement programmes: measurement supports a learning process in which project teams learn to distinguish which aspects are important and how process-product relationships exist.

Other aspects that were learnt from this project were:

- Modules with high user interaction produce much more failures whenever a fault is present and should therefore be reviewed quite intensively.
- Module size in e.g. number of source lines appears to be an indicator for faults. This conclusion is drawn based on the increasing fault density for large modules. The measurement programme also confirmed that there is a relationship between cyclomatic complexity and module size.
- Availability of the right test equipment increases detection of faults before delivery. This conclusion is based on the identification of the project team that most failures found after release were caused by a different configuration than available during test. They took action to make sure the appropriate equipment became available. The measurements supported them to convince management to make this investment.

What became clear from this case-study was that measurement programmes indeed have two purposes for product development:

- to monitor current status of the product, and as such the conformance of the product to the requirements
- to observe the relationship between a process action and its impact on product quality

Relationships between a process action and the product were monitored to find out whether they re-occurred as expected, but such relationships were also discovered if the project team did not know they existed. So process-product relationships can be determined and evaluated.

In this case it is also illustrated that practical guidelines on carrying out measurement programme engineering in practice are needed. Setting up a measurement programme is a difficult task and should be carried out correctly. Support from guidelines needs to be available for practice. Furthermore it appeared that feedback to the project team, and interpretation of measurement data by the project team is crucial to the success of measurement programmes. In fact it became clear that measurement programmes are learning processes, in which project developers learn about the impacts of their way of working [Latum et al. 1996][Solingen et al. 1997].

5.5.3 Refined definition of measurement programme engineering

The definition of measurement programme engineering as defined at the start of this section will be refined to fit this thesis. There are no issues for a major change of the definition; however, the objectives of measurement programme engineering in this thesis are defined, and can therefore be integrated in the definition. The two objectives of measurement programme engineering as defined in section 5.1.3 are:

- to evaluate conformance of the end product to the product quality requirements
- to evaluate relationships between process actions and product quality

Measurement programme engineering is therefore defined in this thesis as *the design and implementation of a set of process, product, and resource metrics, to evaluate product quality and evaluate process-product relationships.*

5.5.4 Literature on measurement programme engineering

Much literature is available on measurement programme engineering. This section will only present those findings from the literature that are in accordance with the previous definition of measurement programme engineering, and that will be used in the context of this thesis to construct practical guidelines for product focused SPI.

Measurement programme engineering is centred around explicitly stating measurement goals. Measurement goals focus on *understanding*, *controlling* or *improving* [Basili 1993] [Fenton and Pfleeger 1996] [Solingen and Berghout 1999]. Within this thesis this means:

- understanding: learning effects of process actions
- controlling: comparing effects of process actions with the expected effects, or monitoring the product
- optimising: improving effectiveness and efficiency of specific process actions

General findings on measurement programme engineering in literature are [Hall and Fenton 1994] [Paulisch and Carleton 1994] [Basili et al. 1994b] [Basili and Rombach 1988] [Latum et al. 1996] [Latum et al. 1998] [Briand et al. 1996] [Pfleeger 1991] [Fenton and Pfleeger 1996] [Möller and Paulisch 1993] [Grady and Caswell 1987] [Grady 1992]:

- On *definition* of a measurement programme:
  - Measurement should be performed towards explicitly stated goals. This 'goal-oriented measurement' helps to provide traceability between improvement goals, measurement goals, and measurement.
  - Have baseline performance data available before improvements are implemented: combine measurement with process improvement goals.
  - Subjective as well as objective metrics are required to attain measurement goals.
  - Most aspects of software processes and products are too complicated to be captured by a single metric.
  - The measurement process must be top-down rather than bottom-up to define a set of operational goals, specify the appropriate metrics, permit valid contextual interpretation and analysis, and to provide feedback for tailorability and tractability.
  - The development and maintenance environments must be prepared for measurement and analysis.
- On *data collection* for a measurement programme:
  - Multiple mechanisms are needed for data collection and validation.
  - To evaluate and compare projects and to develop models we need a historical experience base that should evolve into a knowledge base.
  - Collect data on the development process only: do not measure people: do not ignore the fear of people for metrics.
- On *interpretation* within a measurement programme:
  - Interpret measurement information based on characterisation and understanding of the organisational context, environment and goals. Make effective use of graphics for presenting measurement results.

- Models and metrics cannot be directly transferred between different environments.
- Metrics must be associated with interpretations, but these interpretations must be given in context.
- Crucial to the success of measurement programmes is the analysis of the measurement data. This analysis should be carried out by those people that have knowledge on the context in which the data was collected, and ideally consists of those people that actually collected the data.
- Use measurement feedback sessions to analyse data, answer questions and define actions.
- Integrate software measurement in the development process. Prevent separate 'metrics' groups that perform 'ivy tower' analysis.

5.5.5 Summary on measurement programme engineering

Measurement programme engineering is defined in this thesis as the design and implementation of a set of process, product, and resource metrics, to evaluate product quality and to evaluate process-product relationships. As this definition indicates, measurement programmes has two purposes:

- the evaluation of the conformance of a product to the product quality requirements specification
- the evaluation of the relationship between process actions and product quality

These two objectives of measurement are highly relevant when considering product focused SPI. Measurement programme engineering is therefore definitely a working area. A set of findings from the existing literature was listed to be considered for measurement programme engineering as it is used in this thesis. Furthermore, the case-study concluded that a measurement programme is a learning process, in which developers learn about the impacts of their way of working.

5.6 The RPM Conceptual Model

An elaboration of the three working areas of product focused SPI were presented in the previous sections. In this section these working areas are combined into a conceptual model, which structures and operationalises product focused SPI. This conceptual model is a control model that defines the control loops present for product focused SPI. Product quality will be controlled within this model, through controlling the development process and the product quality requirements specification.

The conceptual model for product focused SPI is called the RPM model and consists of the three working areas (section 5.2):

- Requirements engineering, which is the process of collecting the wishes of all product stakeholders and transforming these wishes into a complete, consistent, unambiguous, and measurable product quality specification
- Process engineering, which is the design of a measurable development process for the development of a specific product that complies to the product quality specification
- Measurement programme engineering, which is the design and implementation of a set of process, product, and resource metrics, to evaluate product quality and evaluate process-product relationships

Furthermore, the conceptual model contains three work products:

- Product quality specification, which are the documented product quality requirements, specified in a complete, consistent, unambiguous, and measurable way.
- Development process model, which is the project specific model of the steps performed to result in an embedded product. Some of these steps are what this thesis calls ‘process actions’, which are selected for their explicit expected effect on product quality.
- Measurements, which are the collected data and their analysis by the development team regarding the conformance of the product to the product quality specification, or regarding the effectiveness of specific process actions.

These working areas, their work products and interrelationships are depicted in Figure 5-5.

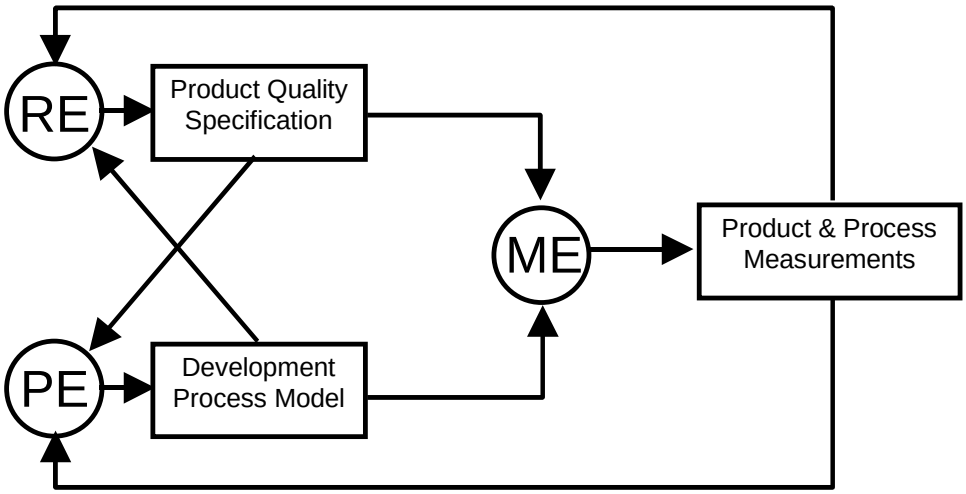


Figure 5-5: The RPM conceptual model for product focused SPI

Requirements engineering is carried out to capture the quality requirements for a product. The resulting product quality specification is used by process engineering to design an appropriate development process. During process engineering it might become clear that

some requirements can not be fulfilled. These requirements are fed back to requirements engineering to adapt the requirements specification. Requirements engineering and process engineering perform an iterative negotiation process in which a trade-off is made between the product quality requirements and the related options in the development process.

The architecture of both the product and the software is a useful object in this communication process. The architecture enables the discussion of such early design decisions and it is possible to estimate based on an architecture the feasibility to fulfil certain product quality requirements [Clements and Northtrop 1996].

In addition, a measurement programme is started that monitors the process in its ability to result in the expected effects on product quality, and furthermore measures the conformance of a product to the product quality specification. Measurement programme engineering feeds back information to process engineering on the effectiveness and efficiency of certain process actions, and to requirements engineering on the conformance of a product to the specification.

The conceptual model in Figure 5-5 contains 3 control loops (see Figure 5-6):

- Product quality loop, which controls the extend in which a product under development complies to the product quality requirements
- Product-process loop, which controls the extend in which the development process is capable of fulfilling the product quality specification
- Process quality loop, which controls the extend in which specific process actions result in the expected effects

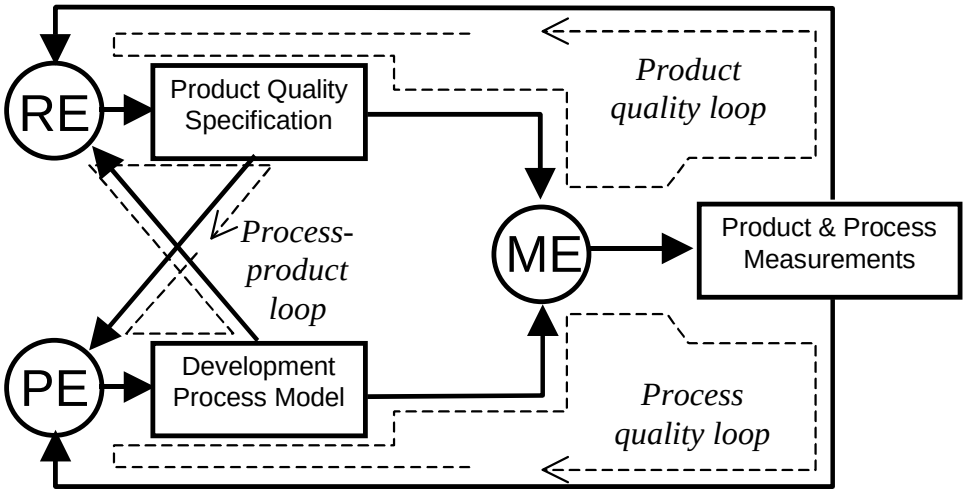


Figure 5-6: The RPM conceptual model and its control loops

5.6.1 Strengths of SPI compared to the conceptual model

In section 5.1 the extent was discussed to which the proposed expansions matched the criteria for product focused SPI in chapter 4. These criteria were designed to overcome some fundamental weaknesses of existing SPI approaches for embedded product development. The conceptual model constructed in this chapter should not only solve those weaknesses, but also maintain the strengths of existing SPI methods. The extent to which the strengths of existing SPI approaches are maintained within the conceptual model of Figure 5-5, is therefore discussed in this section.

The following strengths are also covered by the RPM model:

- *SPI is based on best practices*. This strength is retained, because the application of assessments during process engineering is recommended as a characterisation of current practices. The 'best practice' knowledge from assessment approaches is therefore also included in the RPM conceptual model.
- *SPI is a management tool for improvement*. Since, the RPM conceptual model explicitly addresses the formulation of product quality goals, it provides a management tool for improvement. The objectives for a product are set, the process is defined based on these objectives, and a measurement-based evaluation is carried out. This structure can be used to manage a project, and is therefore a management tool to control product quality improvement.
- *SPI gives explicit priority to quality*. It might be clear by now that the RPM conceptual model puts product quality as the basic objective, thereby giving explicit priority to quality of both the product and the process.

There are also some strengths of existing SPI approaches, however, that are not covered by the RPM conceptual model:

- *SPI changes are prescribed*. Current SPI approaches prescribe the process areas in which an organisation needs to change. This is different from what the RPM approach does. The RPM approach prescribes tasks to do to identify what to change. Instead of prescribing general process changes for every software development process, it prescribes a process that helps in learning where the changes are mostly needed. This is fundamentally different from existing SPI approaches; however, this is probably better, because the end result is the same: a list of improvement areas, while the RPM approach also guarantees that these proposed changes are relevant to the specific project. No action will be taken to overcome this issue.
- *SPI provides a vision*. The RPM conceptual model links product quality goals to the development process that has to create that quality. This goal-oriented structure is, however, mainly focused to control product quality within one specific project. Improvement over projects is not included, which causes long term improvements

of the development processes not to be established. This will be addressed further in this thesis.

The main conclusion of this section is that the RPM conceptual model presented in Figure 5-5, contains the strengths of existing SPI approaches that were identified in chapter 4 largely; however, a long term vision of improvement over projects is missing.

5.7 Need to investigate learning theory

Until now this chapter has introduced the RPM conceptual model, investigated the components of this model, and presented practical experiences with each component. Based on this, it becomes time to investigate whether this RPM model can work in practice.

The practical experiences with the three working areas already indicated that there are some specific issues related to the practical implementation of the RPM model:

- For requirements engineering there are difficulties in creating a 'good' set of product quality requirements, because there is not much knowledge on creating good requirements. Furthermore it is not always feasible to fulfil stated requirements from stakeholders.
- For process engineering there is a clear absence of knowledge on valid process-product relationship models, which makes it difficult to 'guarantee' the effects of a development process. Furthermore, the impacts of process actions on product quality are not generic. The impacts of a certain process action might be totally different for different situations.
- It was identified for measurement programme engineering that such programmes should be carried out carefully to be successful, and that feedback to the project team, and interpretation of measurement data by the project team is an important success factor due to the transfer of knowledge among the people.

Furthermore, it was concluded that, although the RPM model contains most strengths of existing SPI approaches, a long-term improvement vision is lacking. Knowledge transfer from one project to the other is not included in the RPM model.

All these issues are centred around one problem: missing knowledge, and especially missing knowledge in the software engineering domain. This is a problem that can not be solved within this thesis. There is, however, a basic solution to missing knowledge, and that is *learning*. When certain knowledge is not available it can be acquired. Product requirements can be determined by making them explicit, implementing them and carrying out an evaluation. Feasibility of product requirements can be determined through trying to fulfil them. Product-process relationships can be determined by stating expectations and evaluating whether these expectances were right. Situational effects of process actions can

be determined by measuring them. Carrying out measurement programmes is an organisational learning process in which knowledge is created from current practices [Basili et al. 1986] [Solingen et al. 1997].

From now on, this thesis will take the view that product focused SPI is a learning process; therefore, a more thorough analysis of learning processes is relevant. The need for integrating learning concepts into software engineering has been supported by several authors before [Brooks 1975] [Bemelmans 1998] [Basili et al. 1994a]. It seems, therefore, wise to integrate concepts from learning theory into the RPM conceptual model to include such learning cycles explicitly. The lack of knowledge on software engineering cannot be solved within this thesis, however, it is possible to make sure that the RPM approach works towards a solution.

So, learning theory needs to be investigated to provide guidelines on how to carry out the model of Figure 5-5 in a good learning way. From these learning theories it is expected to comprise concepts and guidelines that will support in making the RPM model practically applicable in industrial contexts. The learning literature is investigated in the following chapter.

6. Learning: the basis of improvement

In the previous chapters a conceptual model was constructed for product focused SPI. Based on an analysis of this model it was concluded that learning theory needs to be incorporated. Learning is often the main process in organisations [Garvin 1993], and it is definitely so for software development, because this development consists of mainly cognitive tasks.

In this chapter learning theory is explored and an indication is given of which parts of learning theory are applicable within the context of product focused SPI. Based on the learning concepts found in learning theory, the RPM conceptual model is adjusted. A set of learning enablers is also identified and it is discussed how these enablers should be interpreted within the three working areas of the RPM conceptual model.

6.1 Restricting learning concepts

This thesis does not contain a full analysis of learning theory or an extensive discussion of the pros and cons of certain views. Learning theory is addressed in this thesis with an explicit purpose: retrieving operational guidelines to increase the effectiveness of the learning processes that are present during product focused SPI. In order to do so, in this thesis only those parts of learning literature will be addressed that contribute to this objective.

Learning is the process by which existing knowledge is enriched or new knowledge is created [Weggeman 1997]. Learning deals with expanding *knowledge*. Knowledge is the personal ability that enables a person to perform a certain task [Weggeman 1997]. This ability is the product of information (I), experience (E), skill (S) and attitude (A) of a person at a certain time (K=I·ESA) [Weggeman 1997].

Not all types of knowledge are relevant for this thesis. Knowledge is often subdivided into declarative knowledge and procedural knowledge [Anderson 1990]. Declarative knowledge is knowledge about facts and things, often learned in a classroom setting. Procedural knowledge is knowledge about how to perform various cognitive activities, often learned in a ‘community of practice’ [Wenger 1990]. Since, in this thesis learning theory is investigated with respect to the impacts of processes on product quality in a practical environment, the focus of this chapter is on procedural knowledge. Declarative

knowledge on the relationships between process and product are unexplored for software engineering (see chapter 5), and thus not part of computer science curricula in education.

Several classifications of the process of learning are described in the literature. For example: cognitive versus motor learning [Ayas 1997], explicit versus implicit learning [Swieringa and Wierdsma 1990], or rationalistic versus empirical learning [Weggeman 1997]. Nonaka and Takeuchi distinguish four different learning *processes* [Nonaka and Takeuchi 1995]:

- 'socialising': a learning process between people in which *implicit* (tacit) knowledge is transferred by copying, imitating, master/pupil relationships, and experiencing by trial and error
- 'externalising': a learning process, individual or between people, in which implicit knowledge is made explicit by for example model building, dialogues, and hypothesis formulation
- 'combining': a learning process in which explicit knowledge from different sources is combined by for example: studying, analysing, reconfiguring, and integrating
- 'internalising': an individual learning process in which explicit knowledge is made implicit through learning by doing, creating routines, and enlarging operational efficiencies

Although, all four learning processes are present during, and relevant for, product focused SPI, this thesis focuses on the explicit learning processes: externalising and combining. This decision was made because the approach that is constructed addresses learning as an explicit task and provides guidelines to perform these tasks. This is by definition an explicit process.

With the decision to focus on explicit learning processes and on procedural knowledge in mind, this thesis continues with an exploration of learning theory. First, individual learning will be considered, followed by group learning.

6.1.1 Individual learning

During individual learning, the knowledge of one single person expands. A well-known theory is Kolb's Experiential Learning theory [Kolb 1984] that defines an explicit learning process. According to Experiential Learning theory, learning is a process in which *experiences* are *transformed* into knowledge, through model building and model testing.

Experiences are divided into concrete experiences: observations like seeing, feeling or hearing, and abstract conceptualisations: theories and models about observations and their relationships. Transformations are divided into reflective observations: analysing observations and developing new models and theories, and active experiments: testing

models and theories in practice. According to Experiential Learning theory, neither the experience nor the transformation alone is a sufficient condition to achieve learning.

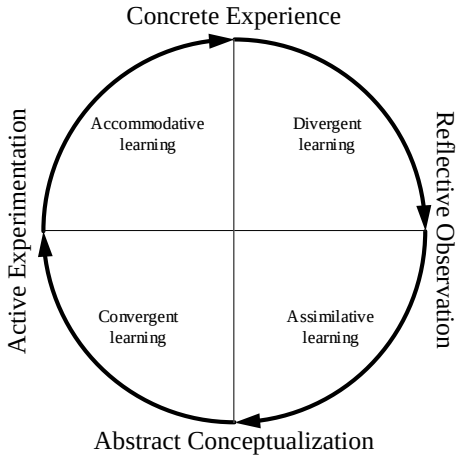


Figure 6-1: Experiential Learning [Kolb 1984]

Following the different classes of experience and transformation, four different modes of learning are distinguished. These modes are:

- 'divergent learning' during which observations are analysed
- 'assimilative learning' during which models are built
- 'convergent learning' during which models are tested in practice
- 'accommodative learning' during which experiments are observed

According to Kolb, the combination of these four modes of learning produces the highest level of learning. The combination requires the learning process to include: '*observing phenomena, analysing them, developing models and theories about them and testing these theories and models in practice'* [Kolb 1984]. In fact this is what was identified in chapter 5: product quality control in practice is enabled through model building and model testing of process product relationships.

6.1.2 Group learning

When considering learning in software process improvement, it is important to realise that work is performed in an industrial environment. An industrial environment demands group learning. Software development and process improvement is carried out within teams, projects, departments or companies; it always concerns a group of people. The improvement objectives and learning processes are therefore shared.

The term 'group learning' indicates that a set of people, over a period of time, share the same learning goals and learning process. In such a situation, knowledge has to be shared

among organisational members and to contribute to the synergy of the organisation [Jelinek 1979]. This is also often termed: 'organisational learning'.

Organisational learning is defined as *a skilled process in which knowledge is created, acquired, and transferred, and through which behaviour is modified based on the new knowledge and insights* [Garvin 1993]. It is important to note that organisations cannot learn: the individual *people* can learn and learn together [Weggeman 1997].

This definition reflects that learning happens when new insights arise. Sometimes they are newly created, sometimes they arrive from outside the organisation or are communicated by knowledgeable insiders. Such new insights are, however, not enough. *Without accompanying changes in the way that work gets done, only the potential for improvement exists* [Garvin 1993]. George Huber states similarly that learning occurs when '*the potential behaviours are changed'* [Huber 1991]. Behaviour does not need to be changed for every situation, but the potential ways of working need to be expanded. So, effective learning results in altering (potential) behaviour. If behaviour is not changed, learning has apparently not occurred.

Argyris and Schön make a distinction between two modes of learning: single loop and double loop [Argyris and Schön 1978]:

- Single loop learning. This is learning in which the actor only learns within the confines of his or her theory in use. There is a focus on the operational level: based on detecting and correcting errors, competencies and routines.
- Double loop learning. Double loop learning starts when an event is diagnosed as incompatible with the actor's current theory in use. With double loop learning current theory and models are altered through new insights.

In practice most organisations are only focussed on single loop learning [Argyris 1993]. Optimisation is only done within the current way of working. This in itself is not wrong. Through repetitive experiences, organisations get skilled in their work, and create competitive advantages based on these skills.

Sometimes new approaches become available that an organisation has no experience with. In such cases it might be better to switch to such a new approach, because it fits better than the historic approaches. This is double loop learning, which many organisations tend to see as a threat because it conflicts with existing and established habits.

It is also dangerous for an organisation to constantly adopt new ways of working, because all knowledge gained until then might immediately become outdated. '*The known can be in many situations be preferred over the unknown'* [March 1991]. A balance should be found in optimising the current processes (single loop learning) and experimenting with new theories and approaches to find out whether those are much better than existing ones (double loop learning). So, learning theory promotes a parallel application of optimisation

of current practices and experimentation with new ones. This idea should be considered for the RPM conceptual model.

The skills and capabilities of learning organisations are divided over three classes [Senge 1990]:

- 'aspiration': the capacity of individuals, teams, and eventually larger organisations to orient toward what they truly care about, and to change because they want to, not just because they need to
- 'reflection and conversation': the capacity to reflect on patterns of behaviour and assumptions deeply hidden in a persons behaviour, both individually and collectively
- 'conceptualisation': the capacity to see larger systems and forces at play and to construct public, testable ways of expressing these views

According to Senge, there are three groupings of learning skills. Firstly, there is the motivation to learn and improve. This includes having time for learning, learning objectives, interest in learning, etc. Management commitment for learning tasks is also one of the aspects that falls under aspiration. Secondly, there is the willingness to discuss deep assumptions. This is what Argyris and Schön call 'double loop learning'. Finally, there is conceptualisation, which corresponds with model building and testing of the experiential learning theory [Kolb 1984]. These three skills and capabilities for establishing learning need to be addressed by the RPM conceptual model.

Learning theory supports that a learning method should specify learning goals explicitly [Garvin 1993]. Defining these goals is difficult, but in a business environment it makes sense to base them on business goals. These goals can be different for different organisations. Differences include the market in which an organisation operates, the type of product that is delivered, the organisation of the development teams, or the country in which the products will be used. Learning practices should be directed to goals of the organisation, which can be made operational by, for example, managing on performance indicators [Garvin 1993]. *Goals for learning always vary between organisations, because of different strategies* [Agarwal et al.1997]. So, learning theory indicates that learning objectives should be situational, depending on the specific needs of an organisation.

In chapter 5 it was indicated that the RPM conceptual model does not prescribe generic process improvement goals for all organisations. This is different from current SPI approaches that prescribe a generic set of priorities and sequence in which improvements should be implemented. In chapter 5 it was discussed that defining organisation specific improvement objectives is probably better. Learning theory appears to support this decision. The way in which these goals are reached is not prescribed: the learning process of each organisation can be different, because the context in which learning is established is also different.

The final aspect of organisational learning relevant for this thesis is based on a phenomenon called ‘creative tension’ [Senge 1990]. This is the difference between current reality and a desired future. The gap between the current reality and the desired future should not be too large, because the objectives of the people become too abstract and concrete actions towards improvement are not clearly visible. On the other hand the gap between current reality and the desired future should not be too small either, because this will result in no action at all, since the need for action might seem unnecessary. Note the resemblance between ‘creative tension’ and ‘assessment based’ improvement programmes for software engineering, in which yearly benchmarks are used to set next years objectives. This creative tension principle will be adopted, because it appears practical to steer learning towards reachable objectives. So an improvement method for embedded product development should focus on goals that are reachable and within the principle of ‘creative tension’.

Beside general guidelines on implementing organisational learning, learning theory also provides several criteria for successful learning. Such ‘learning enablers’ are also relevant for this thesis and are therefore be handled in the next section.

6.1.3 Proposed incorporation of learning concepts in the RPM conceptual model

Based on the foregoing analysis of learning theory, the time has come to summarise which concepts need to be incorporated in the RPM conceptual model for product focused SPI, which was presented in chapter 5. The following learning concepts will be incorporated:

1. Learning is the process by which existing knowledge is enriched or new knowledge is created [Weggeman 1997]. Knowledge is the personal ability that enables a person to perform a certain task [Weggeman 1997]. Transferring implicit (tacit) knowledge is not addressed in this thesis.
2. Organisational learning is defined as a skilled process in which knowledge is created, acquired, and transferred, and through which (potential) behaviour is modified based on new knowledge [Garvin 1993]. Organisational learning is done within groups of people. Organisations do not learn, but the people in those organisations are learning.
3. During learning experiences are transformed into knowledge, through *model building* and *model testing*. A learning process contains: observing phenomena, analysing them, developing models and theories about them and testing these theories and models in practice [Kolb 1984].
4. Two types of learning exist in a learning organisation [Argyris and Schön 1978]:
   - *Single loop* learning: during which the current way of working is optimised.

- *Double loop* learning: during which current ways of working are rigorously changed by adopting alternative theories.

5. Establishing skills and capabilities of a learning organisation consist of [Senge 1990]:
   - *Aspiration*: Creating a context in which people are willing, motivated and able to learn.
   - *Reflection and Conversation*: Creating a context in which double loop learning is established.
   - *Conceptualisation*: Creating a context in which knowledge is made explicit by model building and model testing.
6. Learning goals are different for different situations, and these learning goals should be stated explicitly [Garvin 1993] [Agarwal et al. 1997]. These goals should be in accordance to the principle of 'creative tension' [Senge 1990], which means that they are a challenge to reach them, but that it is also feasible to reach them.

Although learning theory provides insights in how to increase learning within product focused SPI, learning theory provides no practical guidelines on *how* work gets done in a learning organisation. Organisational learning theorists themselves confess that their concepts do not provide any framework for action in reality [Garvin 1993] [Senge et al. 1994]. It is, however, indicated that cyclic models from the 'quality area', such as Plan-Do-Check-Act [Deming 1986], provide systematic methods that can be used to integrate learning into practice [Garvin 1993] [Senge et al. 1994].

6.2 Improving the RPM conceptual model

In chapter 5, the RPM conceptual model was presented, this contains the three main working areas and their inter-relationships for product focused SPI: requirements engineering, process engineering and measurement programme engineering. In this section the RPM conceptual model will be altered based on the selected learning concepts. What does learning theory confirm or potentially change in the RPM conceptual model of chapter 5?

Firstly, definitions of learning, knowledge and organisational learning are provided (6.1.3 point 1 and 2). It becomes clear that the focus of learning within an organisational setting should focus on the learning process of people, and on the group learning process of these people. Improving software process and product quality requires a focus on the people that develop these products and use these processes.

Secondly, in learning theory centring the learning process around model building and model testing is promoted (6.1.3 point 3). This confirms that the *modelling* of process-product relationships is a good approach, and furthermore confirms that measurement is

rightfully selected as a major component of the model, as it supports in objectively testing and evaluating of the models built. Measurement is used in the conceptual RPM model to learn the effects of process actions on product quality, and to evaluate the conformance of the end-product to product quality needs. These two purposes are both based on model building and model testing and therefore stimulate learning.

Thirdly, there is a need to distinguish two types of learning: single loop learning and double loop learning (6.1.3 point 4). For the conceptual RPM model this implies that single loop improvement must be distinguished from double loop improvement. This was not done in chapter 5, and this concept should therefore be included.

Fourthly, learning theory indicates that there are three types of skills and capabilities for successful learning: aspiration, reflection & conversation, and conceptualisation (6.1.3 point 5). This confirms that the organisational context and culture are an important prerequisite for success, which is also the case for improvement programmes. Furthermore it confirms that, making implicit models explicit and testing and measuring them, supports a learning approach.

Finally, making goals explicit and obtainable using the principle of ‘creative tension’ is prescribed in learning theory (6.1.3 point 6). This confirms that learning should be a direct objective in SPI programmes. Furthermore, the learning objectives must be made explicit. It also confirms that assessments are a powerful tool for process improvement, as they are based on the creative tension principle. Assessments can be used to determine the baseline for the current status of the software process and should therefore remain as an important component. This is already the case in the conceptual model, since process assessments are part of process engineering.

6.2.1 Adjusting the RPM model based on learning theory

It is indicated in learning theory that many learning concepts are already included in the RPM conceptual model of chapter 5; however, two changes were proposed from learning theory in section 6.1.3:

- distinguish between single loop and double loop improvement
- make learning goals explicit and address them explicitly

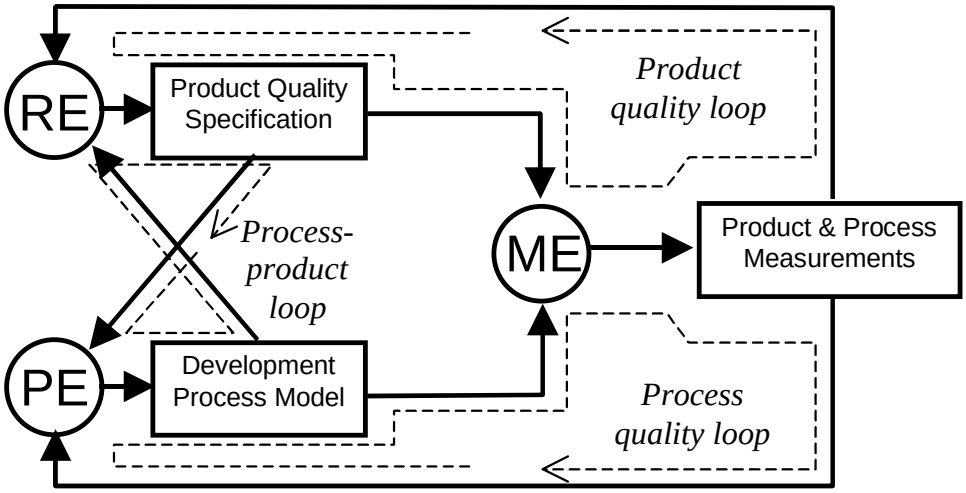


Figure 6-2: The RPM conceptual model (from chapter 5)

In Figure 6-2 the conceptual model as presented in chapter 5 is shown. Three main processes are included: requirements engineering (RE), process engineering (PE) and measurement programme engineering (ME). During requirements engineering a product quality specification is developed. During process engineering a development process model is developed, which contains a specification of the process that should be followed to result in a product that complies to the product quality targets. The results of measurement programme engineering are process and product measurements that identify discrepancies on the product level when product measurements differ from their product targets, and discrepancies on the process level when process measurements differ from their expected effects. Figure 6-2 includes three control loops:

- The product quality loop, which controls the compliance of the developed product to its product quality requirements.
- The process product loop, which controls the ability of the process model to comply to the product model, and in case of differences adapts the process model or the product model. This loop expresses the trade-off between process aspects: cost, duration, risk, etc. and product aspects: quality, functionality, etc.
- The process quality loop, which controls the compliance of the development process to its expected effects.

The conceptual model of Figure 6-2 should be expanded to include the two proposed changes from learning theory: distinguishing between single and double loop improvement, and explicitly addressing learning objectives.

Taking a closer look on Figure 6-2 it becomes clear that this conceptual model only addresses the single loop improvements. This conceptual model focuses on optimising current practices by tuning the development process to the product quality requirements.

The process that is being tuned only consists of those process actions that the specific organisation is familiar with. In other words: the conceptual model of chapter 5 only uses those capabilities that the organisation possesses, but overlooks the expansions with new capabilities. New capabilities, such as new ways of working, new techniques, methods, or tools, need to be considered for double loop improvement. Double loop improvement focuses on changes to the available capabilities. Expanding capabilities, however, is a difficult process. The conceptual model of Figure 6-2 should therefore be expanded with a double loop process that helps in selecting the right capabilities. This process should support in identifying and introducing those capabilities that are mostly needed by the organisation.

Once it has been decided to expand the current set of capabilities with a new capability, this should be introduced. Introduction of a new capability requires a learning process that:

- teaches the organisation the skills for applying that capability
- teaches the impact that the capability has on product quality.

This last point is important. For each new capability it is necessary to learn what the impact is on product quality. Even though experiences from other organisations might indicate certain effects, it is not guaranteed that these effects also occur in the own organisation. It is therefore necessary to guide the introduction of a new capability with measurements to evaluate its effects, and develop the PPR model accordingly. The knowledge on effectiveness of new capabilities can then be used by process engineering for designing the process models for other products.

6.2.2 The expanded conceptual model

The expansion of the conceptual model with a double loop improvement process is visualised in Figure 6-3.

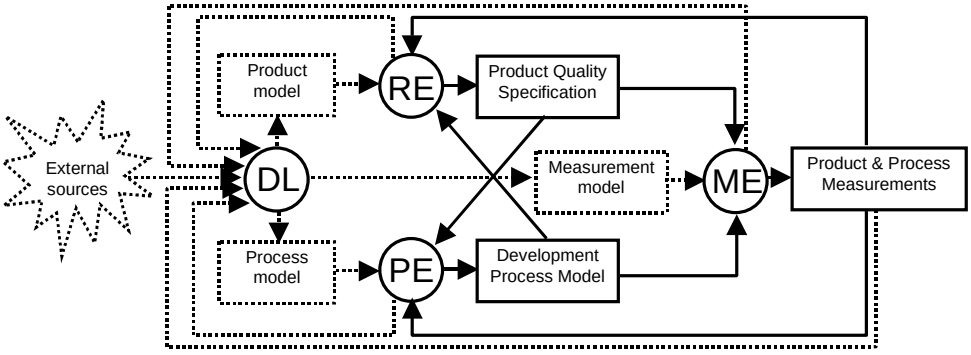


Figure 6-3: The conceptual RPM model with double loop improvement

In Figure 6-3 the conceptual model of chapter 5 is shown expanded with a double loop improvement decision (DL) and the corresponding processes (the dashed lines). Each of

the three working areas: requirements engineering, process engineering and measurement programme engineering, use an underlying set of assumptions. In this set of assumptions the ‘best practices’ of that organisation are ‘stored’. Experiences during single loop learning can indicate that a rigorous change of working is required when these underlying assumptions appear to be no longer correct. These assumptions are visualised with a product model, a process model, and a measurement model.

The product model contains assumptions on product quality that are normally used during requirements engineering. An example of such a product assumption is that product reliability and functionality are the most critical product quality attributes. When developing an embedded product for users with no experience at all with technology there might be a need to refocus to usability.

The process model contains assumptions on process actions and these are used during process engineering. An example of such a process model is that testing is an important contributor to product quality. When quality requirements increase, the time required for testing is increased; however, during testing it is often difficult to correct major quality problems. The introduction of software inspections is a more likely, double loop, improvement.

The measurement model contains assumptions on measurements and these are used during measurement programme engineering. An example is the use of the Mean Time Between Failure (MTBF) for product reliability. When experiences indicate that ‘reliability’ is much broader than defects, and involves soft issues such as ‘trust’ and ‘confidence’ the measurement of reliability is likely to change largely.

Additional capabilities that will be adopted, read: double loop improvements, are selected based on their impact on product quality. It is therefore important to know which product quality attributes normally give many problems, or that need improvement for the future. Such information comes from single loop learning processes. Information on process actions that are not successful are also a good source of information to select additional capabilities.

Rigorous changes during double loop improvement (DL) are selected. Input to this selection process comes from all three processes: requirements engineering, process engineering and measurement process engineering. Furthermore, external sources are also a source to identify potential improvement opportunities, since successful experiences from other organisations might point to additional capabilities that are also relevant for the own organisation. These external sources can indicate that new process actions have been developed that have a higher or better guaranteed impact on a product quality attribute than the ones currently used. For example, newly developed usability techniques become

available that replace the need for a field test to test the usability perception of the end-user. A set of improvement opportunities is defined based on these multiple sources.

The new double loop improvement process that was introduced in Figure 6-3, has made the necessary changes to the conceptual model of chapter 5, because it now distinguishes single loop and double loop improvement processes. Furthermore, it includes addressing explicit learning goals and incorporates an explicit learning process.

6.2.3 Operationalising the learning processes in the conceptual model

The conceptual model presented in Figure 6-3 contains a description of the processes, and work products for product focused SPI; however, this is still just a conceptual model. Using such a model in a practical environment still requires some work to be done. The operationalising of each of the processes in the model is done in chapter 7 by providing a set of guidelines for each of the three working areas.

Learning theory can again be of use to support this process. Operationalising the processes and improvement loops of the conceptual model, requires setting up procedures to carry out the required work. As discussed before, process improvement is a learning process. So, operationalising such a learning process can also be supported by learning theory. Learning theory contains several publications on successful learning processes. It describes process conditions that should be in place to enable learning in practice. These process conditions should also be considered in this thesis when operationalising the conceptual model.

Learning enablers are therefore discussed in the next section, and further details to operationalise these learning enablers for product focused SPI are provided.

6.3 Learning Enablers

The literature on learning theory presents multiple criteria for a successful learning process and organisation. As this chapter has the objective to explore learning theory as a support to increase learning effectiveness of product focused SPI, these criteria also need to be explored. These criteria taken from several sources, will be used to define a set of *learning enablers* that contain those process conditions that increase the learning effectiveness of product focused SPI. These learning enablers are elaborated in such a way that their goal is made clear, their meaning for the conceptual model is made clear, and the way in which they can be used for the improvement loops is also made clear.

Three sources that present learning process success criteria will be handled, in historical order. Firstly, Peter Senge introduces the five learning disciplines in his book '*The fifth discipline: The art and practice of the learning organisation*', [Senge 1990]:

- *'systems thinking'*, which means the use of a modelling viewpoint to analyse situations and clarify causes and effects. This is done by the identification of for example: processes, products, interrelationships, causal impacts, controls, etc.
- '*personal mastery'*, which means the drive of an individual continuously to improve herself and increase her skills and capabilities
- '*mental models'*, which means making the underlying deep implicit models explicit and evaluating whether they are correct or need adjustment.
- '*shared vision'*, which means the development of a common vision for the future that is accepted and adopted by the people of the organisation
- '*team learning'*, which means the process in which the learning processes of the individual members are combined, the learning results are shared, and the total learning result is more than the sum of the individual learning results.

These five 'disciplines' appear to be quite abstract. Systems theory is considered to be an important baseline for organisational learning. Furthermore personal learning, on an individual level as well as on a team level is emphasised. These five dimensions might seem simple; however, using these dimensions in practice, and applying its principles in actual organisations is not so easy [Senge et al. 1994]. Take for example the first dimension: systems thinking. Much theory and literature is behind such a topic [see e.g. Leeuw 1986 and Veld 1975]. Applying these principles in practice requires a lot of work and the application of principles that are not always fully elaborated.

With the ideas and concepts of Senge in mind, Garvin defined that learning organisations are skilled in [Garvin 1993]:

- *'systematic problem solving'*, which means relying on a scientific method, such as Plan/Do/Check/Act, insisting on measurement data, and use of (simple) statistical analysis
- '*experimentation with new approaches'*, which means the systematic searching for and testing of new knowledge on new ways of working
- '*learning from own experiences and past history'*, which means the review of successes and failures, systematic assessment, and recording of lessons learned in an easy to find and accessible way.
- '*learning form experience and best practices of others'*, which means to look for powerful insights and new perspectives outside the individuals direct environment
- '*transferring knowledge over the organisation quickly'*, which means spread knowledge quickly and efficiently throughout the organisation and making learning more than a local affair

Again, these five skills appear trivial; however, they are not that easy to apply in practice. For example, 'learning from past history' does not occur automatically, and it is not clear how this should be enabled and supported. As support a 'systematic assessment' is mentioned: But what is this? How do you do that? With which information? Who takes care of that? With what skills? etc. These questions remain unanswered and express the difficulty of creating a learning organisation.

Garvin also supports the systems theoretical perspective. The personal learning aspects are grouped over learning processes: learning from experiments, past experiences and external experiences. Furthermore, Garvin mentions the distribution of knowledge as an explicit process.

Ten facilitating factors for organisational learning are described by Nevis, based on previous work by e.g. Senge and Garvin, are [Nevis 1995]:

- *'scanning imperative'*, the gathering of knowledge about conditions and practices outside the direct environment.
- '*performance gap'*, establishing a shared perception of the gaps between actual and desired state of performance or situations
- '*concern for measurement'*, defining metrics, collecting and analysing data to create specific quantitative knowledge
- '*experimental mind-set'*, establishing an openness to trying new things and learn about them, failures are accepted without punishment
- '*climate of openness'*, establishing a context with free flow of information, open communication, sharing of problems and lessons learned, and open debate of ways to solve problems
- '*continuous education'*, an ongoing commitment to education on all levels of the organisation, both individual and team learning
- '*operational variety'*, the appreciation of diversity: variety of methods, procedures, and systems
- '*multiple advocates'*, new ideas and methods are supported at all levels
- '*involved leadership'*, leaders articulate vision and take part in the implementation of ideas, and are actively involved in the learning processes
- '*systems perspective'*, problems and solutions are seen in terms of systemic relationships among processes and products.

Nevis also addresses systems theory perspective explicitly. Still these dimensions remain quite abstract. It is for example still unclear how these facilitating factors should be implemented in practice, what management exactly has to do to facilitate learning along these directives, etc.

The above ten facilitating factors will be used in this chapter for abstracting recommendations to increase the learning effects of the RPM conceptual model. Not every facilitating factor is, however, equally useful for this purpose and not all these facilitating factors will therefore be addressed. A selection will be made from these facilitating factors that will be provided with more detail. The ones that will not be used because they are the basic starting points for the RPM conceptual model and already fully covered are: 'concern for measurement', 'experimental mind-set' and 'operational variety'. The fact that these factors are already included in the RPM conceptual model indicates that this model already addresses these learning enablers. From the factor 'continuous education' we will only address the team education issues, because this process is most relevant for the RPM conceptual model, so 'team learning' will be applied. Furthermore, the 'systems perspective' will be more detailed, because this appears to be one of the basics of learning, but is still quite abstract. From systems theory we will use the five criteria for control to cover the systems perspective [Leeuw 1986]:

- '*explicit goal definition*', targets are made explicit in an objective and measurable way
- '*modelling of the system under control*', the process that will be followed will be modelled
- '*information on context and current state*', the current status of a project, product and the organisation will be made explicit
- '*possibilities for control*', it is possible to take (corrective) actions by executing new or additional process actions to create certain effects
- '*sufficient resources*', the execution of the learning processes will never be limited due to a lack of resources

This last criterion will not be used further on, because having all necessary resources available is one of the main assumptions for the RPM conceptual model. Of course this will not be the case in practice, but this thesis will not investigate this allocation process.

6.4 Learning Disablers

Beside learning enablers there are also learning disablers: factors that negatively influence learning processes. If an organisation attempts to increase the learning effects of SPI by installing several learning enablers, but forgets the learning disablers the learning process is still not ideal.

Examples of learning disablers include: not providing feedback on product quality and having information in this feedback loop that is not correct [Brombacher et al. 1995], using product and process quality information to evaluate people [Goodman 1993] [Solingen et al. 1997], having a long period between information gathering and information feedback [Grady 1992], and ad-hoc 'fire fighting' management instead of

looking at the underlying causes of problems and learning how to tackle these problems [Garvin 1993] [Senge 1990] [Agarwal et al. 1997] [Humphrey 1989].

This thesis does not have the objective to provide a complete list of the learning disablers that an organisation has to abandon, nor does it present how to abandon those disablers. The statement should, however, be made that in addition to focusing on learning enablers, it should no be forgotten to get rid of the learning disablers.

6.5 Learning within the RPM working areas

In this section it will be analysed for each of the learning enablers what these learning enablers imply for learning in each working area.

6.5.1 Interpreting the learning enablers

The learning enablers that were presented in the previous section will be used to identify how the learning effects of the RPM conceptual model can be increased. The learning enablers are analysed and an attempt is made to operationalise them for the RPM approach.

For each enabler a description will be provided of what the enabler means within the context of the RPM conceptual model, and the purpose will be made explicit. Furthermore, each enabler will be operationalised in such a way that is clarified what this enabler means for the RPM model.

Enabler 1: Climate of openness

A climate of openness addresses the establishment of an environment in which free flow of information, open communication, sharing problems and lessons learned, and open debate of ways to solve problems, is available. Such a climate or 'learning culture' seems a simple concept, but is difficult to establish in practice. Research has indicated that current structures for control and management in organisations tend to disable such climates of openness and with that decrease the commitment of their people [Amabile 1998] [Ulrich 1998]. The intrinsic motivation of people especially is crucial for establishing a creative and learning oriented environment. Practical actions that managers can take to increase the intrinsic motivation of people are grouped in six categories [Amabile 1998]:

- '*Challenge'*, by matching the right people with the right job assignments in such a way that employees definitely do not feel bored, but neither are overwhelmed or threatened by a loss of control.
- '*Freedom'*, by giving people autonomy concerning the processes they apply. Management needs to set the goals, preferably as clear as possible, but the way to achieve these goals should be left to the people themselves.

- '*Resources'*, by carefully allocating time and money. Time pressure can increase motivation providing that deadlines are for real and not too tight. Money should be assigned properly to prevent people trying to find additional money themselves instead of doing their work.
- '*Work-group features'*, by carefully designing teams that are diverse, excited about on the goal, willing to support team-mates through difficult periods, and where each member contributes a unique significant amount of knowledge.
- '*Supervisory encouragement'*, by praising creating efforts spent by their people. Appraisals are not considered to be effective when they are given in extrinsic rewards such as financial bonuses. Freely and generously recognising creative work by employees already encourages largely. Managers should not be sceptical towards new and rigorous ideas.
- '*Organisational support'*, by establishing sufficient organisational support for the people in the organisation. This organisational support should enable learning efforts and support learning processes. Furthermore, the value of learning should be emphasised by the procedures and systems in the organisation.

A climate of openness appears to be one of the most crucial prerequisites for organisational learning. It requires a context in which people are willing to learn from their mistakes and willing to discuss underlying causes and models for these mistakes. For example, handling software failures as personal error disables learning largely, because software failures will always be a structural problem (see chapter 3). The way in which they are inserted and what to do to prevent them is an important learning topic, which will not be explored if software failures are considered to be personal failures of the software developers.

Enabler 2: Scanning for knowledge

In the broadest sense this means that there should be a continuous search for knowledge that could be relevant or applicable in the specific learning situation.

Scanning for knowledge from previous products, competitor products, similar products, or new methods is an important input to requirements engineering. The main point is that this loop does not build product quality requirements every time from scratch but attempts to learn from previous experiences. Furthermore, knowledge can be collected from previous projects that created similar products. Carrying out post-mortem analysis to find out whether a certain used process model was sufficient, is a good source of knowledge to increase the learning effects. Double-loop learning also requires scanning for knowledge. External knowledge on process actions that the organisational has no experience with should be collected and analysed. Specific process actions that other organisations apply and that create specific product qualities are of interest. Reading publications on achievements in software engineering by the software developers, is a way to scan

knowledge. Experiences from other organisation are in that way fed into the own organisation. Sending people to conferences, seminars and training is also a solution. What, for example, can be done is to have a weekly 'work lunch' in which one employee presents the contents of a paper he or she has read recently and clarifies the way the contents can be used within or influences the own organisation.

Enabler 3: Information on context and current state of the system

Learning adds knowledge to an existing situation and is influenced by the state of external influences. Information is needed on the context and current state to learn appropriately, and select the best suited additions.

The retrieval of information on the context and current state of product and process quality is an important part of the RPM approach. This learning enabler is already partially incorporated in the conceptual model; however, additional attention is required. Making processes explicit, measuring the performance of processes, or the current state of product quality is a useful source of information for this learning enabler. On the double loop level, typical structural problems or product quality attributes that require extra attention can be identified. Based on such insights, corrective action can be taken to change the current way of working to cope with such structural problems.

For example, carrying out software process assessments frequently supports in making current processes explicit. Knowing explicitly what the capabilities of an organisation are and making explicit which process actions they can use, contributes to this learning enabler. If for example, process assessments indicate that configuration management is a weakness in the organisation, projects that have high product maintainability targets will know that they need to take some specific action.

Enabler 4: Team learning

Team learning is an important part of an organisational learning process. It means that learning is established within groups of people that work together towards a shared vision and mutual objectives. Joint formulation of learning objectives, information sharing, discussion, and drawing conclusions together takes place within team learning.

Team learning can be used to find out a good way in which product quality requirements need to be specified to let the final product comply to them. Not every way of specifying product quality requirements will be equally effective. In this thesis, the multi-party chain approach [Kusters et al. 1997] has been used, but for different projects different approaches might be better. It is also important that development teams learn the behaviour of different development processes. A specific process action may not always give the same effect within different projects, for different products, with different team members. These differences and the causes for these differences need to be determined. Measurement is a powerful mechanism to enable this group learning, and discussing these

measurement results within a project team and challenging a team‘s interpretations, is a means to establish team learning.

Furthermore, teams should determine why certain product quality attributes are structurally problematic in projects. Based on such understanding, corrective process actions can be introduced and learnt by the teams to overcome such problems. It can also be determined which product quality requirements are always difficult to comply to, or difficult to specify. Corrective action can be taken once it is learnt what the causes for such issues are.

An example of team learning in one of the case-study companies has been published [Solingen and Berghout 1999]. In one of the development projects, a group of new engineers was hired for the software development work. These people were experienced in software development, but had no knowledge of the system that was developed in this department. As a solution the whole team decided to carry out ‘reviews’ of documents and software code to teach the new people the content of the system. As a consequence, the new engineers were trained remarkably fast; however, one of the side effects was considered to be an even more important benefit. This benefit was that due to the mutual reviews, the team started to structure and write their software in a more common manner. The team determined together what they considered to be ‘good’ coding and established this in their department, both the new and the already available engineers. This had never been done before and was largely contributed by the reviewing process.

Enabler 5: Modelling of the system under control

In order to control a system, a model needs to be created form this system and its influencing factors. In the RPM model this can be modelling of process actions, their relationship to the product quality requirements, and the effects of the process actions. Another example of modelling is the development of a development process model, or the modelling of user groups and their mutual relationships to support the identification of all stakeholders.

Examples from practice are that in projects explicit models are made from the process that is intended to be used, or models are made of the expected impacts of a certain process action. In one of the case-study companies for example, they introduced ‘incremental development’ by which the product was developed in three sequential increments, each expanding the previous one with specific functionality. The expectations of this change were modelled by making them explicit and identifying whether these expectations were legitimate. The measurements and observations showed indeed that these expectations were valid. The explicit model of this process-product relationship is now used in that organisation.

Enabler 6: Possibilities for control

In order to be able to steer a process towards the required outcomes, possibilities for control need to be available. This means that during an improvement programme (corrective) action can be taken whenever necessary. For example, when it appears that the intended reliability level can not be reached, it should be possible to take action to improve that situation. Such possibilities for control are a prerequisite for systems theory, and are a basic foundation of the RPM conceptual model.

The idea of the RPM approach is that a product specific development process is designed for each project separately. This process is designed during process engineering. Process engineering should therefore have a sufficient set of process actions available to choose from. From this set, a selection can be made that suits product quality specification. In a double-loop fashion, the available set of process actions should be expanded with new ones that suit the specific organisation. Additional process actions should therefore be found that provide support for those product quality attributes that need such expansions most.

Enabler 7: Involved leadership

Managers should articulate vision, take part in the implementation of ideas, and be actively involved in the learning processes. The role of a manager for the establishment of organisational learning, and motivating the people in the organisation is crucial [Senge 1990] [Garvin 1993] [Amabile 1998]. In a learning organisation managers and the role of the manager is changed largely compared to traditional management styles. The largest differences are that the manager is a designer of the learning organisation, a teacher of the view on reality, and a steward for the people they manage [Senge 1990]. According to Senge these special roles require the five special skills described in section 6.3.

Practical implementation of such a different management style is not always easy, because both the manager and the people that are managed are used to a different style. In an organisation where a manager always defines the procedures that are to be used, and the manager suddenly leaves the freedom for the process to his people, it is likely that people might abuse this freedom. Such a change in management style should therefore be carefully planned and a smooth transition should be established. In creative intellectual work, however, such management styles are often already present. Furthermore, the manager has a large influence on the working environment, which should be a ‘climate of openness’.

Enabler 8: Explicit goal definition

In order to have clear targets towards learning, these goals should be defined and made explicit. Learning processes benefit if it is clear what the goals are and in which area learning is required to attain such goals.

The whole RPM conceptual model is based on setting explicit goals for product development. Both product quality and process goals are explicitly stated. Product quality demands are made explicit and targets are set. For the process, measurement goals are set to monitor the performance of specific process actions, and the measurements are analysed explicitly to learn the effects of such a process action. For double-loop learning, explicit learning goals are defined to learn effects of process actions with which no experience exists. This learning purpose is already an explicit goal within the RPM conceptual model; however, this learning enabler confirms the importance of defining quality goals and learning goals. Formulating expectations (hypothesis) for the effects contributes to attaining these learning goals, because expectations can be compared to actual values and reasons for differences can be identified.

An example of the use of explicit goals for learning was the identification of re-use effects in one of the case-study organisations [Solingen and Berghout 1999]. The project team defined the explicit goal to measure the effects of software re-use on product reliability. Their expectation was that this contribution was high. The measurements showed indeed that the defect level of fully re-used modules was remarkably low. An important learning point from this project was the indirect effect of re-use on reliability. The project team learned that they were more strictly reviewing and testing re-used modules, because they did not 'trust' them as much as the one they had developed themselves. As a consequence these modules were much more reliable, because the reliability problems that did exist were already identified before release. The project team concluded that both direct and indirect effects of re-use largely influence product reliability. This learning point was packaged in a process-product relationship model and stored in the experience base.

Enabler 9: Monitoring performance gap

Monitoring the differences between target and actual situations is an important prerequisite for learning. It supports in identifying what is going well, and what needs improvement. Through this performance monitoring, people get feedback on their way of working and learn where to improve. Measurement programme engineering is an important contributor to this learning enabler in the RPM conceptual model.

The performance gaps between stated product quality requirements and the actual product quality is monitored in the RPM approach and this gives information on the predictability of the software development process. Specifying product quality requirements in measurable terms makes the monitoring of these gaps even easier. Monitoring performance gap is not only done for the product, but also for the process in the RPM conceptual model. The performance of process actions is monitored, and if differences exist between expected and real effects of process actions, corrective action can be taken. In fact, one of the main purposes of measurement programme engineering is to monitor the performance gaps on the product and the process.

For example, in one of the case-study companies a structural problem was present. Due to the large number of countries being supplied and the large differences in government regulations across the countries, it was difficult to address all country specific requirements. This caused many ‘change requests’ after products were released to the national representatives in the countries. The performance gaps between wanted and actual product quality could be made explicit for this organisation, and corrective action was defined. One example solution was to develop the country specific requirements in close co-operation with the national representatives and use these requirements as input to the product architecture design. As a result, a product architecture was designed that was capable of attaching product specific software customisations after release of the product.

6.6 Conclusion

In this chapter learning theory was explored to find relevant recommendations for product focused SPI. Based on this investigation, the RPM conceptual model was enhanced with a double loop learning process.

Furthermore, a set of learning enablers was identified that can increase the learning effects of the RPM conceptual model when addressed properly. These enablers were further analysed and operationalised for the learning processes in the RPM conceptual model.

Using these learning enablers from this chapter and the findings from software engineering literature presented in chapter 5, practical guidelines can be designed for operationalising the RPM conceptual model in order to make it applicable in industry. This will be done in the next chapter.

7. Guidelines for product focused process improvement

A conceptual model for product focused SPI was constructed in chapter 5 and enhanced according to learning theories in chapter 6. The model contains three working areas for product focused SPI: requirements engineering, process engineering and measurement programme engineering.

Additional support is required to facilitate the usage of this model, and to integrate its application into practice. Such support is described in this chapter by a set of guidelines for each of the three working areas. These guidelines originate from multiple sources: literature on the working areas, literature on learning theory, and experience from applying the conceptual model in industrial projects.

The guidelines in this chapter mainly describe 'what' can be done to support product focused SPI in the three working areas. As such, these guidelines point to the tasks and activities that are part of each of these working areas. These guidelines do not present 'how' this should be done, because it is assumed that 'how-guidelines' largely depend on the specific context. It is impossible to describe non-situated 'how-guidelines'; however, the case-studies and case-study procedures in chapter 8 can work as an example how to apply these guidelines in industrial projects.

7.1 Guidelines for Requirements Engineering

Requirements engineering was defined in chapter 5 as the process of collecting wishes of product stakeholders and transforming these wishes into a product quality specification. The product quality specification is used for two purposes:

- to design a development process that will produce the specified product quality within the constraints of the development project
- to evaluate compliance of the final product to the product quality requirements

The guidelines will be structured along the three mechanisms for standardisation presented in chapter 3 [Mintzberg 1983]: inputs, work processes and outputs. Firstly the guidelines regarding the inputs of requirements engineering will be presented, followed by the guidelines regarding the process of requirements engineering, and finalising with the guidelines on the outputs. This is depicted in Figure 7-1.

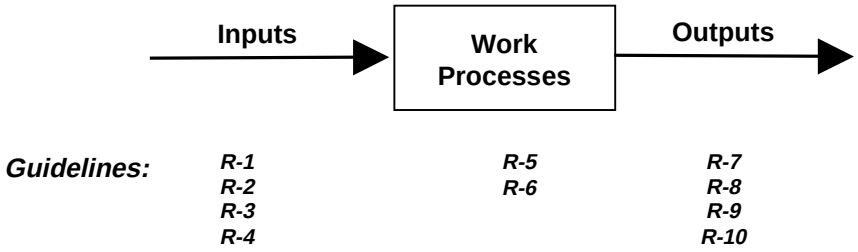


Figure 7-1: Guidelines on inputs, work processes and outputs of requirements engineering

Figure 7-1 contains the subdivision of the guidelines for requirements engineering over inputs, process and outputs. These guidelines are:

R-1. Identify all stakeholders for the product, and involve each stakeholder in the requirements engineering process.

R-2. Let stakeholders state their product quality wishes in their 'own terminology', and transfer those wishes into (standard) engineering quality terminology.

R-3. Use experience with a similar type of product or older version that already exists, as input to the creation of a product quality specification.

R-4. Make a distinction between essential, stringent and additional wishes.

R-5. Requirements engineering should be considered to be a negotiation process during which decisions are made on the level of satisfying product quality wishes. This negotiation process should discuss both functional and non-functional product wishes.

R-6. Communicate rejection or selection of a product quality wish to the stakeholders.

R-7. Handle the abstract concept of product quality by subdividing quality into operational attributes.

R-8. Specify the (relative) importance of product quality attributes for a new product, and visualise this in a Product Quality Profile (PQP).

R-9. Specify product quality requirements in measurable terms.

R-10. Show the trade-off between quality demands and the incurring cost/effort to realise these demands.

These guidelines will be described in detail in the next section. Some terminology confusion might occur; therefore Figure 7-2 clarifies and visualises the terms used.

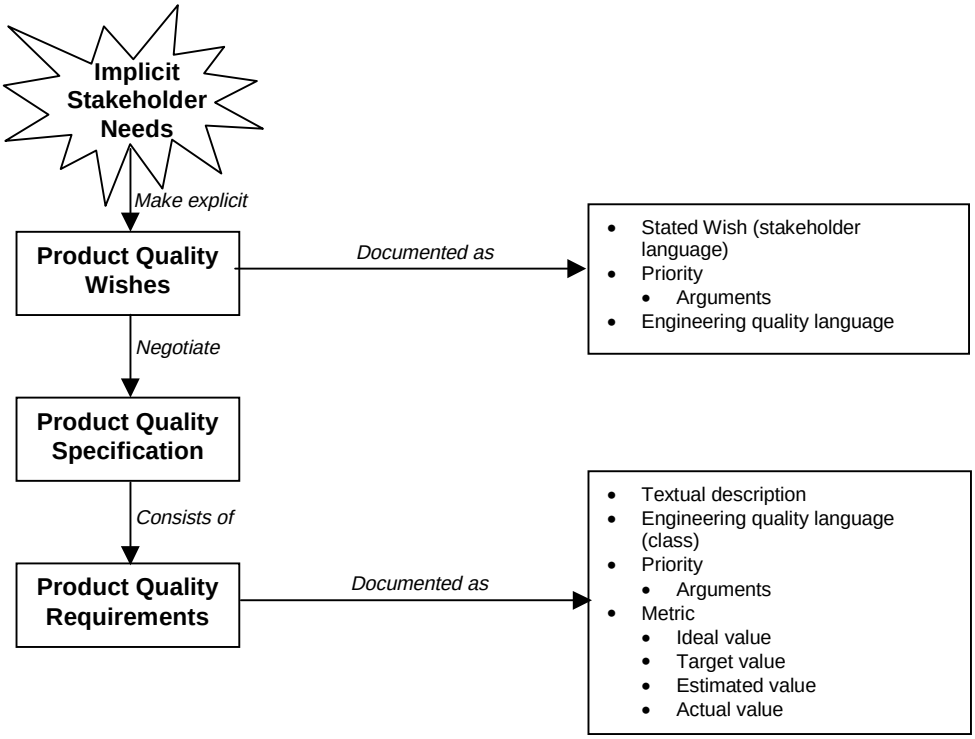


Figure 7-2: Terminology clarification

Stakeholders have, often implicitly, various wishes and demands for product quality. These needs have to be made explicit and expressed in 'product quality wishes'. A wish is preferably expressed in the language of the stakeholder, will get a priority and has arguments for this priority, after which all wishes are expressed in an engineering quality language. Not all wishes can be fulfilled: some are accepted, others rejected, wishes are reformulated, or tuned to the possibilities within certain conditions. The total set of product quality wishes that are intended to be fulfilled are documented in the 'product quality specification'. This is the main deliverable of requirements engineering and contains the complete set of 'product quality requirements'. Each product quality requirement contains an exact specification. This specification consists of a textual description of the product quality requirement, the product quality class which it addresses, a priority (and arguments) and metrics. These metrics are used to identify how attainment of the product quality requirement can be tested and validated. Possible values that are stored for these metrics are the target value, goal within the project, maximum value, if all wishes of all stakeholders are addressed, estimated value, what is expected, and the actual value, measured once it is possible.

7.1.1 Guidelines on Requirements Engineering inputs

R-1: Identify all stakeholders for the product, and involve each stakeholder in the requirements engineering process

Each product goes through several stages as it is designed, produced, transported to the customer, installed, used, repaired and recycled, etc. For each of these stages in the product life-cycle, 'users' can be distinguished that use the product; however, the way of use and the related quality needs for this use, will be different per user. In chapter 3, it has already been stated that product quality is multi-dimensional: it means different things to different people. Within this research a modelling technique has been created, which distinguishes product users as being 'stakeholders', while every stakeholder possesses one or more responsibilities (roles) [Kusters et al. 1997] [Kusters et al. 1999]. For each product a model can be made that identifies the stakeholders for a product and their interrelationships. This guideline is based on the assumption that making a quality product implies addressing the specific needs of specific stakeholders, their specific roles and their specific personal interests.

A stakeholder is defined as *an identifiable person, or homogeneous group of persons that has a legitimate interest in the degree of quality of the product.* A role is defined as *an area of responsibility of a stakeholder, determining a view on the type and degree of quality required* [Kusters et al. 1997]. Interrelationships between stakeholders and roles are especially interesting, because they reflect the mutual dependencies and therefore the related requirements.

The guideline to identify all stakeholders and their roles might seem easy; however, this is quite complex, because the number of stakeholders and their roles expands fast. Take for example the well-known ITIL method for information systems maintenance [ITIL 1987]. ITIL describes several stakeholders, including a helpdesk, problem management, change management, application management, configuration management, financial management and capacity management. Each of these stakeholders consists of several roles, for example the helpdesk has operator roles, management roles, service roles, etc. Identifying stakeholders and roles is a difficult task, and it is much more difficult to make sure that the specific demands from these stakeholders and roles are sufficiently addressed.

Ideally each stakeholder is involved in the process of requirements engineering. Their demands and wishes regarding a product can then be captured. The product quality wishes from a stakeholder are related to the role of that stakeholder. The way in which the stakeholders are consulted can be different, depending on the best way to capture the knowledge of each stakeholder. It will often be infeasible to consult all roles and stakeholders directly, because this could mean that, in some cases, hundreds of people should be consulted. A practical alternative to this is to consult key-people in the organisation that have much knowledge and experience, and that are able to represent

other stakeholders in their specific roles. For example, a marketing manager that previously worked as a service department manager can be used as a representative of the service and customer stakeholder, due to his past experiences.

R-2: Let stakeholders state their product quality wishes in their 'own terminology', and transfer those wishes into (standard) engineering quality terminology

Stakeholders have implicit ideas and needs for product quality. In order to prevent formulation problems it is recommended that stakeholders express their product quality wishes in their own language. This has several benefits: firstly, the stakeholder can express implicit needs more easily, making his explicit wishes more reliable. Secondly, many stakeholders have no experience with a standard quality terminology, nor are they always willing to learn it. Furthermore, it prevents that each stakeholder has its own interpretation of such a standard terminology and mistakes are prevented; however, when stakeholders state their wishes in their own terms, these wishes have to be transferred to engineering quality terminology. During this translation step, wishes can be expressed in a standard quality language, such as for example ISO 9126 [ISO 9126 1991].

This step is not only a translation step. In many cases it will be mainly a classification step, in which stakeholder wishes are classified over product quality attributes. On the other hand, there will be cases in which the wish expressed by the stakeholder should be translated to a language that can be understood by developers. Specifying quality wishes in such a standard quality terminology will also help in communicating issues on product quality to all stakeholders and will support in the reuse of the product quality wishes for future products.

R-3: Use experience with a similar type of product or older version that already exists, as input for the creation of a product quality specification

If an older version of the product or a similar type of product is already used in practice, experience with this product can be a valuable reference for requirements engineering ('anchoring and adjustment': [Davis 1982]). This guideline resembles the concept of 'product families' (see e.g. [Erens 1996]), which is based on the notion that next generations of products have a close resemblance and are based on previous generations.

Such experiences can lead to expressions such as: 'reliability should be equal to the previous version', 'usability needs to be higher' and 'the functionality was fair but needs some specific expansions'. Such references make clear for developers, what needs to be focused on. It provides them with information on where 'they did well' and where 'they need to make things better'. Experiences with older versions of a product are also an excellent source to find out the way in which a stakeholder uses the product.

R-4: Make a distinction between essential, stringent and additional wishes

Not every product quality wish is equally strong. It is recommended to make a distinction between:

- essential wishes that must be addressed by the product. Without addressing these strong demands the product will be useless
- stringent wishes, for which it is highly recommended that they are addressed; however, under certain conditions it is possible to ignore such wishes
- additional wishes that are not essential nor stringent, but it can be beneficial if they are addressed

This distinction supports the requirements engineering, because it indicates the level of negotiation that is possible for each wish. To support in the selection of stringent and additional wishes, it is recommended that their relative importance is made explicit through assigning priorities and noting the arguments for these priorities.

7.1.2 Guidelines on Requirements Engineering processes

R-5: Requirements engineering should be considered as a negotiation process during which decisions are made on the level of satisfying product quality wishes. This negotiation process should discuss both functional and non-functional product wishes

Based on the total set of stakeholder product quality wishes, a selection will be made from this set. The decision to which extent a certain product quality wish will be satisfied is a complex negotiation process. Criteria that play a role in the acceptance or rejection of a wish are: costs, benefits, technological feasibility, efforts involved, time-to-market, level of contradiction with other wishes, or risks. It is advised not to limit this negotiation only to the quality aspects of a product. The functional demands also need to be discussed, because functional and non-functional wishes are related. This negotiation process also addresses investment issues, because a decision is made where to invest resources for the product. These resources can be used for both functional and non-functional properties of a product, and should therefore be discussed at the same time.

This negotiation process is not part of requirements engineering alone. It is done in iteration with process engineering, because process engineering provides insights on the costs and time issues for each specific product quality requirement. This negotiation process does not end. When additional requirements are formulated during the project, which is the case for almost every software development project, again a trade-off and negotiation will be made. Ideally a product quality specification is made once and never changed, but this is rarely the case in practice.

In the case where wishes are rejected for a valid reason (too expensive, not feasible, etc.), this can have serious consequences for some stakeholders. These stakeholders deserve to

get the opportunity to restate the wish, to make it for example less strict, or to have the opportunity to repeat that a specific wish has a high priority. This already points to the next guideline on communication of the outcomes of the negotiation.

R-6: Communicate rejection or selection of a product quality wish to the stakeholders

Given that there is a process during which all product quality wishes are evaluated and a decision is taken to accept or reject a wish, the outcome of this decision process must be communicated. The main reason for this is that the approach presented in this thesis addresses product quality explicitly. The decisions taken should therefore also be made explicit and communicated to the people involved.

Furthermore, this communication is necessary to manage the expectations of stakeholders. Stakeholders implicitly expect that wishes be fulfilled, when stated. This creates high expectations. Once the product is delivered and does not comply to those wishes, stakeholders will be disappointed and perceive the product as being of low quality. If the decisions on the level of satisfying a certain product quality wish are communicated, a stakeholder has the opportunity to adapt expectations, early on in the product development process.

7.1.3 Guidelines on Requirements Engineering outputs

R-7: Handle the abstract concept of product quality by subdividing quality into operational attributes

Quality has many dimensions. Examples are: maintainability, usability, reliability, etc. These dimensions are termed ‘quality attributes’ when considering a product. In order to make the abstract concept of quality more operational, it should be specified in operational quality attributes. Even though these attributes might also have multiple meanings, they are at least more concrete than the general term ‘quality’. Several product quality standards exist (chapter 3), which mostly use a hierarchical structure in which quality is refined to attributes, each attribute to sub-attributes, etc., sometimes going as far as detailed metrics for each attribute.

For software product quality, this thesis proposes to apply the ISO 9126 standard for the division of product quality into the attributes: Functionality, Reliability, Usability, Efficiency, Maintainability and Portability [ISO 9126 1998]. ISO 9126 calls these attributes ‘characteristics’, and refines them into sub-characteristics followed by several metrics for each sub-characteristic.

R-8: Specify the (relative) importance of product quality attributes for a new product, and visualise this in a Product Quality Profile (PQP)

If product quality is specified by means of product quality attributes, it is recommended to visualise this in a Product Quality Profile (PQP). A PQP visualises the product quality along the product quality attributes.

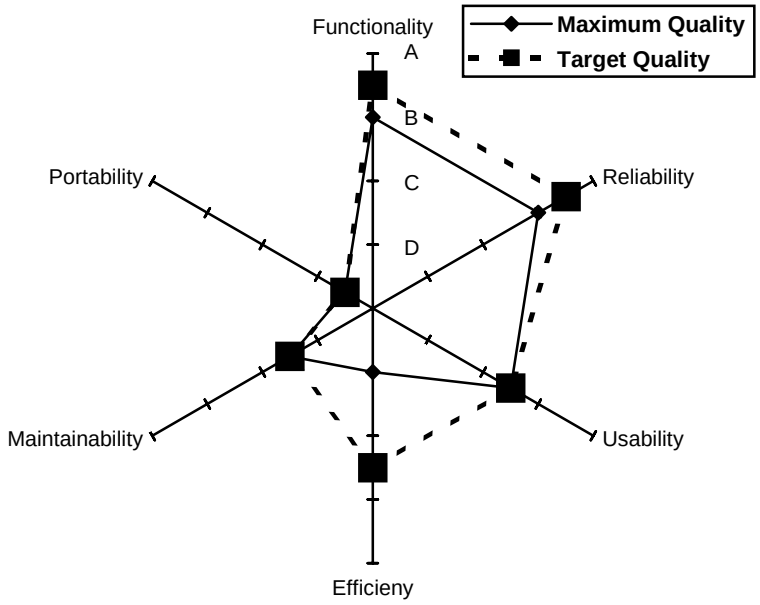


Figure 7-3: Example Product Quality Profile (PQP)

In Figure 7-3 an example of a PQP is presented. Two lines are included along the six product quality characteristics:

- maximum product quality, which express product quality if complied to all wishes stated by all stakeholders
- target product quality, which express the product quality if complied to all selected wishes that are included in the product quality specification

The sum of all quality wishes belonging to a specific class indicates the maximum level of quality. If the product complies to all wishes it is experienced as high level quality by all stakeholders. The sub-set of wishes that is selected based on the trade-off with other conditions is specified in the product quality specification. In the chart of Figure 7-3 an example profile is shown in which the maximum quality and the target quality are visualised. Such a chart can be expanded by adding lines, such as the actual product quality once developed, or the estimated quality when using a specific process. Such charts

do not contain all underlying details, but provide an overview of product quality and can therefore be used to communicate and discuss quality issues [Gillies 1992].

The scale of Figure 7-3 is the A to D scale, introduced in chapter 3. This thesis does not address how such scales need to be designed. Some ideas have been published [Uijtregt 1998]. In this example each attribute uses the same scale, it is however also possible to use a different scale per attribute including for example: costs in dollars, time in person months, reliability in MTBF and usability on a high, medium, low scale.

Priorities are not included in Figure 7-3. It is visualised which targets are set, but it is not indicated whether it is more important to fulfil the reliability targets or the maintainability targets. Setting priorities is done during the negotiation process, which is carried out in iteration with process engineering.

R-9: Specify product quality requirements in measurable terms

Ideally, product quality targets are specified as objectively as possible; therefore it is recommended that product quality needs are specified in measurable terms [Basili and Rombach 1988] [Gilb 1994]. By specifying requirements measurably, explicit goals become available, and the performance gap between target and actual quality can be monitored. This creates the benefit that the way in which these targets are fulfilled is left open, creating possibilities for variation and control by the developers. After all, requirements engineering specifies ‘what’ to build, and not ‘how’.

R-10: Show the trade-off between quality demands and the incurring cost/efforts to realise those demands

The results of the trade-off of quality to cost and effort should be made explicit and communicated. The highest level of quality is often not the objective for embedded products. High quality costs money and often only a few customers require that high level and are therefore willing to pay for it. In chapter 3 it has already been concluded that quality should be viewed in a value-based way [Garvin 1984], by balancing it against other conditions such as cost, effort and time-to-market. This balancing should be done explicitly and the relationships between quality and cost should be made clear.

Making such a trade-off means investigating, for each wish, how it is possible to address it, and what it costs. This involves looking at the process and looking at the available resources that are capable of addressing each wish. Furthermore, addressing certain wishes may have impacts on time-to-market i.e. development duration. Balancing wishes to their incurring costs involves an iterative process between requirements engineering and process engineering.

7.2 Guidelines for Process Engineering

Process engineering is defined in chapter 5 as the design of a measurable development process for the development of a specific product that complies to the product quality specification. A development process consists of a set of process actions with explicit expected effects on product quality. During process engineering a set of process actions is selected that contribute to the required product quality. These process actions are then assembled into a product specific development process model. A process action is an action, taken to achieve an explicit expected effect. Process engineering can also be seen as configuring and tuning a situated software development process, based on the product quality specification.

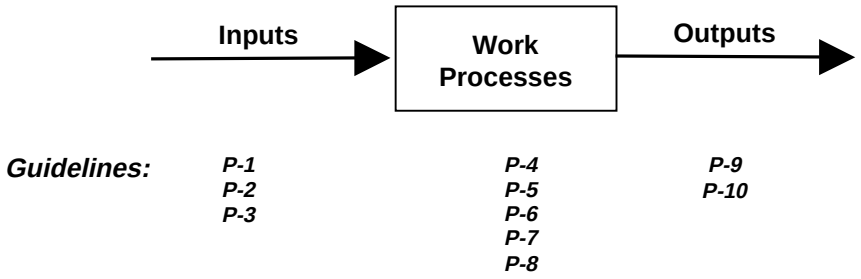


Figure 7-4: Guidelines on inputs, work processes and outputs of process engineering

Figure 7-4 contains the division of process engineering over inputs, process and outputs.

P-1. Only start with process engineering if the stakeholders wishes for product quality are made explicit.

P-2. Make the set of essential process actions and the set of supplementary process actions explicit.

P-3. File the expected effect of process actions on product quality in an experience base.

P-4. Develop, for each specific product, a separate development process model that makes the set of process actions, taken to control product quality, explicit.

P-5. Consider that effects of a specific process action can be both positive and negative, and that they can be different for different projects, because context factors vary.

P-6. Estimate whether the selected set of process actions is capable of complying to the product quality targets.

P-7. Use the information in the experience base for the selection of process actions.

P-8. Innovate by introducing new process actions with which no experience exist, in order to improve and learn.

P-9. Make the learning objectives for the application of certain process actions explicit.

P-10. Revise the development process model when significant changes occur.

These guidelines are described in detail in the next sections.

7.2.1 Guidelines on Process Engineering inputs

P-1: Only start with process engineering if the stakeholders wishes for product quality are made explicit

The main objective of process engineering is to configure a product specific development process. This can only be done if the implicit needs from the stakeholders are made explicit: process engineering depends on the results of requirements engineering. The exact product quality targets are, however, set in iteration with requirements engineering because deciding what the targets will be is always a trade-off with aspects such as feasibility, time-to-market, effort, costs, etc. Process engineering enables estimating what product quality will be when using a certain process within certain conditions. In iteration with requirements engineering, a development process is designed and the product quality targets are set.

During requirements engineering product quality is refined in relevant attributes, such as functionality, cost, reliability and maintainability. The wishes that exist for a product should be available before process engineering can start. Without an overview of product quality wishes, it is not clear what the specific demands regarding product quality are, and therefore it becomes impossible to implement product focused SPI. Without an explicit specification of product quality it is not possible to focus the improvement efforts to the product, because it is not possible to identify which product quality attribute(s) need improvement.

P-2: Make the set of essential process actions and the set of supplementary process actions explicit

Configuring a development process for a specific product is never started from scratch. Every organisation has its own ‘standard way’ of doing projects and its own ‘standard’ set of process actions. This set of essential process actions, always taken in development projects, must be made explicit. In addition to this it must be made explicit what the experiences (expectations) are of the impact of each process action on product quality.

Beside a set of essential process actions that are always taken, there is also a set of supplementary process actions that can be taken if the specific situation demands it. This supplementary set must also be made explicit, together with the experiences (estimates) of effects on product quality.

P-3: File the expected effect of process actions on product quality in an experience base

The essential and supplementary process actions that the people in the organisation are capable of using, and the effects of these process actions on product quality, should be modelled explicitly and stored in an experience base. This experience base is a dynamic and evolving storage medium in which new experiences and measurements with effects of

process actions can be stored, and adapted based on new insights. The experience base is also consulted to support in decision making during process engineering.

For the set up of such an experience base and its structure we refer to the literature [Basili and Rombach 1988] [Genuchten 1991] [Hamann et al. 1998a] [Hamann et al. 1998b]. A prototype was developed, and used in the case-studies of this research (see Appendix A).

7.2.2 Guidelines on Process Engineering work processes

P-4: Develop, a separate development process model for each specific product that makes the set of process actions, taken to control product quality, explicit

This thesis is based on the assumption that there is no one best way of making a quality product, because product quality depends on the specific needs of all stakeholders and the context in which it is being developed and used. In line with this assumption, every product requires its own development process to result in the specific product quality. This process needs to be made explicit.

Making process actions explicit needs to be done at several different moments of development. For example: during project planning when the intended set of process actions is defined, during project execution if certain process actions are omitted or additional ones are taken, and after project finalisation when it becomes more clear what the right set of process actions would have been.

Making these process actions explicit means identifying for each process action the time when it should be taken, in what way, using which technique, by whom, with what expected effect, etc. Most process actions have effects on more than one product quality attribute. For example the use of fault tolerance techniques has a positive effect on product reliability, but a negative effect on product efficiency and project duration. For embedded products it is recommended to make a distinction between overall process actions, hardware specific process actions and software specific process actions, and their intended effects.

Practical experiences identified that what this thesis addresses as ‘process engineering’ resembles what is done in practice during ‘project planning’, although implicitly. During project planning, project managers define a development process with deliverables, deadlines, resources, etc., that is intended to result in a product that fulfils the project targets; however, this process rarely addresses product quality explicitly. The customisation of the development process towards the project targets is mainly done implicitly; however, the point of time in which such a project plan is made, suits the point of time on which process engineering could be performed. It is therefore recommended to bring process engineering in line with the project planning work. Ideally they become one single task.

P-5: Consider that effects of a specific process action can be both positive and negative, and that they can be different for different projects, because context factors vary

Process actions that influence a specific goal in one area might decrease the effectiveness of the development process in another. This is often overlooked. It is recommended always to consider the multiple effects of process actions. Not only to focus on the product quality attribute that requires improvement, but also to consider the effects on the other product quality attributes. This ensures that an increase of product quality on one product quality attribute does not imply a decrease in another.

Although it might be likely that a certain process action will result in certain product quality, the effects of a process action on product quality can almost never be guaranteed. Multiple factors influence process actions, such as people experiences, project pressure, other process actions and current status of product quality; however, it can be made *likely* that certain effects will occur if the process action is taken in a similar project, under similar conditions, for a similar product, etc. The decision to take, or not take, a certain process action should be made for each project separately, dependent on the confidence (expectation) that the intended effects of a process action will occur.

In order to support this decision it is recommended to store in the experience base for every process action what effects were observed (measured) under which conditions. Examples of such conditions are: experience with the process action, product type, differences in process action application, type of technology, etc. These conditions that influence success or failure of process actions also vary per situation. It is therefore recommended to consult the development team for the identification of conditions in the project that had a clear influence on the effects of the process action [Hamann et al. 1998b] [Birk et al. 1999].

P-6: Estimate whether the selected set of process actions is capable of complying to the product quality targets

It is recommended to make an estimate of the product quality that a certain development process is likely to deliver. This estimated product quality can be compared with the product quality targets to identify whether the selected set of process actions is sufficient, or that corrective action should be taken. This corrective action can be changing the set of process actions, or changing the product quality targets, or both. From the experience base expected effects for this set of process actions can be retrieved, and an estimate can be made of what product quality will be created.

P-7: Use the information in the experience base for the selection of process actions

If a change has to be made to the already selected set of process actions, the experience base can be used to identify the process actions that have an effect on the specific product

quality attribute that needs additional attention. These changes can be twofold: additional process actions need to be selected, or already selected process actions need to be omitted. It should therefore be possible to search the experience base to find process actions that have a positive or negative impact on a specific product quality attribute. Furthermore, this experience base can be used to visualise which process actions have already been selected and which not, and what the impacts of each process action are on the other product quality attributes. It is for example possible that in a project additional attention is required for product reliability, while the project is exactly on track. Selecting process actions on the critical path that have a negative impact on project duration might in that case be not preferrable, because they will negatively influence the schedule of the project.

P-8: Innovate by introducing new process actions with which no experiences exist, in order to improve and learn

The experience base with models of process product relationships needs to be expanded with information on new process actions. Not every project will be suitable to experiment with innovative process actions on which hardly any knowledge is available. For example a product with high product performance requirements will not necessarily be favourable for experimenting with a new technique that focuses on product efficiency; however, experiments should be carried out to establish double loop learning (chapter 6). It is therefore recommended to assess the possibility for every project to experiment with new process actions, and to learn its effects on product quality. These process actions can for example be on a less critical product quality attribute, and used in a project with less critical deadlines.

7.2.3 Guidelines on Process Engineering outputs

P-9: Make the learning objectives for the application of certain process actions explicit

It has been identified in chapter 6 that learning should be a direct objective in process improvement programmes. For process engineering this implies that the learning objectives are stated explicitly. Learning objectives are defined for specific process actions. On these process actions the objective can be, for example, to identify what the effects are on product quality, what the conditions are under which these effects occur, or to monitor whether the intended effects occur and what the reasons for discrepancies are.

P-10: Revise the development process model when significant changes occur

The main product of process engineering is a development process model that indicates which process actions are taken, at what time, with what amount of effort, by whom, etc. This development process model is the main deliverable that ensures that the development process is capable of addressing the product quality targets. It should therefore be complete, correct and consistent with the work carried out in the development process.

This process model is not static: it evolves over time, based on changes in the project. Such changes can occur for a number of reasons such as: additional requirements are stated, expected effects of process actions do not occur, targets are set higher, or process actions are omitted due to time constraints.

As the development process model makes explicit 'how' the product is developed, it is essential to keep this model up to date during project execution. Especially since measurement programme engineering depends on this process model, it should be ensured that its content is correct and complete.

7.3 Guidelines for Measurement Programme Engineering

Measurement programme engineering is defined in chapter 5 as the design and implementation of a set of process, product and resource metrics, to evaluate product quality and to evaluate process-product relationships. During measurement programme engineering a set of metrics is defined, collected and analysed with two purposes:

- evaluating the compliance of embedded product quality with the stated product quality targets
- evaluating the effects of a certain process action on product quality, when used within a specific context in a specific way

Measurement programme engineering can also be seen as the process that provides feedback on the effectiveness of the process actions, and facilitates learning as was introduced in the conceptual model presented in chapter 6.

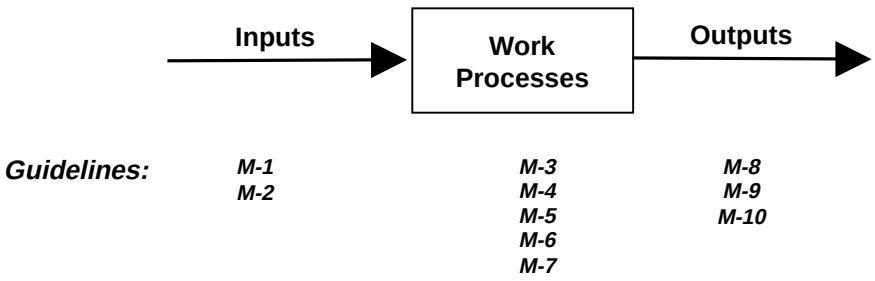


Figure 7-5: Guidelines on inputs, work processes and outputs of measurement programme engineering

The guidelines for measurement programme engineering are also structured along inputs, work processes and outputs (Figure 7-5). These guidelines are:

M-1. Prepare the software developers for participating in measurement programmes.
M-2. Know what the product quality targets, process model and learning objectives are before starting measurement programme engineering.
M-3. Measurement programme engineering should be goal-oriented to ensure that a limited but relevant set of measurements is collected.
M-4. Specify expectations (hypothesis).
M-5. Analyse and interpret measurement data regularly, which preferably needs to be done by those people that have performed the actual measurements.
M-6. Focus analysis and interpretation of the measurement data on: a specific process action, the overall process, or to the product quality targets, but not on the performance of individuals.
M-7. Assign dedicated resources to support the development team in measurement programme engineering.
M-8. Evaluate the differences between actual and target product quality.
M-9. Evaluate the effects of process actions.
M-10. Store the knowledge on the effects of a process action within a specific situation in the experience base.

These guidelines are described in detail in the next sections.

7.3.1 Guidelines on Measurement Programme Engineering inputs

M-1: Prepare the software developers for participating in measurement programmes

Measurement of software processes and products is not something that can be immediately done. Experiences in research identified that to carry out software measurement successfully the development context including the developers should be prepared [Basili and Rombach 1988] [Goodman 1993] [Fenton and Pfleeger 1996] [Solingen and Berghout 1999]. This is not only a finding from the software engineering field, but also a finding from learning literature which clearly emphasises the establishment of a 'learning environment' before organisational learning will occur [Senge 1990] [Garvin 1993] [Senge et al. 1994].

While both software engineering and learning literature recommend the preparation of an organisation and its people for measurement programmes, neither field states explicitly what this preparation consists of. The objective is to decrease the resistance to measurement and to establish a 'culture' that is willing to measure and improve itself; however, *how* this should be done is not made clear.

During application of these ideas in industry, this has been done by presenting the benefits of measurement in other projects or companies, and by making the exact impact of measurement for the developers explicit. The developers were told upfront what their involvement would look like, what they needed to prepare and carry out, what time it would cost and what benefit it would bring. Furthermore, it was arranged that only those issues would be measured that the developers supported, and that the developers were always in control of the work and direction of the improvement programme. If the development team decided that they had attained their goals and wanted to stop or refocus measurement, this was done, since they were the owners of the measurements. These actions were taken to prepare the organisation for measurement.

M-2: Know what the product quality targets, process model and learning objectives are, before starting measurement programme engineering

The role of measurement in this thesis during measurement programme engineering implies that the goals are available upfront. Measurement programme engineering starts whenever the measurement goals are defined. These goals are stated in the product quality targets, and learning objectives that result from both requirements and process engineering. With the product quality targets it is possible to measure during product development what the conformance of the actual product quality with the targets is. The learning objectives specify which process actions need to be measured and the purpose of these measurements. The process model specifies the process actions taken and sequence of activities in the project.

7.3.2 Guidelines on Measurement Programme Engineering work processes

M-3: Measurement programme engineering should be goal-oriented, to ensure that a limited but relevant set of measurements is collected

One of the findings in chapter 5 was that measurement should be driven by goals [Basili and Rombach 1988]. This has the benefit that measurements are only collected towards an explicit stated purpose and that only those measurements are taken that are necessary. This limits the costs (and burden) of measurement and supports in focusing measurement on only those process and product aspects of interest. Refining goals into metrics is a difficult process, which can be facilitated by specifying an intermediate level of ‘questions’, which at the same time provides a framework for interpretation of these measurements [Basili and Weiss 1984]. Operational support and guidelines for using goal-oriented measurement (GQM) is described in the literature [Solingen and Berghout 1999].

M-4: Specify expectations (hypothesis)

In order to increase the learning effects of measurement, it is necessary to make explicit what the expectations for the measurements are. These hypotheses need to be defined before the measurements are taken. Discrepancies between hypotheses and actual values trigger causal analysis of such differences and support in learning the effects of process actions [Entwistle 1981] [Basili and Rombach 1988] [Solingen et al. 1997]. Without the specification of hypotheses, learning effects tend to be much lower.

During the industrial work of this research, no hypotheses were defined in some cases, because the development teams had no idea what measurement results to expect. In these cases the development teams did not provide hypothesis. When the actual measurements became available they tended to conclude that these measurements where exactly according to their expectations: they had already forgotten that upfront they had no idea of the expected outcome. As such, the development teams learned less.

M-5: Analyse and interpret measurement data regularly, which preferably needs to be done by those people that have performed the actual measurements

In chapter 5 it was stated that measurement data should be interpreted in context [Basili and Rombach 1988]. This means that the analysis of measurements should be carried out by those people that have knowledge on the context in which the data was collected, and therefore ideally consists of those people that actually collected the data. In order to support the group learning aspect of measurement, chapter 6 found that analysing and interpretation of measurement data should be done in groups of people. A way in which this can be operationalised is organising so-called ‘feedback sessions’ in which the measurements are presented to the development team in order to draw conclusions about the measurements, make decisions, or take action [Latum et al. 1996] [Latum et al. 1998] [Solingen and Berghout 1999].

Interpretations of measurement data can be mostly twofold. If for example the number of defects found in a software module is relatively high, the conclusion can be that the module is of poor quality because it has many defects, but the opposite conclusion can be that the product is of excellent quality because nearly all defects are removed. In almost every case, measurement data can be interpreted differently. As developers tend to have a positivistic attitude and people have the general tendency to look for the first plausible interpretation, the opposite interpretation is often overlooked. It is therefore useful that someone takes the ‘devil’s advocate role’ during the interpretation of measurements and challenges and questions the various interpretations from opposite viewpoints. The discussions that result from such opposite statements support better interpretation and create deeper insights [Solingen et al. 1997].

M-6: Focus analysis and interpretation of the measurement data on: a specific process action, the overall process, or on the product quality targets, but not on the performance of individuals

Interpretation of measurements should also be done towards the measurement goals. If the intermediate level of 'questions' is used this interpretation is much easier, because the measurement data should provide 'answers' to these 'questions' [Solingen and Berghout 1999]. The measurements are always taken to support a learning process. As such, these measurements may not be used to judge people [Humphrey 1989] [Goodman 1993] [Solingen et al. 1997], because this will directly block the learning process of the people, and as a consequence the complete improvement programme might fail.

M-7: Assign dedicated resources to support the development team in measurement programme engineering

The development team will have both project and learning objectives. The project objectives are often much more concrete and attainment of these objectives is possible in shorter time. In the case of deadlines or project pressure, there is a risk that the learning objectives will be put on hold. To tackle this risk, it is recommended to limit the effort of the development team and involve them only in those elements of measurement programme engineering that add to the learning process. The non-learning tasks can be performed in parallel by other dedicated resources. These resources can be software developers that work on the product, but that have dedicated time to spend on the improvement programme. It is also possible to establish these dedicated resources in an external group, such as a quality assurance department; however, one of the findings of chapter 5 was to prevent the establishment of separate metrics groups that perform 'ivy tower' data analysis. These dedicated resources should therefore be facilitators of the improvement programme and not substitutes. They provide a service to the development team by taking over all tasks that do not necessarily require the involvement of the development team.

7.3.3 Guidelines on Measurement Programme Engineering outputs

M-8: Evaluate the differences between actual and target product quality

One of the reasons that measurement programme engineering is carried out is to evaluate the conformance of the (final) product with the product quality targets. The product is therefore measured and it is evaluated whether differences between the actual quality and target quality of the product exist.

If it is indicated during measurement programme engineering that there is a negative gap between actual and expected effects of the set of measures, corrective action should be taken. This corrective action can be:

- taking additional process actions to influence the specific product quality attributes positively
- altering the product quality targets to the current product quality

This last option should not be overlooked. Product quality is always a trade-off decision with time and cost. In many situations it might be acceptable not to spend time and money to make improvements when the product quality level is lower than the target. This depends on the market situation, competition, financial situation etc. The only recommendation is to make this decision an explicit one, and to analyse the consequences of not improving product quality towards the targets. This shows the relationship of measurement programme engineering with process engineering and requirements engineering.

It can also be possible that the evaluation shows that product quality is higher than intended. This may be a reason to start new negotiations or take specific action. Although unlikely, action can be taken to decrease the quality. A better option might be to consider increasing the product's price.

M-9: Evaluate the effects of process actions

The other purpose of measurement programme engineering within this thesis is to identify and learn the effects of process actions on product quality. Process actions are taken with an explicit purpose in mind; however, it can not always be guaranteed that these effects actually occur, because these effects depend on several (possibly unknown) conditions. In cases where there is a high dependency on the effectiveness of a process action it can be decided that measurements are required to monitor its results. These measurement results have to be analysed and compared with the expected results (hypothesis). In the case of discrepancies it is necessary to identify the causes (conditions) for these flaws. When it is clear that a certain process action does not give the intended effect, or produces unexpected side effects, corrective action can be taken.

As discussed before, it is also possible to monitor process actions with an explicit learning objective. This can be done for new process actions, or process actions for which it is still unclear what their detailed effects, or success-factors are. In such measurement programmes there is a clear learning objective and the focus is therefore not just on monitoring, but on reflection, conversation and conceptualisation (see chapter 6).

M-10: Store the knowledge on the effects of a process action within a specific situation in the experience base

When the goals of a measurement programme are attained, a lot has been learned. This knowledge must be stored in the experience base of process-product relationship models. Besides the effects of a certain process action on product quality, context information also needs to be stored in the experience base. Context information means for example: information on the specific situation in which the effects occurred, success factors and context conditions. Such information is necessary for future decision making, because it helps in making an estimation of the likelihood that a process action will give a certain effect, and therefore supports in the estimation of product quality during process engineering. If the type of product, development team and external conditions largely resemble a past situation in which a certain effect occurred, it is more likely that the same effects will occur then when all context factors are different. This context information is equally important as the information on the effects on product quality that have been measured.

7.4 Conclusions

In this chapter, a set of guidelines for each of the three working areas of the RPM conceptual model was presented. These guidelines focus on ‘what’ should be done and do not describe ‘how’ this should be done. The main contribution of this chapter was to operationalise the RPM model of chapter 6, and to provide support for practical usage of the RPM model in practice.

Validating the model and these guidelines still needs to be done. This is the topic of the next chapter that presents practical experience of using the RPM conceptual model and the guidelines in industrial case-studies.

8. Industrial application of the RPM approach

In this chapter the case-studies of this research are presented. The chapter contains experiences and results from applying the RPM conceptual model and the guidelines in four industrial projects. Furthermore it contains an analysis of the costs and benefits involved in applying the RPM approach.

8.1 Introduction

Case-studies are an important foundation for the methodological justification of this research, as presented in chapter 2. The case-studies are designed to validate both the RPM conceptual model and the guidelines.

In this chapter the companies and projects in which the RPM approach was used, is described. These projects were carried out over a period of four years and involved many people. It is out of the scope of this thesis to present here all the details from these case-studies. Details can be found in the individual papers that have been published on these projects [Solingen et al. 1999a] [Solingen et al. 1999b] [Oivo et al. 1999] [Bicego et al. 1999].

A section is provided for every case-study that briefly presents the results of requirements engineering, process engineering, and measurement programme engineering in that case-study. Examples are provided from the case-studies to support the findings and to illustrate what was actually done in practice when carrying out the RPM approach. Before these results of the case-studies are presented, the procedure used during the case-studies will be described.

8.2 Case-study procedure

The procedure that was used when applying the RPM conceptual model and the guidelines in the industrial case-studies, is presented in this section. The steps that were taken and the deliverables that were produced are presented. In this section, a reference to the guidelines of chapter 7 is presented in brackets. The case-study procedure consists of three parts: one for each of the three RPM working areas.

8.2.1 Requirements Engineering case-study procedure

The procedure applied for requirements engineering is depicted in Figure 8-1. The main deliverable of this procedure is a product quality specification. The negotiation process on product quality wishes and the feasibility to realise them within process engineering, and negotiating regarding costs and time-to-market is part of this procedure. Even though it was shown in chapter 5 that this is a negotiation process between requirements and process engineering, the case-studies carried out this negotiation process during requirements engineering. This also has the benefit that the switching between requirements and process engineering is not explicitly included in these procedures.

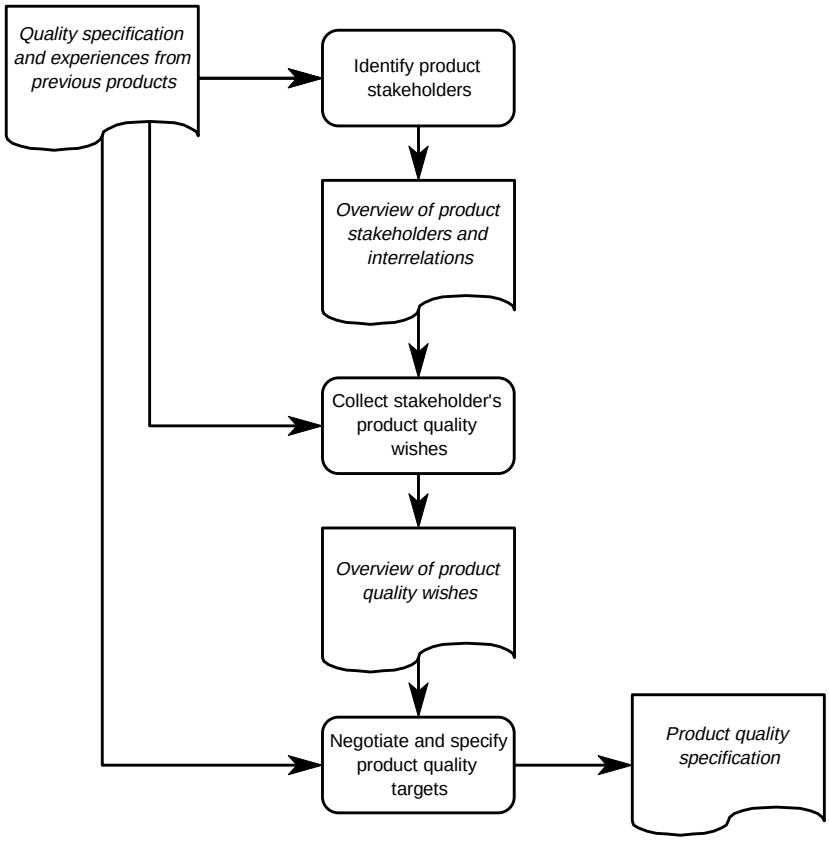


Figure 8-1: Requirements engineering procedure

In Figure 8-1, the process of requirements engineering is depicted. The input to requirements engineering is the quality specification of a previous product and experiences on quality with such previous products (R-3). First the stakeholders that have quality demands for a product are identified (R-1). Second, those stakeholders are consulted to collect the wishes they have for product quality. These wishes are specified in an overview

that describes the wishes in both the natural language of the stakeholder (R-2) and a generic engineering quality language (R-7). During the negotiation process (R-5) the product quality targets are set and the whole set of product quality requirements is documented in a product quality specification (R-8, R-9). This negotiation process looks at all wishes, their priorities (R-4), their cost, their feasibility and the time required to fulfil that wish (R-10). The output of requirements engineering is the product quality specification, which is an input to process engineering but should also be communicated to the stakeholders (R-6).

8.2.2 Process Engineering case-study procedure

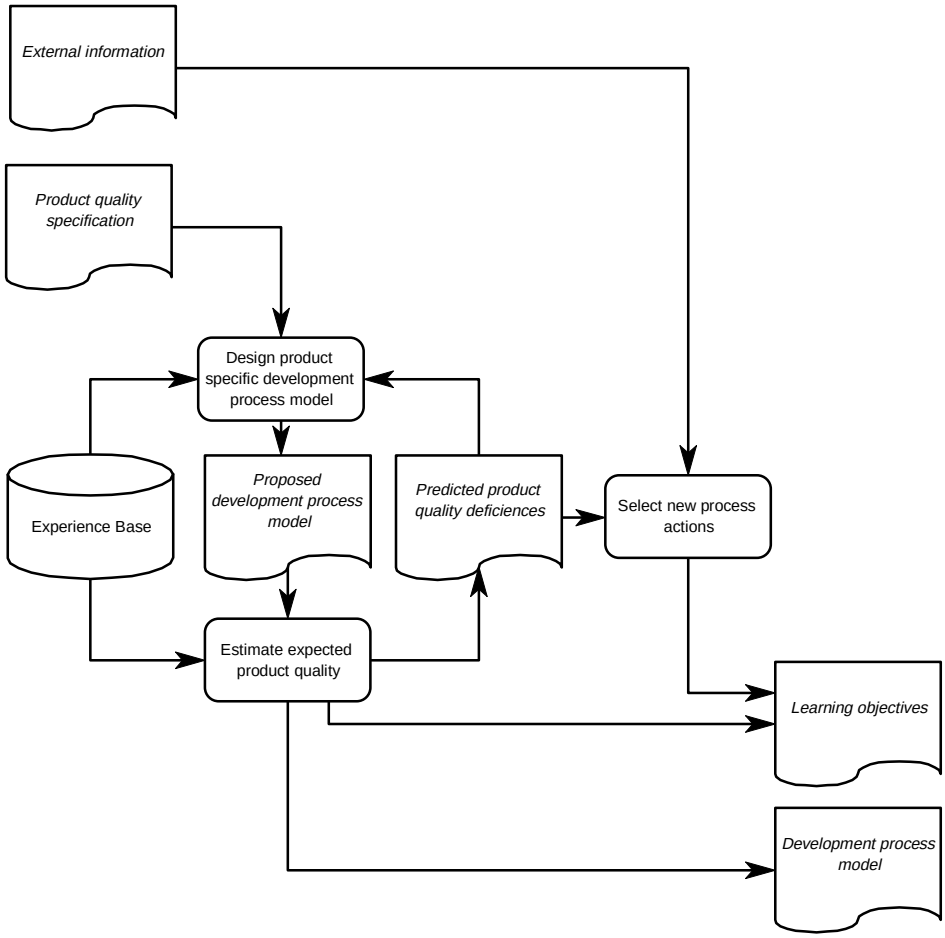


Figure 8-2: Process engineering procedure

The procedure applied for process engineering is depicted in Figure 8-2. The main deliverable of this procedure is a product specific development process model.

Furthermore, learning objectives are defined. The negotiation process on product quality wishes and the feasibility to realise them within process engineering, is considered to be part of requirements engineering. It is therefore not included in Figure 8-2.

Figure 8-2 has three input sources: the product quality specification from requirements engineering (P-1), an experience base on effects of process actions in the specific organisation (P-2, P-3), and external information on generic (effects of) process actions. Based on the product quality specification, a product specific development process is constructed (P-4) based on information from the experience base (P-7). Using the information in the experience base an estimation is carried out (P-6) of the product quality that can be expected if that process is used (P-5). When this expected product quality is compared to the product quality specification it is possible to identify deficiencies of product quality. In an iterative manner an (improved) development process is designed. The result from these iterations is a development process model that will be applied to develop the product (P-10). Furthermore, some learning objectives are defined that specify which process actions should be monitored, because their effects are uncertain (P-9).

In addition, process engineering contains the double-loop learning concept that identifies the need to select new process actions, or change the current way of working rigorously (P-8). This selection of new process actions is based on product quality deficiencies and external information. If a new process action is selected that will be introduced, learning objectives are also defined regarding the effects of this process action on product quality (P-9).

8.2.3 Measurement Programme Engineering case-study procedure

The procedure applied for measurement programme engineering is depicted in Figure 8-3. The main deliverables of this procedure are the measurement results. These measurements provide evaluation results on:

- actual product quality compared to the product quality specification
- process action effects compared to the expectations

Measurement programme engineering is depending on the outputs of both requirements and process engineering.

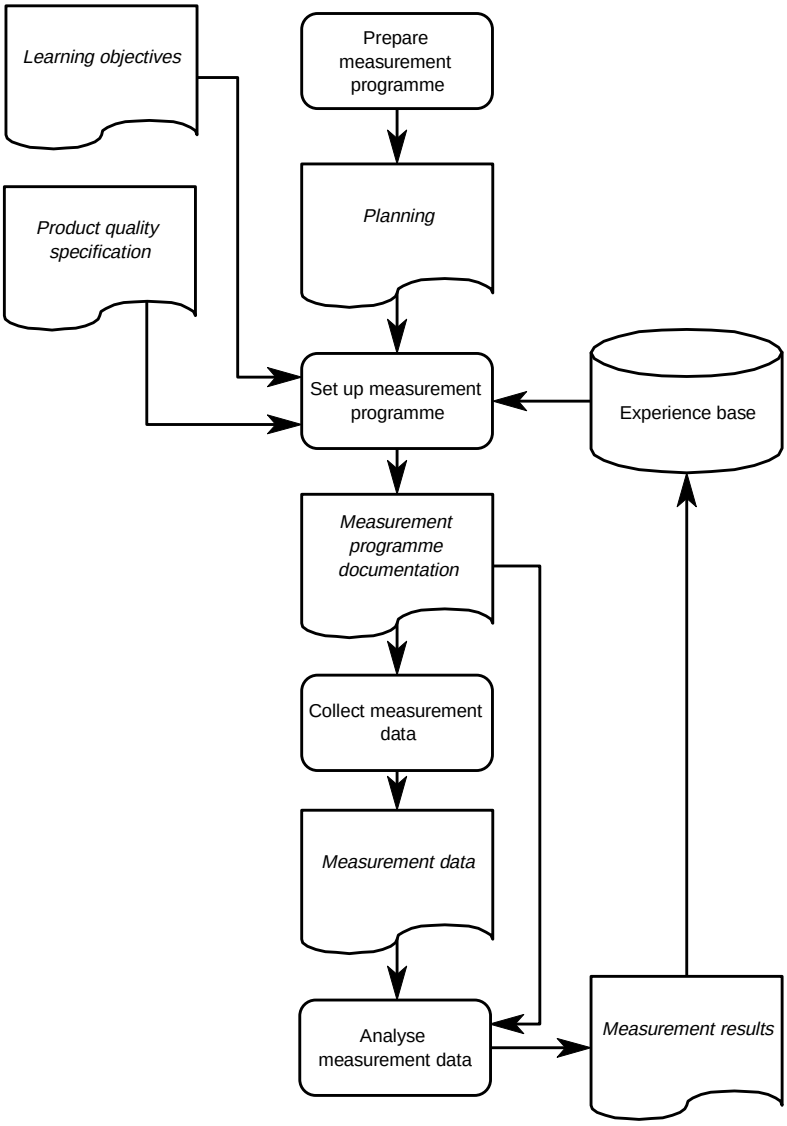


Figure 8-3: Measurement programme engineering procedure

Measurement programme engineering has two inputs (M-2): the learning objectives for which measurement goals should be defined, and the product quality specification to which the actual product should comply. First, the measurement programme is prepared by assigning dedicated resources for measurement support (M-7), training the software developers on software measurement programmes (M-1), and making a planning. After that, the measurement programme is set up along measurement goals, questions and metrics (M-3). Important input for this are the opinions, experiences and expectations (M-4) of the software development team. These people are consulted during the set up of the

measurement programme. The documentation of the programme, such as measurement plan, measurement goals, and data collection forms are used during data collection and data analysis. Based on this documentation, the measurement data is collected and analysed (M-6). The analysis is done in 'feedback sessions': interactive sessions with the software development team in which the team interprets the data, draws conclusions and defines action (M-5). The results coming out of these feedback sessions are evaluations of product quality (M-8) and evaluations of process action effects (M-9). This information is stored in the experience base for use in future projects (M-10).

8.3 Schlumberger RPS/Tokheim

Three of the four case-studies were carried out at the Bladel site of Schlumberger Retail Petroleum Systems. This division of Schlumberger was sold in October 1998 to Tokheim, the world market leader for retail petroleum products and services.

Tokheim produces Fuel Dispensers, Point of Sales, Electronic Funds Transfer equipment, Back-Office and Forecourt Controllers. The impact of software in the Tokheim products is increasing rapidly. In some cases 80% of a project's budget is spent on software development. The case-studies were carried out at the software development department in the Bladel site in the Netherlands. The software practices used at the Bladel site are certified according to TickIT [ISO 9000-3 1997] and ISO 9001 [ISO 9001 1994].

8.4 Tokheim WWC-project

The first case-study was the World Wide Calculator (WWC) project. In this project the central control unit, called: 'calculator' was developed for the new product family of dispensers. A calculator is the central component in a dispenser, which measures the amount of fuel taken by a customer. The calculator displays the amount of fuel, the price for the fuel transaction and the fuel price. A calculator controls the whole dispenser, meaning it controls the pumps and valves, but also communicates with the cash register or payment device. The WWC project was a sub-project of a larger project to develop a new product family of fuel dispensers. The main driver for this new product family was a 30% product cost reduction compared to the previous dispensers, and therefore the WWC project had also a 30% cost reduction target. Cost meant in this case: material and production costs of the hardware.

The WWC project was carried out in two sites: the Bladel site in the Netherlands and the Schwelm site in Germany. The team consisted of 3 hardware engineers, 4 software engineers, and a project manager. The project had a total duration of two years of which the first year resulted in a first increment of the product, which was expanded with functionality in the second year.

8.4.1 Experiences with Requirements Engineering

In this case-study, requirements engineering was used for the specification of product quality. All product quality wishes were evaluated by the project manager to decide to what extent a wish would be accepted or rejected. The result of this work was a product quality specification, in line with the guidelines of chapter 7.

During the inventory of the product quality wishes, eleven stakeholders were identified. The product quality wishes were captured in three interviews with stakeholders and representatives of stakeholders. This first case-study did not take into consideration that the development department was also a stakeholder.

The resulting product quality profile from this work is depicted in Figure 8-4. This figure shows the target quality, after selection by the project manager, for the product along the product quality dimensions. This figure was used to summarise the findings of requirements engineering and to provide a qualitative overview of what product quality is. The dimensions are the ISO 9126 characteristics expanded with time and cost. The scale of the axes is the A, B, C, D risk classification as presented in chapter 3[2]. Figure 8-4 shows that for this product: Cost, Time, Functionality and Usability have the highest values on this scale: C, which indicates a large financial impact when compliance to these requirements is absent. Figure 8-4 also contains the estimated product quality, a result of process engineering. Comparing the targets with the estimation reveals insights into which targets are least likely to be fulfilled and therefore require special attention.

The estimation of portability identified that it was unlikely that the targets would be reached. This mismatch between estimation and target was discussed with the project manager. He decided that the portability targets had been set too high and that the developers would not have sufficient time in the project to fulfil those targets. The project manager decided to decrease the portability target, as the result of an analysis carried out by him and a discussion on the impact of lower portability with the development team. This decrease of the portability target to the same value as the estimation is not yet visible in Figure 8-4.

---

[2] This scale was expanded with intermediate levels to make a better distinction between the quality characteristics. This expanded scale provides three intermediate levels [Uijtregt 1998]. For example from C to D the scale is: C, C^{-}, D^{++}, D^{+}, D.

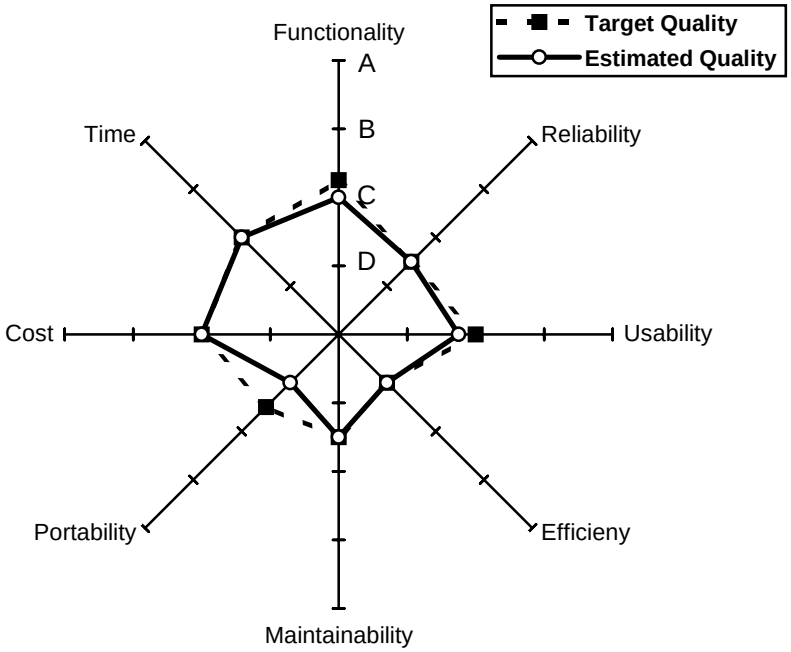


Figure 8-4: Product Quality Profile of the WWC product

Some wishes from manufacturing and service were deliberately rejected by the project manager, and therefore were not included in the target. Once the first version of the product was available these stakeholders restated the rejected requirements, because the current state of the product was insufficient for them. Corrective action was taken to improve the situation. This issue shows that rejecting stakeholder requirements does not necessarily imply that such requirements will not have to be addressed. Addressing these issues in later stages of the project is often much more expensive.

One of the experiences gained from this first RPM application was learning that the development department is also a stakeholder, and that its wishes should also be made explicit. For example, the product quality profile indicates that reliability is expected to be according to target; however, the development team had many arguments to support the idea that this would be a major risk area. The targets for costs and duration were so strict that the development team expected that they would have to squeeze their efforts at the end of the project during the testing and release work.

As a result of these arguments the project manager and team decided to select product reliability as improvement goal. During process engineering the development team therefore looked at process solutions that support in improving product reliability.

8.4.2 Experiences with Process Engineering

Process engineering was started with an assessment of the processes carried out, with an explicit focus on those process actions to create a reliable product. All process actions that influenced reliability were identified according to the development team. These process assessment results and the process-product relationship knowledge base were used to estimate product quality as has been visualised in Figure 8-4. This estimation was based on the process actions that the development expected to apply. Their impact on product quality was estimated and was compared to the product quality targets. Discrepancies between the two were used to set the product quality improvement goals, in this case: product reliability.

The findings of the process assessment and the comparison of estimated with target product quality did not result in major changes. The development team was confident that their normal way of working was sufficient to create the required quality, as long as they had sufficient time available to follow their normal procedures. Their expectation was that the time pressure in the project would negatively influence this and therefore negatively impact product reliability. The process improvement focus was therefore placed on the testing process, because this was the main risk and main contributor to product reliability in the final stages. Changes made to the testing process were: setting up test reporting, setting up test planning, assigning a dedicated test resource, reusing reliable software components and cross-personnel testing.

8.4.3 Experiences with Measurement Programme Engineering

A conclusion from the measurement programme was that in the project all software tests were carried out according to the test plan. The development team concluded that this was enabled by several reasons, such as a stable product architecture and some delays in mechanics development, but also for a large portion by the reliability focus of the measurement programme and the feedback sessions. The field test results showed that the test process was sufficient in identifying the main reliability problems.

It was shown in the case-study that measurement is a sound technique that can be used by a whole development team for focusing on a specific product quality attribute. Beside the explicit attention for reliability, the project manager noticed that reliability was also addressed in the implicit actions of the developers. The experience was that the measurement programme gave testing a higher priority in the process.

A set of questions is defined in a measurement programme, which are refined to a set of metrics. Data is collected for those metrics during the software development work. In this case-study the following metrics were collected:

- size of the tested object in source lines of code

- complexity of the tested object divided over three classes: high, medium and low
- type of the test performed divided over eight test types: module, integration, communication, system, stress, pre-approval, approval and field test
- test effort per tested object divided over three stages: test preparation, test execution and repair work
- number of defects found per tested object divided over three classes: minor, major and fatal
- test dates: planned and actual start date and planned and actual end date
- reason to stop testing divided over six classes: test plan completely executed, time for testing is spent, 'good feeling' that the tested object is ok, sufficient period without failures, sufficient external testing, other reason: explained in text
- stakeholder that detects a defect found after the test phase
- reason that a defect found after the test phase was not found before

An example of the feedback on one of these metrics that was provided to the development team is depicted in Figure 8-5.

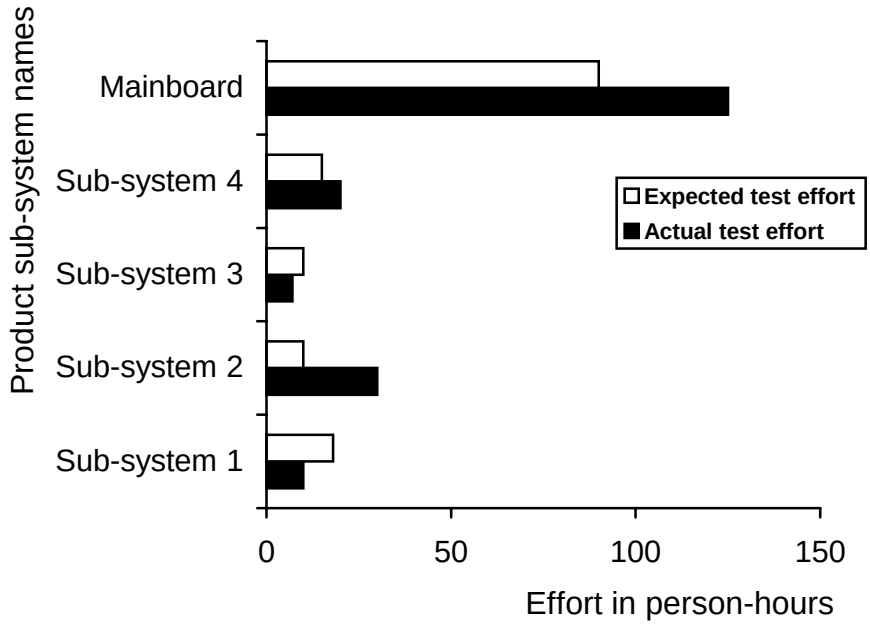


Figure 8-5: Test effort on the product sub-systems; expected versus actual

In Figure 8-5 it is shown that the effort spent on testing the sub-systems of the product, compared to expectations. In this chart it is visualised that in most cases more effort was

spent than expected on testing the sub-systems and mainboard and accompanying software. This was used by the development team to clarify why product reliability was better than they initially expected: they could spend more time on testing than planned originally.

An other experience of this measurement programme was that the test report was not a good medium for collecting all the measurement data. In the project all tests were carried out as planned, but reporting on these tests was incomplete. As a result there were quite some measurements missing. This made a large portion of the GQM questions difficult to answer because of this missing information. This might also be caused by the lack of explicit learning goals in this project. As the development team had no clear learning objective for the measurement programme, it was sufficient for them to carry out the tests to attain their goal: product reliability.

Comparing product quality metrics of the final product with the targets, identified that the overall targets were met. Some targets were set outside the needs or decreased during requirements engineering due to the estimated quality. As a consequence it appeared that the service departments were not so happy with the first version of the product, because the cost reductions caused lower serviceability than this stakeholder required. Furthermore, the manufacturing people were less happy, because the product cost reduction made the calculator less manufacturable. The calculator contained several electronic boards that had to be installed separately, increasing the time to install a calculator in a dispenser. Currently action is being taken to redesign the calculator as a generic box, which will decrease this production effort and increase the serviceability.

8.4.4 Experiences with applying the RPM approach

This case-study applied the RPM approach completely. Relevant product quality improvements were set and attained, process improvements were implemented, and the application of the RPM approach was integrated with the development project work.

The focus of the case-study was on requirements engineering and measurement programme engineering, but the reason for this was the specific demands of the project. So, it was shown that the RPM conceptual model is sufficiently flexible to address the specific needs of a specific project. Process engineering was carried out, but did not make major changes to the process. Merely a refocus of priority and effort assignment to testing was established which contributed to the attainment of the reliability targets.

Estimating product quality based on the intended process provided feed-forward information, and enabled the proactive control of product quality. One mistake made during requirements engineering was to leave out development as a stakeholder during the

interviews, but this was corrected by using their input for the selection and attainment of the product quality improvement goals.

Insufficiently addressing guidelines R-1, R-4, R-5 and R-6 during requirements engineering caused some product quality problems in a later stage of the project and corrective action was taken. This confirms the importance of these guidelines for requirements engineering.

Applying RPM appeared to be no burden for the project team, they were motivated to participate and experienced that their effort spent on these quality activities was relatively low. Evaluating the costs to the benefits showed the investment to be cost effective. The close co-operation of the development team especially with the QA department that supported the RPM application, was experienced as beneficial. The project manager experienced this co-operation as beneficial and expressed the view that this caused 'quality awareness' within his team.

Investigating the main success factors of this project revealed that having highly skilled and experienced engineers was one of the most effective reasons for the success of the project. Although the development team was happy with the result of RPM application, they expressed the view that good people remain a major condition for a successful project.

8.5 Tokheim OPT-project

The second case-study was the OPT project. The family of Outdoor Payment Terminal Products (OPT) are products that provide the facility to purchase fuel without the necessary intervention of a station operator or cashier. The fuel purchaser can initiate and complete a fuel purchase transaction with the use of an OPT. The OPT is equipped with several peripherals (functions):

- card reader: to pay with cards;
- cash acceptor: to pay for the fuel with cash
- user keyboard: to interact with the system
- user display: to interact with the system
- receipt printer: to provide a receipt

The OPT project was carried out at the Bladel site in the Netherlands, with a team of 3 software engineers and a project manager. The project was supported with the RPM approach for a period of six months.

8.5.1 Experiences with Requirements Engineering

During requirements engineering product quality was made explicit through interviews with stakeholders or representatives of stakeholders. The wishes from all stakeholders were made explicit and a selection was made for the product quality specification. In total nine stakeholders were identified and three interviews were held. During requirements engineering in this second case-study, development was considered to be a stakeholder.

The results of requirements engineering are visualised in Figure 8-6. In this figure the quality targets are shown for the product along the ISO 9126 characteristics on the A, B, C, D scale explained in chapter 3. In Figure 8-6 it is shown that the Functionality, Reliability and Efficiency product targets have the highest value. Furthermore this figure contains an estimation of product quality as a result of process engineering.

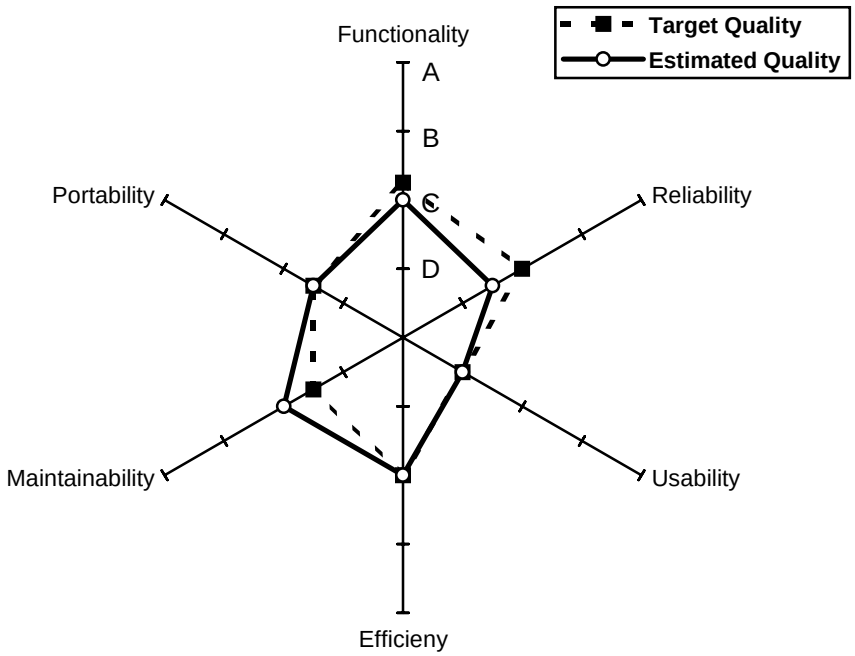


Figure 8-6: Product Quality Profile of the OPT product

In Figure 8-6 it is shown that the estimated values of functionality and reliability are lower than the targets. Furthermore, the maintainability estimation is higher than the targets, which is remarkable. Discussing this finding with the development team revealed that they confirmed that their way of working results in a level C maintainability, but according to them this was necessary. The development team took the position that the maintainability target was too low. After discussing this issue with them it appeared that the development team puts the emphasis on maintainability because it makes the work for them easier when

changes are required. Investigation of the stakeholder wishes showed that this is mainly a wish from the developers and that this emphasis on maintainability could be decreased; however, the development team decided not to change their process, because they were quite happy with the current maintainability of their software.

8.5.2 Experiences with Process Engineering

The process assessment results and the PPR knowledge base were used to estimate product quality, as visualised in Figure 8-6. Comparing these estimates with the targets revealed that functionality and reliability were expected not to fulfil the targets, while maintainability was expected to become higher than the target. The development team did not share the estimate of 'too low reliability'; therefore they decided to investigate reliability using a measurement programme.

Initially process engineering was only carried out to create an estimate of product quality. The next part of process engineering: configuring a process was deliberately not done: the development team did not support the need for a process change and wanted to investigate product reliability first, before making changes.

The reason that product reliability was estimated lower than the target was because it appeared that the development team did not apply all process actions for reliability completely. Trade-off decisions on time and expected outcome were made in the project implicitly by the engineers, which meant that in certain cases process actions were partly taken. As a consequence the effects of these process actions became insecure and this caused the product reliability estimate to be low. So, it was decided to first measure product reliability before making changes. This is typical single loop learning; no double loop learning goals were set.

8.5.3 Experiences with Measurement Programme Engineering

The product measurements led to the conclusion that product reliability complied to the targets. The trade-off decision of the development team on application of the reliability process actions appeared to have been done sufficiently. No big changes were required. One improvement that was carried out was the design and distribution of a test checklist to be used by the service departments. This was expected to detect problems in the most critical product areas before the product was installed in the field.

The metrics that were collected in this measurement programme were:

- size of a module in source lines of code
- complexity of a module divided over three classes: high, medium and low
- amount of reuse of a module divided over five classes: 0%, 25%, 50%, 75% and 100%

- level of review divided over two classes: review has been done and review has not been done
- number of defects found per module divided over three classes: minor, major and fatal
- reason that a defect found after release was not found before, divided over seven classes: no module test, no regression test, too little test time, not possible to test, hardware reasons, not reviewed, external reasons
- detector of each defect found after release divided into three detector classes: engineering, service department, end-users (field)
- type of service problem report divided over failures and change requests

One feedback example with one of these metrics is depicted in Figure 8-7.

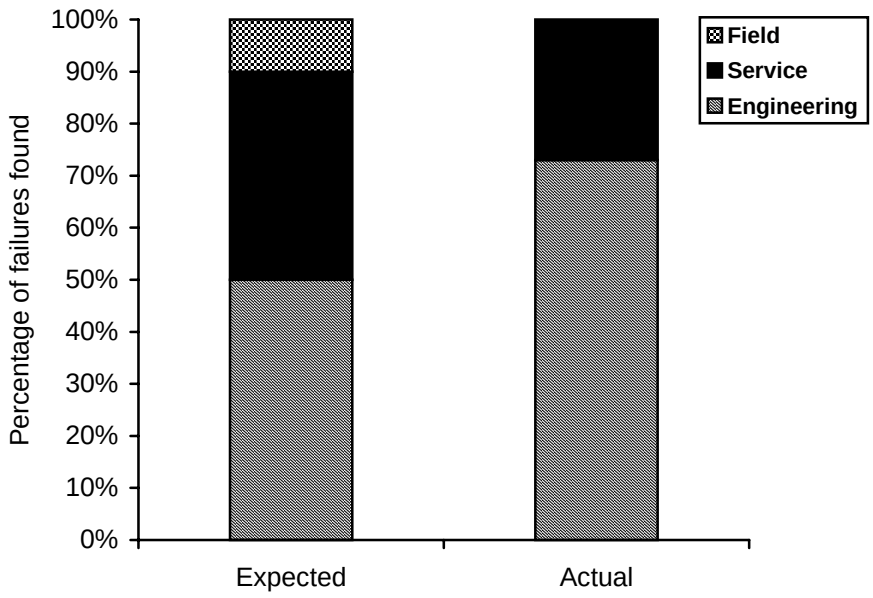


Figure 8-7: Chart that shows the stakeholder that detects a failure versus the hypothesis

In Figure 8-7 the percentage of failures detected by the three product stakeholders that identify failures is shown. It appears that the development team is much more important for the detection of failures than the other stakeholders. This gave them the insight not to rely too much on other stakeholders for making the product reliable. This learning point had occurred before in the company in another project [Latum et al. 1998] [Solingen and Berghout 1999], in which the detectors of product failures were also measured. In this project it also became clear that 75% of the product failures are found by the developers, while the assumption is made that other stakeholders are also important failure detectors.

This insight mainly caused the development team to set up this testing checklist for the service department, because they were of the opinion that the service departments were a good source for finding failures, but that this was apparently not yet established.

After the first feedback session in which the goals had already been attained, the project leader (senior engineer) and one of the developers left the company. They could not be replaced immediately, because of a shortage of qualified people. The vacuum that was created because of this change, caused the product development to be put on hold, and influenced the continuation of the improvements. New objectives were not defined, since the project manager wanted to wait until a new team had been established.

8.5.4 Experiences with applying the RPM approach

This case-study was a relatively small application of RPM. It was carried out over a short period of time with a small development team for a relatively small project. This case-study applied the RPM approach only partially, since the role of process engineering was quite limited; however, the experiences showed that RPM supports the identification and attainment of product quality targets. The activities for RPM were integrated in the development project activities, and the developers experienced that the effort required was small and the benefits exceeded the cost. The specific wishes of the development team were considered and the RPM approach provided sufficient flexibility to support them in developing a quality product. The development team was enthusiastic and motivated about the RPM approach. They experienced that the approach addresses issues that are relevant for them and that it is not a burden in their work. It also resulted in improved visibility of product reliability and some process improvements.

The need for process changes did not become clear from the product quality requirements. Therefore a measurement programme was set up to investigate product quality and to identify the necessary process changes. This is a different role for measurement programme engineering than it was designed for, but it appeared also to suit this purpose.

Although costs and time issues were not considered during requirements engineering, they appeared to be clearly present. For example: during the measurement programme it became clear that certain defects were not detected in the testing phase due to the limited amount of test time available. This was acceptable for the team, because this shortened their time to market, although it negatively influenced the reliability of the product. The conclusion from this experience is that costs and time issues should also be considered as part of product quality during requirements engineering.

The final observation from this case-study was that a change of development team members endangers continuation of the RPM application. In this case-study half the

people of the development team left the company after the first feedback session. As a result the improvement programme was stopped earlier than intended.

8.6 Tokheim Omega-project

The Omega project was aimed at the functional extension of an existing fuel-station management system. The project consisted of several sub-projects to build additional functionality. Omega is a retail automation system designed and developed specifically for the needs of service station managers and operators. The Omega system is modular and configurable from a simple fuel pump console to a comprehensive multi Point Of Sales (POS) configuration with a dedicated 'Back Office' workstation for site management purposes. The Omega system is a PC-based embedded system. It consists of several computers that are linked and in this way are used to manage a fuel station. Proprietary hardware is included to perform communication with the fuel dispenser calculators, outdoor payment terminals and other external equipment on the station forecourt. A large part of the system functionality is developed in software (C++ on an OS/2 platform).

The Omega project was carried out at more than ten development sites. The central development was initially carried out at three sites, but is currently dealt with at one site. At this central site ten developers work on the project. In total about 25 persons are developing software for this product line. Initially this improvement programme focused on the whole development team, but in the later stages the focus moved to the system testing process of the test team. The test team consisted of three testers and a manager.

8.6.1 Experiences with Requirements Engineering

Requirements engineering was carried out for the specification of product quality. Twelve stakeholders were identified and five interviews were held with these stakeholders or representatives of these stakeholders.

Figure 8-8 shows the product quality targets of the Omega product along the ISO 9126 characteristics. The scale of the axes is the A, B, C, D scale explained in chapter 3. The chart in Figure 8-8 shows that functionality, reliability and maintainability have the highest scores. Furthermore, Figure 8-8 contains the results of estimating product quality when using the set of process actions identified during a process assessment. This product quality estimation is a result of process engineering.

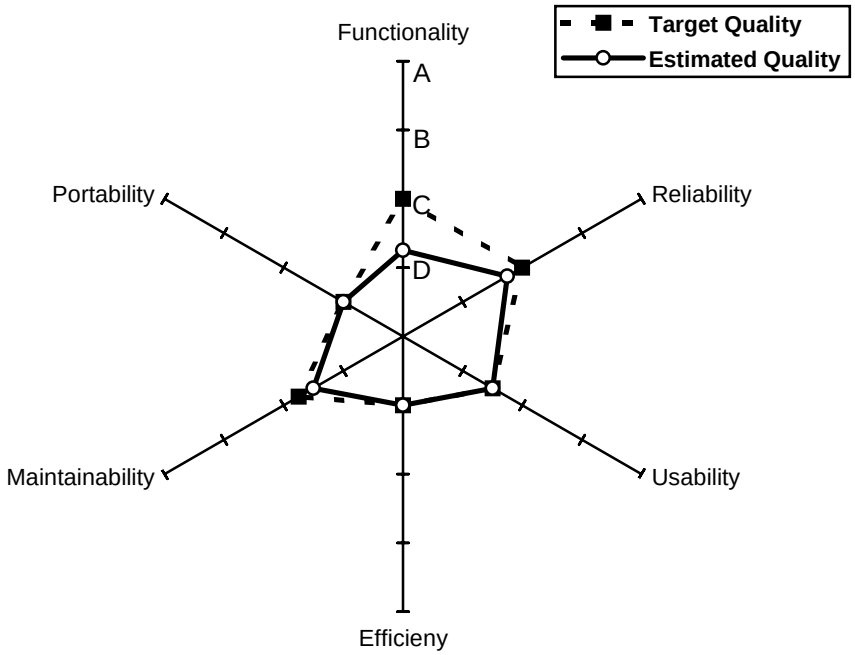


Figure 8-8: Product Quality Profile of the Omega product

It is shown in Figure 8-8 that Functionality is an area that requires the most improvement. The product is sold to a large number of different countries and different customers, for which the functional product requirements are often unknown, resulting in a too low score on estimated functionality. Furthermore, reliability and maintainability were expected to be lower than required. The chart in Figure 8-8 appeared to be a good tool to provide an overview for the development team on product quality targets and the status of their process. The development team supported the findings; however, they had the opinion that the product functionality issue was not something they were responsible for, or could do something about.

Maintainability was expected to become too low, as a direct consequence of the functionality problems. Due to insufficient functionality, many changes could be expected to the product in the future, creating higher maintainability requirements. This illustrates that product quality attributes are interrelated.

8.6.2 Experiences with Process Engineering

The results of a process assessment were used to estimate product quality for the quality profile. Based on the findings displayed in Figure 8-8, product reliability and functionality were selected as product improvement goals.

The two main changes that were implemented in this project were:

- the introduction of multi-site configuration management
- the installation of an independent dedicated test team

The PPR knowledge base was used for the selection of these process improvements. The selected process changes and their expected impact on product reliability and functionality are visualised in Figure 8-9 (++ means a medium positive impact, +++ means a high positive impact).

Process actions	**Reliability**	**Functionality**
Put all Omega deliverables under CM		++
Detailed process for CM		++
Module audits	++	
Full regression testing	+++	+++
Full system testing	+++	
New test methods	+++	
Automate testing	+++	
Requirements acquisition in planning	++	

Figure 8-9: Selected process actions and their expected effects

Experience with using the PPR knowledge base for identification of additional process actions shows that many process actions are not selected because their impacts occur after a period in which the development team learns to apply it. Most process actions do not give their effects immediately after initial introduction. This supports the finding that additional learning goals are necessary to experiment with new process actions on less critical product issues in order to train developers in applying such a process action. Learning a new technique that does not result directly in the intended effects will not be selected if that effect is critical. The consequence is that one focuses on single loop learning, and the double loop learning effects are disabled.

Performing queries to the PPR knowledge base to provide information on possible process actions was successful. The knowledge base was able to provide optional process actions for all product improvement targets. This depends of course on the amount of knowledge stored in the knowledge base. In this case-study, the knowledge base was filled with more than one hundred process actions [Soerjoesing 1998].

8.6.3 Experiences with Measurement Programme Engineering

The measurement programme for the Omega project was focused on the independent and dedicated test group. The measurement goal was to understand the relevant parameters for the testing process, in such a way that a decision could be made on balancing the testing efforts to cost, duration and Omega product quality. The project has been guided using measurement for one year at the moment of writing this thesis. Measurement and feedback is still being continued.

The test team learned their baseline performances, such as the amount of time they need to test certain functionality, time required to set up the test environment, duration of a full system test cycle, etc. Furthermore, the measurement programme identified that the test team under the current circumstances is unable to carry out full system test cycles. The frequency in which they are confronted with new versions of the product is too high to carry out a full test cycle. This information was used to clarify why certain failures were not detected, and to communicate and negotiate this release frequency. During the feedback sessions of the measurement programme, the test team also performed causal analysis on the defects found in the field. These causal analysis had the objective to identify the reasons that certain failures were found under field conditions and not by the test group. In most cases the reasons appeared to be that functionality had not been tested because of the time pressure. One of the other reasons appeared to be the 'positivistic test style' of the test team: they did not test to find failures, they tested to prove that the product 'worked'. Special training was held for the test group to provide them with the skills to carry out good system testing and to decrease the number of failures that slip through their test process.

Several metrics were collected to support in answering the questions set in this measurement programme. The GQM questions specified for this measurement programme were:

1. What are the preconditions for good testing, and are these preconditions met in practice?
2. What are the costs of testing?
3. What is the contribution of the testing process to Omega product quality?
4. What is the duration of the test process?
5. What are good criteria for the decision to stop testing and release software for field testing?
6. What are the rules of thumb on the test process of the testing group?

These questions were refined into the following set of metrics:

- availability and perceived quality of test script and software specification
- effort spent on testing subdivided over: test script definition, test script selection, test script execution, and testing the failure resolution
- severity of detected failures divided over four classes: cosmetic, minor, major and fatal
- status of a failure divided over five classes: open, ready for testing, pending, killed verified
- duration of a test cycle for a pre-release and for a full release

A feedback example on one of these metrics to support in answering question 3: 'What is the contribution of the testing process to product quality?' is depicted in Figure 8-10.

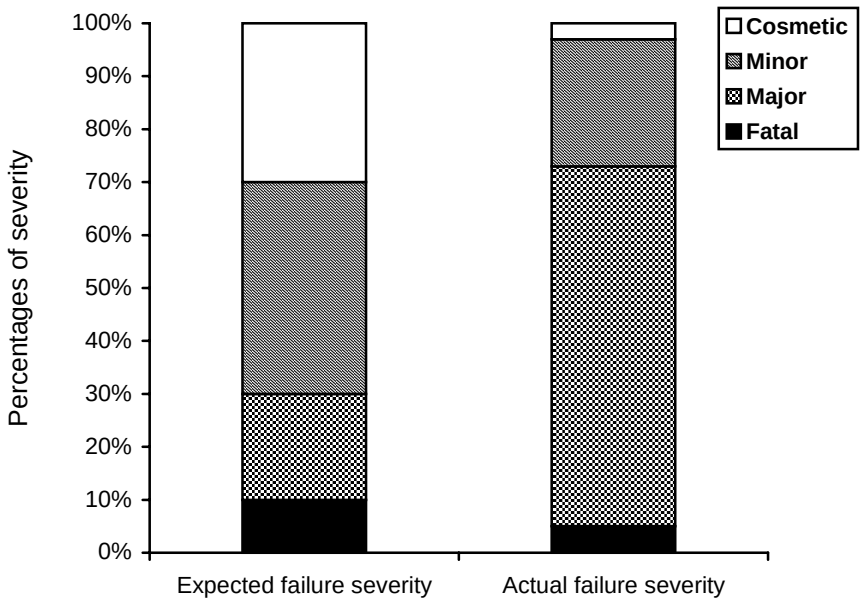


Figure 8-10: Chart that shows the severity of failures found by the test group versus their expectations

It is shown in Figure 8-10 what the differences are between the severity of the failures found by the test group and their expectations (hypotheses). It appears that the severity of the failures found is much higher than expected. The test group initially expected to be confronted with a quite stable product in which they would try to find the deeply hidden problems. Practice showed them that this was not the case. They were confronted with large numbers of serious problems, which caused the number of minor issues to be much lower than expected. Furthermore, this high number of serious issues was also one of the main reasons for the high release frequency, because the test group was often required to

have a new release in which these problems had been solved, before they could restart testing. The information in the measurement programme enabled the test group to evaluate their own work and learn basic rules of thumb for their work. In addition to that they had enough information to show the effectiveness of their work process and to communicate to others what the causes of ineffectivities were. The information in Figure 8-10 could, for example, be used to communicate to the development team that their module testing process was inadequate, and that this was the main cause for the insufficient regression testing by the test team. The test group is still active in improving their process and trying to improve the conditions under which they have to carry out their testing work.

8.6.4 Experiences with applying the RPM approach

This case-study was a full application of the RPM approach. Requirements engineering was used for product quality needs identification, process engineering was used for product quality improvement identification and process change selection, and measurement programme engineering was used to guide the improvements and learning the effectiveness of the process changes.

The RPM approach was integrated in the daily work of the testing team. The QA department supported them in the non-learning tasks, and the test engineers participated in those RPM activities that required their involvement, such as interviews, target setting, and feedback sessions.

Focusing the improvements efforts on those product quality attributes that required the most improvement was supported by the RPM approach; however, even though the largest problem area in this case-study appeared to be functionality, the decision was taken to focus on the reliability aspects. This decision was made by the development team because they had the feeling that they had no impact on this functionality problem. Only improvement areas were identified by the RPM approach that the team supported which is a strength of the approach; however, the largest problem was still not addressed in this specific case.

The overall conclusion is that the RPM approach supported the project well, and the RPM activities were integrated in the work of the test team. Analysing the costs of doing the RPM work versus the benefits showed that RPM application was cost effective.

8.7 Dräger Medical Technology

Dräger is a 1.4 billion DM multinational operating primarily in the fields of medical and safety technology, with limited operations in aerospace technology. It has about 8100 employees. The three divisions of Dräger are Medical Technology, Safety Technology and Aerospace. The core business of Dräger Medical Technology is the development,

production and service of gas monitors, single and multi-parameter patient monitors, fluid pumps, incubators and defibrillators for application in anaesthesia, intensive care, neonatal and emergency care. These products have recently been extended with network and central station capabilities to enable parallel monitoring of a number of patients as well as to enable central documentation.

8.8 Dräger Medical Technology BSW project

Dräger MT-M, the monitoring sub division of Dräger MT, developed a complete new line of patient monitoring devices. This family of devices can be used to create a BedSide Workstation (BSW) around each bed location in a department in a hospital. The BSW's are intended for use at the intensive care stations and in the operation room. The system incorporates network connections between various elements of the system, and allows patient data exchange and the viewing of patient data at several locations.

The development of BSW was organised in a project. The development activities took place on two sites: at Lübeck, the home city of Dräger in Germany, and at Best in the Netherlands. To develop the BSW product, multiple disciplines were required. Five Dräger hardware engineers designed the electronic components. All the software, both for the embedded systems and the PC products, was developed at the Dräger office by a team of 30 software engineers.

8.8.1 Experiences with Requirements Engineering

Requirements engineering was not carried out as described by the RPM approach. No product quality requirements were specified as described in this thesis. The product improvement targets were set, however, they were based on the opinion of the project manager and the developers, and not based on consulting the multiple product stakeholders and comparing the wanted quality with the current or expected quality. These targets were specified in ISO 9126 terms, and a product quality profile was made.

The impact of this lack of product quality requirement specification is not as yet clear. The company has considerable experience in the field, so it is well possible that the people have indeed selected the right quality improvements; however the project has still not been introduced to the market, so this feedback has not been provided for the case-study. The focus of the project has been on improving product reliability and usability.

8.8.2 Experiences with Process Engineering

The development team specified the product improvement targets based on their personal experience and knowledge, and based on the outcome of the process assessments. During

assessments, the interviewees and the assessors identify product improvement areas that served as an input to set improvement targets.

Changes to the processes were made in this project, based on these product improvement targets. The organisation did not have access to a PPR knowledge base, so the process changes were not identified from the knowledge base. The process changes were based on expected effects on product quality, or, in other words: based on implicit PPR models of the development team and the project manager. These implicit PPR models were made explicit by stating the expectations (hypothesis) and measuring whether these expectations were valid during measurement programme engineering.

The changes made to improve product reliability were:

- incremental development
- implementation of test strategy
- inspection of design documentation
- design robust system and software architecture

This approach of selecting process actions is limited to the knowledge of the project manager and the developers, and therefore is limited to single loop learning. A double loop learning strategy would also look at process actions that are not so obvious at first glance, or that the people have no experience with.

8.8.3 Experiences with Measurement Programme Engineering

Measurement programme engineering was carried using the GQM approach [Basili and Weiss 1984] [Solingen and Berghout 1999]. Measurement programmes were implemented to measure product reliability, the effectiveness of inspections, and the effectiveness and efficiency of the testing process.

In total six feedback sessions were held in which the data was analysed by the development teams. These feedback sessions were held with a large number of people (30-50), which made discussion difficult. As a solution the results were discussed in a smaller session with the group leaders.

The measurement programme is still in progress and product reliability is still being measured.

8.8.4 Experiences with applying the RPM approach

Applying the RPM conceptual model was integrated with the development practices, and the development team was enthusiastic about the results. This case-study applied RPM only partially. Requirements engineering was implemented only implicitly. The product quality improvement targets were directly specified by the development team and their

manager, without consulting stakeholders and comparing the wanted quality with the current or expected quality. The impact of insufficiently addressing the requirements engineering guidelines is not clear currently because the project is not finished yet.

Two ways of organising the application of RPM were tested. In this case-study firstly, no dedicated support team was established for RPM application. Two software developers were scheduled part-time to work on the improvement programme. When product development came to a critical phase, this effort was reduced, putting the whole improvement programme at risk. After a period without activities on the improvement programme, a dedicated resource was hired to perform the work as a separate resource, which guaranteed continuation better.

It became clear that using the concept of PPR models is also feasible in a situation in which no knowledge base is available. The goal-driven concept that changes should have specific expected effects that would contribute to the product targets, was perceived as valuable. This could be made explicit from the process changes in the project, because almost all process actions that were established had an implicit PPR model behind them.

8.9 Benefits of RPM application in the case-studies

A cost/benefit analysis was carried out on RPM application in the three Tokheim case-studies. Not enough data was available for the Dräger project to carry out such a cost/benefit analysis within the scope of this research.

The direct benefits of RPM application at Tokheim are presented in this section, divided over the three working areas separately. In addition some indirect benefits are also identified.

Identifying the benefits of RPM application was done in two ways:

- analysing the measurement results
- observing the people in the organisation and discussing effectiveness of the approach with them

First of all, the benefits of RPM application become clear from the explicit improvements taken in the projects and measuring their results. This explicit information can be used to identify the impacts and benefits of RPM application.

Furthermore, the observations of the researcher have been an important source of information for the identification of benefits. The researcher was present for all activities carried out for RPM application and could see what the effects were in the development projects. The researcher discussed results with the people in the improvement programme meetings, and during informal off-line conversations. The researcher relied on observation to measure the perceptions by the development teams and their experiences with RPM

application. An alternative could have been questioning the development teams using for example a structured survey; however, this would have been overkill in this environment, and would probably not have been appreciated by the development teams.

8.9.1 Benefits of Requirements Engineering

During the case-studies it was confirmed that requirements engineering is important. The specification of what product quality actually means improves the way in which product quality is addressed by the developers. During all the case-studies at Tokheim a full requirements engineering phase was carried out in which the requirements of all stakeholders were specified. As a consequence the requirements for product quality were openly discussed, and the outcomes were commonly accepted. Furthermore, it became possible to evaluate compliance of product quality once the product was available.

Specific observations in the case-studies triggered the importance of requirements engineering. For example in the Tokheim WWC case-study some product quality targets were set lower than the needs, because of costs and duration reasons. These product requirements were restated as soon as the product became available, and were addressed in this later stage. This showed the importance of requirements engineering and the importance of seriously considering all stakeholder requirements.

Another example of an observed benefit of requirements engineering was that in the Tokheim Omega case-study, the requirements engineering work identified that there was clearly a product functionality problem. The development team supported this conclusion but had the opinion they could not take action on this; however, the results of requirements engineering gave them some additional arguments to convince higher management to take action, and to support them in solving this product quality problem. This example supports the benefits that can be achieved by addressing product quality explicitly, and making an explicit product quality specification for a product.

8.9.2 Benefits of Process Engineering

The case-studies supported in the configuration of a product specific process depending on the specific project targets. In all case-studies specific improvement objectives were set and corresponding specific process changes were selected and implemented.

The case-studies revealed that an overview on the current established process actions supports in the selection of product improvement targets. Estimations were made of the product quality that was expected to come out of a certain process. Comparing this estimated product quality with the target product quality revealed those product quality attributes that were at risk. In the case-studies it was also shown that such graphical

representations of the product quality profile, could be discussed with all kind of stakeholders and were a suitable medium to communicate results of process engineering.

The selection of product quality improvement goals was supported by process engineering, which also supported the projects in selecting the process actions to attain these goals. During the case-studies some specific observations triggered the importance of process engineering and identified its benefits. For example during the Tokheim WWC project explicit priority was set for the testing process and a dedicated resource was installed for product testing. Making such explicit process changes, based on the product quality improvement objectives, is a benefit of process engineering. During process engineering explicit attention is paid to the tuning of the development process to the project objectives.

An important benefit of the combination of process engineering and requirements engineering is that the right product quality and process improvement goals are selected. At Tokheim there was already much experience with measurement based improvement programmes, however, selecting goals appeared to have been difficult in the past [Latum et al. 1998]. The iteration between requirements engineering and process engineering supported in selecting the right goals.

8.9.3 Benefits of Measurement Programme Engineering

The benefits of measurement programme engineering are divided over the two purposes: evaluation of product quality, and evaluation of process effects.

Regarding the evaluation of product quality it was shown in the case-studies that measurement programme engineering is a relatively fast and easy way to perform this evaluation. The product quality specification already contained the metrics and the target values, and feedback was provided to the development teams regarding compliance to these targets. Furthermore, the measurements made it feasible to provide feedback to stakeholders in their own language, because information was available on product quality according to the metrics specified by the stakeholders. Thanks to the regular feedback sessions in which the measurements were analysed, early feedback on product quality could be provided to the development team making corrective action possible in earlier stages than in situations where feedback on product quality was given during field tests or system tests.

Regarding the evaluation of process effects, all case-studies identified that measurement programme engineering is the main facilitator for learning such effects. During the measurement programme, and especially during the feedback sessions, the development team learned the effects of their own work. Specific process actions were closely monitored with measurement to determine the detailed impacts and success factors. This learning process focuses especially on the product quality goals of interest and therefore is

a beneficial activity. The feedback sessions appeared to be the most effort consuming part of the improvement programme, but at the same time the part that all the people appreciated most.

During the case-studies specific observations were made that triggered the confirmation that a measurement programme is beneficial. For example, during the Tokheim Omega case-studies, the independent test team thoroughly evaluated the problems that slipped through their test process, and that were detected in field situations. These analyses were normally not made, but because of the measurement programme they were carrying out, objective feedback was provided on where to look for improvement. An other example of the benefits of measurement programme engineering comes from the Tokheim OPT case-study where the feedback sessions were the forum in which representatives of development, marketing and quality assurance discussed the status of the product, and balanced corrective action against the cost and duration issues.

8.9.4 Indirect benefits

The application of the RPM approach also had some indirect benefits that are significant, and that might in some cases be worth the investment.

The establishment of a ‘quality culture’ or ‘improvement focus’ in the development teams is an important benefit. Due to the special attention paid to product quality, the explicit specification of product quality targets, explicitly linking the process improvements to these targets contributed to those benefits. The teams became interested in product quality and started implicitly to take actions for product quality. Furthermore, it appeared that as long as the teams were the ones that defined the process changes, they were motivated to participate in the improvement programme. There was no ‘resistance to change’ because the team decided on the changes, and the changes were explicitly related to the objectives of the project.

An other indirect benefit was the co-operation between the quality assurance department and the development teams. Historically the quality assurance department had a role in which it dealt with only checking available procedures and audited projects on compliance to these procedures. Application of the RPM approach changed this role. The quality assurance people became actively involved with product development and supported the development team in setting the right product targets and supported them in configuring the right process and providing the feedback on the execution of that process. This increased co-operation was experienced as beneficial for both the development teams and the quality assurance people.

The last indirect benefit of RPM application that was observed was the learning processes within the teams. Mainly due to the feedback sessions, the development teams evaluated

their own work, discussed this within the team, and concluded on the effectiveness of their processes. This learning process was an important benefit of the RPM approach, because learning is explicitly addressed.

8.10 Cost of RPM application in the case-studies

The main costs for RPM application are in the time spent by the people on the improvement programme. The cost of RPM application in the Tokheim case-studies are measured via the effort spent on the improvement programme. The effort spent on a task was stored for each task carried out during RPM application. Transforming these effort numbers to financial values could be done using the costs for one person-hour.

Figure 8-11 contains an overview of the effort spent on the case-studies distributed over the three working areas, and over the roles involved with RPM application. Two roles are discerned: development team and facilitators. The 'development team' is the people that design and build a product. These people are supported by the RPM approach to address the right product quality goals. Much of the RPM work does not necessarily have to be done by the development team, but is done by 'facilitators'. These facilitators were people from the quality assurance department that supported the development teams with an improvement programme.

	Requirements Engineering	Process Engineering	Measurement Programme Engineering	Total
WWC project				
Development team	16	12	45	73 (24%)
Facilitators	40	40	156	236 (76%)
Total	56 (18%)	52 (17%)	201 (65%)	309 (100%)
OPT project				
Development team	18	5	15	38 (18%)
Facilitators	54	35	80	169 (82%)
Total	72 (35%)	40 (19%)	95 (46%)	207 (100%)
OMEGA project				
Development team	20	16	48	84 (25%)
Facilitators	50	63	137	250 (75%)
Total	70 (21%)	79 (24%)	185 (55%)	334 (100%)

Figure 8-11: Table with effort expenditure on RPM application (in person hours)

The effort spent by the development team was about 1% of their time. This is considered to be low according to the development team standards. For example, compare this 1% on

product focused SPI with the 20% effort they spent on the non-planned interruptions of daily work [Solingen et al. 1998]. Support given to the development teams by the facilitators was about 75% of the costs of the improvement project, the remaining 25% effort was spent by the development team.

Other data shows that the effort required for RPM application was roughly distributed over the three working areas as follows: Requirements engineering: 25%, Process engineering: 25% and Measurement programme engineering: 50%. It shows that measurement programme engineering requires most effort, and especially the feedback sessions.

Figure 8-11 only contains the operational 'cost' for applying the RPM approach. The initial costs required before an organisation can apply an improvement concept such as RPM are not included, nor are the costs for the maintenance of an infrastructure that supports RPM application included in this table. These cost data can not be acquired from the case-studies, because the initial costs are not representative since the RPM approach was also developed in this period, and the maintenance costs are unclear because the period in which the approach was applied was too short to draw such conclusions. It is, however, expected that these costs will be considerably lower than the operational cost for applying the RPM approach, depending on the motivation and willingness of the people for improvement and learning.

8.11 Are the benefits worth the cost?

The total effort spent per project was less than 3 person months each year, of course depending on the number of developers in a project. Of this effort only 25% (3 person weeks) was spent by the development team, which is less than 1% of their time. The facilitators spent the remaining effort (9 person weeks). The amount of effort spent on an improvement programme using RPM is rather small, and it should be clear that not many benefits are required to make application of RPM cost effective. For example, past measurements at Tokheim have shown that a problem detected early in a development project saves 2 weeks of development effort [Solingen and Berghout 1999]. If RPM application results in 2 problems detected early, its application is already cost effective for the development team. If 7 problems are detected early due to RPM the whole improvement programme is paid back.

The section on RPM benefits has revealed that many benefits could be identified during RPM application. This large amount of benefits and the relative low costs for RPM application leads to the conclusion that RPM application will be cost effective in most cases, and that the pay back time can be short.

8.12 Validity of the case-study findings

After looking at the results of these case-studies, the question remains whether the RPM approach and its guidelines are valid. In the case-studies it is shown that using the RPM approach results in: a product focus, the definition of specific product targets, process changes, and attainment of product targets. As such one can conclude that the approach fulfils its promises, which leads to the conclusion that internal validity is confirmed. In chapter 2 it was discussed that the selection of the case-studies promises to result in externally valid results, because they sufficiently cover the embedded product domain. One questions still remains before it can be concluded that the approach is indeed valid. This question is:

What would have happened if the RPM approach had not been applied in the case-studies?

The positive results in the case-studies are claimed to be caused by the application of RPM; however, these results might have been similar without applying this approach. To answer this question, a closer look was taken at the contribution made by RPM to the selection and attainment of the product goals. Furthermore, it will be analysed whether these goals were the right goals for the projects in the case-studies.

8.12.1 Contribution of RPM

It is difficult to determine objectively what the contribution of RPM was, because one can not compare it with a situation in which RPM was not applied. The solution found for this validation was twofold. Firstly, to look at the specific changes in the working processes and the causes for these changes. If such a change was caused by RPM application, it is assumed that this change would not have been made without RPM. Secondly, the development team was asked what their opinion was about the impact of RPM application, and whether the changes to the process were beneficial and would have been made without RPM.

The impact of the RPM approach on the changes in the process and the impact of these changes on the product was investigated. According to the findings, requirements engineering largely supported in the identification of the product quality targets. The results also showed that process engineering, and especially process assessments, had an impact in the selection of the product quality improvements. The process assessments supported in identifying these improvements, but were also an important source of information to carry out the product quality estimations. The impact of measurement was different for the projects. The impact of the measurements in the WWC project were low, although the development team claimed that the feedback sessions supported them in their focus on product reliability. In the OPT project the measurements were only used to

confirm expectations, since no unexpected issues arose; however, in both the Omega and BSW projects, measurement played an important role in learning effects of certain process actions, and in identifying and monitoring the changes to the processes. The impact of the measurement programmes was considered higher when it had an explicit learning purpose.

The development teams claimed that the impact of RPM application was higher than the study showed[3]. The main reason for this is probably that the RPM approach not only causes changes to occur, but also supports in the establishment of a quality-oriented improvement 'culture', which was also a major change for most of the people. Learning and improvement had not been one of their objectives before, and it was observed in the projects that these cultural aspects also caused some impacts on working processes implicitly. One of the project managers stated that the 'quality awareness' of people increased largely because of the application of the RPM approach and the co-operation with the quality assurance department.

8.12.2 Addressing the right product quality goals

When looking at the product after field introduction, it is possible to identify the main problems, and to identify whether these were also identified by the RPM approach.

Analysis of quality problems with the product after development, showed that the RPM approach successfully identified the right product quality attributes that needed improvement. Those were, in all cases, also the attributes for which the improvement goals were defined. One exception was the Omega project that decided not to address the functionality issues sufficiently, since the development team took the position that they were not able to solve these problems. Analysis of the main product problems showed that these were directly related to this issue: suitability was weak and as a result changeability became a much stronger requirement which was also no longer sufficient. The overall conclusion is that the RPM approach sufficiently supports in identifying the right product and process quality objectives.

8.12.3 Validity of the guidelines

The last validation issue is the validity of the guidelines. It is impossible to validate each guideline individually in industrial case-studies, because they are all applied at the same time, influenced by external variances and also influencing each other. What can be done,

[3] Hawthorne effects are unlikely due to these experiences because there was no explicit research carried out. The RPM application was carried out as facilitator of an improvement programme and the objective was to make changes and measure the impacts of these changes. The fact that this was part of a research project is expected to have no impacts to the outcomes of the improvement programme.

however, is to identify the coverage of the guidelines by the approach used, and analyse whether non-appliance of a certain guideline resulted in specific problems.

The guidelines were largely covered by the approach applied in the case-studies. One major difference was the BSW project in which requirements engineering was not carried out, because the development team decided on the product quality targets. As this project is still in progress, it is difficult to identify the impact of this difference.

Regarding the guidelines the final validity conclusion is difficult, however, the case-studies did not falsify the usefulness of the guidelines. The guidelines appeared to be practical and to support in applying the RPM approach in industrial projects; however, it is recommended to apply this approach and its guidelines also in more projects to acquire experience in applying the approach and enhance the set of guidelines.

8.12.4 Overall validity conclusion

In chapter 2 it was concluded that the selected mix of case-studies in this research had the potential to be externally valid. In the case-studies it was shown that the approach could be applied in practice, that no major problems occurred with integrating the approach in development projects, and that the right product focus was established. The validation approach was to establish a 'context for falsification' (see chapter 2). Within this context all four case-studies did not fail. The three multiple, longitudinal and nested cases at Tokheim fully covering all three product types, and the final external validity check at Dräger, did not show failure of the RPM approach. This mix of case-studies and their positive results lead to the conclusion that the approach is externally valid for the embedded product domain.

8.13 Conclusion

In this chapter the results of applying product focused SPI in four industrial projects, using the RPM conceptual model and the guidelines, were presented. The three working areas appeared to be a good combination that covered the work to be done for product focused SPI. In all cases these three working areas could be recognised, although its specific application was shown to be quite flexible, and was customised to the specific situations. This flexibility is considered to be an important strength.

Application of the RPM conceptual model and the guidelines did not fail in four industrial projects. It can be concluded that both the model and the guidelines have passed the industrial tests. In each case-study a product focused SPI programme was established that focused its improvements to the product quality attributes that had the highest priority for improvement. The improvement goals were attained resulting in significant product quality improvements. Overall it became clear that the application of the RPM approach had quite

some impact on product quality, however it became also clear that a major success-factor for product quality in these projects was also the highly skilled and experienced developers.

The people in the development teams were enthusiastic about RPM application. They especially appreciated the specification of product quality, and the identification of product quality improvement areas. The analysis of measurements in the feedback sessions was also experienced as useful, and learning by the developers could be observed. Furthermore, the development teams stated that they appreciated the co-operation with the quality assurance department, which was shifted from 'procedure checking' to close participation in creating product quality.

Analysis of the costs and benefits of RPM application showed that the effort spent by the development teams is relatively low: less than 1% of their effort is spent on the improvement programme. The return on this investment was significant when considering the direct effects of the improvement programme, and the payback period was about three calendar months. These results show that RPM application is cost effective. The managers of the projects were also enthusiastic about the approach and claimed that the indirect benefits such as the establishment of a quality culture, and the group learning effects, were already worth the investment.

It was furthermore identified in the case-studies that learning is essential for motivating the development team. In chapter 6 it was identified that not only single loop learning, but also double loop learning should be established. The case-studies showed that double loop learning is not done automatically. Although the case-studies mainly focused on single loop learning, some double loop learning principles were applied; however, full application of double loop learning was not established in the case-studies. This is not so strange, as research identified that only few organisations succeed immediately in implementing double loop learning [Senge 1990] [Garvin 1993].

9. Conclusions and Recommendations

The research objective of this thesis was to develop a conceptual model for product focused process improvement for embedded product development. Furthermore a set of guidelines for practical usage of this model was developed. The conceptual model for product focused SPI was introduced in chapter 5 and enhanced in chapter 6 based on learning theory. Furthermore, practical guidelines were presented in chapter 7 for the conceptual model. These guidelines were used in several case-studies which were presented in chapter 8.

Conclusions on this research are provided in this chapter. The conclusions are subdivided in conclusions on product focused SPI in general, conclusions on the RPM conceptual model, and conclusions on the RPM guidelines. These conclusions are summarised in a section with final conclusions. Furthermore, this chapter contains a section that discusses opportunities for further research. This chapter ends with an epilogue that looks back on the research.

9.1 Conclusions regarding product focused SPI in general

The main idea of the product focused SPI approach presented in this thesis is that the software development processes are tuned in such a way that this process contributes most efficiently and effectively to the specific product quality objectives. 'Improving the process' is not an objective in itself, as it is for existing SPI approaches, but is considered as a means to achieve product quality. In that sense, the approach presented in this thesis is a significant improvement, as it expands the rigorous process focus of existing SPI approaches, such as the CMM, to address product quality as an explicit objective.

The research presented in this thesis was carried out to improve existing SPI approaches. Especially, when existing SPI approaches are used in the embedded software domain, some specific problems occur that need to be resolved. These problems are described in chapter 4, and based on a sub-set of these problems a list of criteria was presented to which product focused SPI should comply. A conceptual model was constructed that complies to these criteria and therefore should overcome the problems with current SPI approaches when applied to improve embedded product quality.

A focus on product quality is required

The approach presented in this thesis focuses on product quality. Improvements to the process are made based on their impact on those product quality attributes that have the highest priority for improvement.

Measurement is required

The approach presented in this thesis reserves one of its three main working areas for measurement. Measurement is one of the key practices carried out to improve product quality and software development processes. Measurement is used for two purposes: for the evaluation of product quality, and for the evaluation of process effects.

Analyse costs and benefits

The approach presented in this thesis is goal driven. The attainment of goals is a benefit that can be related directly to the use of SPI. This makes the benefits of SPI more visible. Furthermore, the use of measurement is an important source of information to visualise benefits; both direct and indirect, and measurement is also useful to identify the costs of SPI. It is highly recommended to integrate an analysis of costs and benefits into every investment in process improvement.

Focus improvements on projects

The approach presented in this thesis dedicates one of its three main working areas to the configuration of a product specific process, based on the specific product quality requirements. The approach presented in this thesis focuses on specific products and projects, which reduces the risk of a bureaucracy; however, the risk that process improvement results in an overkill of procedures can not be fully disabled.

Management commitment and continuation

The approach presented in this thesis still needs management support, but the commitment requirements have been partly transferred to lower management. Commitment from a project manager is sufficient to carry out product focused SPI, already establishing improvements within the project and improvements in product quality. The improvements therefore occur faster than with existing SPI approaches. Furthermore, continuation of the process improvements within the project is made more likely, since these improvements address the specific project objectives. Continuation outside the project can not be guaranteed, and still depends on management commitment; however, the cost effectiveness and the short payback time of the approach as identified in the case-studies, are results that stimulate continuation.

Process improvement is a learning process

The final conclusion on product focused SPI is that learning should be the main objective for process improvement. Software process improvement in general focuses on the continuous improvement of software processes. The impacts of changes to software processes are often unclear, although normally expectations are formulated. Unknown impacts can, however, be learned.

9.2 Conclusions regarding the RPM model

This research was started due to the lack of a product focus in existing SPI approaches. During this research a conceptual model was designed that incorporates a product focus into SPI. This model is based on multiple sources: existing SPI approaches, software engineering literature, learning theory and practical experiences from industrial companies. This conceptual model has a sound theoretical foundation and provides a clear direction for use of product focused SPI in practice.

Conceptual model is practically applicable

Application of this conceptual model in industrial case-studies was relatively easy and appeared to be useful. The RPM conceptual model presented in this thesis improved the product focus of SPI. Using this model in practice could be fully integrated with development projects. Improving the processes of these projects, and addressing the product quality targets of these projects, was established. As such, the RPM conceptual model was shown to be beneficial for industrial development projects, and its applicability proved to be feasible. The case-studies showed that a specific focus was put on those product areas of interest, and that effort was only spent on improving those processes that contributed to the product qualities of interest. As such, the presented approach can be used for industrial product focused SPI.

Three working areas for product focused SPI

Product focused SPI can be established under conditions such that three separate working areas are explicitly addressed:

- Requirements engineering, during which product quality targets are set
- Process engineering, during which the development process is tuned to address the product quality targets
- Measurement programme engineering, during which product quality and the effects of process actions are measured and these measurements are analysed

Requirements engineering

Requirements engineering is the main starting point for product focused SPI. It makes product quality explicit. This can be done by making the multiple stakeholder views on

quality explicit. Following such an approach it becomes feasible to address the multi-dimensional aspect of quality, which is not done in existing SPI approaches, nor is it sufficiently addressed by the existing software engineering literature. The main output of requirements engineering is a product quality specification, which describes the quality that is intended to be developed, in a standard quality terminology, and if possible in measurable terms. Requirements engineering is not a task that stands on its own: it is carried out in close interaction with process engineering.

Process engineering

The design of a product specific development process is supported by process engineering. In iteration with requirements engineering, a trade-off is made between 'what' is developed and 'how' it is developed. During these iterations product quality is balanced to criteria such as cost, duration, feasibility, functionality, experience, preferences, etc. During this negotiation process a decision is made on the product and on the process. The main output of process engineering is a development process model that describes the steps taken in the project to develop the required product quality. Monitoring the execution of this development process and evaluating its effects, is done by measurement programme engineering.

Measurement programme engineering

During measurement programme engineering the collection and analysis of software measurements is implemented. These measurements are closely related to the development process and the product. Evaluating the quality of the product, and evaluating the effectiveness of (parts of) the process, is supported by measurement programme engineering. It is important to focus measurements on the important areas. During this research this was done effectively by refining measurement goals to questions, and by refining these questions to metrics [Basili and Weiss 1984]. This approach supported the definition of the right metrics and, especially, the interpretation of the measurement results. Effects of certain process actions could be learned by a careful analysis of the measurements, and by discussing these measurements within the development teams. It was shown that measurement programme engineering is capable of providing feedback to requirements engineering on the attainment of product quality targets, and providing feedback to process engineering on the effects of the development process.

All three working areas need to be addressed

It was shown that these three working areas for product focused SPI appear to be a logical combination and all three should be addressed in industrial improvement programmes. There is a close relationship and interaction between these three areas, making product focused SPI feasible under the condition that these three working areas are an integral part of a software process improvement approach. If one of these three working areas is not

addressed during product focused SPI in practice, the work is likely to fail. Without requirements engineering there will be no product targets and in that way the product focus will be lacking. Without process engineering there will be no tuning of the development process towards the product quality targets and without measurement programme engineering there will be no validation whether process changes were actual improvements, nor is there an evaluation of the attainment of the product quality targets.

Specification of product quality is required

During the case-studies carried out for this research it became clear that to develop a quality product, it is necessary to specify the product quality needs. The importance of product quality specification is often underestimated, both in the literature and in practice. More attention is needed to the specification of product quality and the to impact of this specification on the development process. It is recommended that methods to improve software processes integrate this specification of product quality to make process improvement much more product driven. The approach presented in this thesis is a first step in this direction.

Make process product relationships explicit

A prerequisite for product focused SPI with the RPM model, is that the individual impacts of process actions on product quality are known. Tuning the development process to the product quality targets assumes that individual impacts and their interdependencies are known. This is not always the case. The results of literature research showed that process product relationships are rarely addressed in publications, and these expected relationships are almost never validated. Most people in software development projects have implicit models of these impacts of process actions on product quality. To carry out product focused SPI successfully, this implicit knowledge on impacts of process actions on product quality should be made explicit. Whenever these impacts are known, hypothetical or validated, tuning of the development process becomes possible. During the case-studies it was shown that making these process product relationships explicit and using them to tune development processes was indeed possible.

'Conformance to requirements' versus 'fitness for use'

The approach towards product quality presented in this thesis is based on *specification* of product quality, 'zero defects/conformance to requirements' [Crosby 1979]. Specifying product quality is by no means a guarantee that the final product will be perceived by users as a quality product. It is likely that some requirements will not be explicitly stated by the users, and therefore not be addressed. Furthermore it is sometimes impossible to find users to elicitate requirements from, or if the users are found they do not know in advance how they will use the product. Especially in situations were innovative products are built with which no user has any experience, stating product quality requirements is difficult. This

indicates that it is likely that there will be a gap between ‘what is needed’ and ‘what is specified’ in the product quality specification. Bridging this gap should be done by other means than applying the RPM conceptual model. For example by including a closed-loop principle by which experiences of users from application or service experiences are fed back to the designers of a product [Genuchten 1991] [Brombacher et al. 1996], and the product is evolved and improved in a step by step manner.

Improved communication of product quality issues

During the case-studies it became clear that a benefit of the approach was improved communication of improvement areas, and a better follow-up on these issues. The approach makes product quality explicit and addresses the multiple viewpoints of all the different stakeholders. In this way, the approach as such is also an approach that promotes communication of quality needs. An important aspect found during the case-studies was that many problems or needs for product improvement that were stated by stakeholders, had already been stated by them before. Due to unknown reasons these issues were rarely addressed; however, due to the structure used by the RPM approach, these issues were now raised in a way that was perceived to be more objective, and they were indeed tackled. Remarkably some of them were the same issues that were stated before. The only difference in the way of communicating was in the structure of the approach, not in the message.

Result driven attitude of development team members

Using RPM in the case-studies showed an increase in goal-oriented behaviour of development project people: they regularly asked themselves: If I change this in the process, how will it impact the product? They were making their hypothetical process product relationship models explicit, and were evaluating and measuring them critically. Furthermore, the project people were supported in making the goals explicit and the attainment of these goals was actually measured. This goal-orientation was perceived as a positive point of the approach, since the improvement efforts were only spent on the product quality aspects of direct interest.

Integration of SPI into development work

An important aspect for establishing product focused SPI in practice is the integration of the improvement work with the software development work. During the case-studies it became clear that the RPM conceptual model follows a similar order to development projects. As such this integration could be established. Requirements engineering was carried out during the software requirements phase of the development project. Process engineering was carried out during project planning and project re-planning. Measurement programme engineering was carried out during the software design, implementation and testing work. Feedback on the measurements was given, during the course of the project,

based on which conclusions were drawn and actions were taken. During the case-studies it was shown that the RPM conceptual model was successfully integrated into software development projects.

9.3 Conclusions regarding the RPM guidelines

The product focused SPI approach presented in this thesis is facilitated by guidelines for requirements engineering, process engineering and measurement programme engineering. These guidelines are presented in chapter 7, and were used in the case-studies presented in chapter 8.

The guidelines mainly address ‘what’ needs to be done in the three working areas, and attempt not to prescribe ‘how’ this should be done. This focus was selected deliberately to leave sufficient space and flexibility to customise the approach to the specific organisations and projects. The guidelines address each working area separately. The guidelines are based on several sources: SPI literature, software engineering literature, learning theory, and practical experiences in industrial improvement programmes. This total mixture creates a set of guidelines that is both theoretically sound and practically feasible.

The guidelines were used in the industrial case-studies of this research. The guidelines support in making the RPM conceptual model more operational, and support in the identification of what needs to be done. Discussing the experiences with each guideline individually is not done in this thesis. The main reason for this is that the guidelines were not tested separately but used together; therefore it is difficult to identify the individual impact of each guideline. The experiences in the case-studies did, however, not result in a change of the guidelines: it appeared that the guidelines fulfil their purpose.

9.4 Final conclusions

In this research an attempt was made to let product quality be the direct result of embedded software process improvement programmes. The theoretical findings, ideas and practical experiences are summarised in three final conclusions.

Creating product quality is a negotiation process

In this research it was shown that making a quality product means: involving stakeholders and letting these stakeholders express their product quality wishes. Based on these explicit demands for product quality, a negotiation process is started during which decisions are taken on product quality. A trade-off on quality, cost, risk and duration among all product stakeholders is made during this negotiation process. Furthermore, the feasibility of the

process to create that product quality is addressed. A development process is designed that is expected to create the specified level of quality.

This conclusion might sound a bit trivial, at first sight; however, this research has shown that such a negotiation process is complex and difficult when carried out in practice. Some solutions to support this negotiation process have been proposed and validated during the case-studies. Based on these findings it can be expressed that the problems with poor quality of embedded products are probably mainly caused by an inadequate negotiation process on product quality. Companies that develop embedded products and give priority to quality should support this by installing an explicit product quality negotiation process between the product stakeholders.

Working on product quality requires support from (simple) instruments

Product quality is largely influenced by the negotiation process among stakeholders, who are normally not experts in developing embedded products. As a consequence, instruments are required to support these stakeholders in expressing and discussing product quality issues. These instruments do not have to be complex. Simple instruments that every stakeholder understands and can apply are sufficient. This research has presented such a set of (simple) instruments.

Through usage of the approach and its instruments, as presented in this thesis, it becomes possible to address the difficult topic of embedded product quality in a practical way. Stakeholders are supported in making their wishes explicit, and these wishes are translated to a product quality specification that is useful for the product developers. These developers are furthermore supported in designing a development process that fits to the product quality specification, and they are provided with feedback on product quality early in the development process.

Product quality is created in projects within a context

The case-studies of this research underline that product quality is created in projects. Enabling organisations to make quality products should therefore be an activity that focuses on the single projects that create the product. Existing SPI approaches mainly focus on organisational processes, which cause the ‘improvements’ to be general and not project specific. This in itself is not wrong, but it is simply not enough. A focus on organisational process improvement is an important precondition for product quality improvement, but a focus to the projects should be established as well. The focus of existing SPI approaches is mainly to organisations as a whole and therefore not optimal for product quality; it is recommended that SPI programmes also include much more project specific tasks.

Development projects should be supported in the right way to make product quality. Supporting such projects by specifying product quality and designing a product specific process are sound solutions to overcome some of the basic problems in creating product quality; however, one of the major contributions to the creation of product quality is to establish project specific feedback. Through the collection of project specific measurements, it is possible to provide feedback to the project team on product quality and process performance. This information can be used during the project to take corrective action and to learn the effects of the work. In the case-studies, in which the RPM approach was used in relatively small projects, it was identified that these project specific measurements contributed largely to the success of RPM application in practice.

9.5 Recommendations for further research

Although, the approach presented in this thesis fulfils the requirements for product focused SPI as set in chapter 4, it is still possible to identify issues on which some further research is required.

Firstly, the RPM conceptual model is based on the assumption that external influences exist on software processes from aspects such as politics, people factors, historical norms, cultural differences, etc. do not interfere with the approach. Such external influences are present in practice and their effects may not be underestimated. Further research is required on the impacts of these external influences.

Secondly, in this thesis it was stated that double loop learning processes are required within industrial development projects; however, during the case-studies it appeared to be difficult to install such double loop learning. This could have been foreseen, because learning theory already indicates the low number of companies that succeed in double loop learning. It is recommended to investigate the ways in which double loop learning can be successfully implemented in the embedded product industry, and to investigate how to establish an ‘embedded product development learning organisation’.

Thirdly, the presented model might require in some cases substantial effort as an initial investment. In the case-studies it was shown that applying the approach in development projects does not cost much effort; however, preparing the organisation to apply such an approach also requires some effort. How much this will cost has not been investigated in this research, but it may be substantial as it might involve changing the culture of an organisation. Further research is necessary to learn the effort required to install the RPM approach into an organisation, and to identify the critical success factors for undergoing this change.

Fourthly, the application of the RPM model was limited to the embedded product domain; however, other domains of software development also have clear product objectives.

Application of the RPM conceptual model and its guidelines in such other domains should also be investigated, and research is required to identify the 'fitness for use' of this approach in other domains.

Fifthly, it has been stated that there is a clear relationship between a good architecture and quality. How to address this relationship is, however, not very clear. Further research is necessary on the direct consequences of certain architectural decisions on product quality. Furthermore, research is required to support the design of an architecture; on the decomposition of software into components, subsystems, modules, etc. How to make such a decomposition, which factors in this decomposition process impact quality, how to use product quality requirements in the architectural design, are relevant research questions that deserve to be answered.

Finally, in this research it was identified that the number of process product relationships that are present for embedded product development, are rarely investigated or validated. The lack of valid models of relationships between process actions and product quality make it difficult for the embedded product industry to estimate the level of quality that they are creating with their processes. It is recommended to continue the search for such valid relationships. Furthermore, it is recommended to researchers that publish new methods, techniques and tools for embedded product quality, to first measure the impact of their new approach on product quality. Papers on new approaches should be reviewed critically, and when no measurements have been taken on effectiveness and efficiency, the papers should be rejected. Without establishing such norms for sound scientific results in the software engineering community, it is likely to remain a discipline that is highly 'hype-driven' and that lacks knowledge on the impacts of its own approaches.

9.6 Epilogue

The objective of this research was to develop a conceptual model for product focused process improvement and a set of accompanying guidelines for practice. Hopefully, the results presented will help and provide new insights for everyone working on, product focused, SPI. The approach presented in this thesis is not a fully ready 'cookbook' as yet. It does not prescribe all the ingredients, steps to take, or of timing issues involved for performing product focused SPI. A direction is, however, pointed out, practical guidelines are provided together with examples from successful cases in industry. Hopefully, these concepts and guidelines will be used in practice and enhanced over time, when working on product focused SPI.

Appendix A: RPM Tool Design

The RPM experience base prototype tool is presented in this appendix. This tool was developed within this research and was used during the case-studies for the selection of process improvements and for making the estimation of product quality when using a certain process model. The tool was developed in MS-Access 7.0 (for Windows 95), which is a relational database management system.

System Overview

Context Diagram

The context diagram represents the interface of the system to the 'outside world'. It shows the main process, the external processes and the system stores. The main process is to store the knowledge of the effect of the process actions on the quality attributes and the selection of specific process actions for specific projects according to the project quality profile of the project.

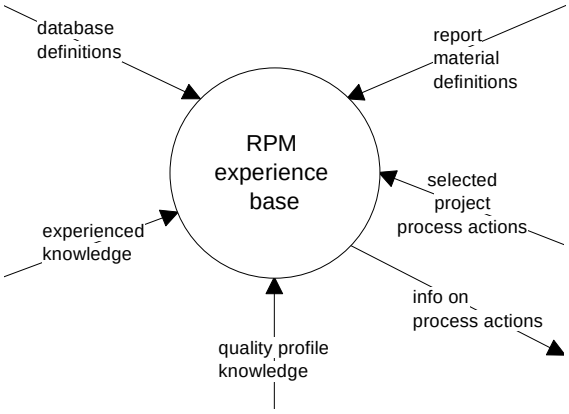


Figure 1: Experience base context diagram

Entity Relationship Diagram

This section presents the highest level entity-relationship diagram of the data processed in the system. The entity-relationship diagram consists of different parts: a generic and a project specific part.

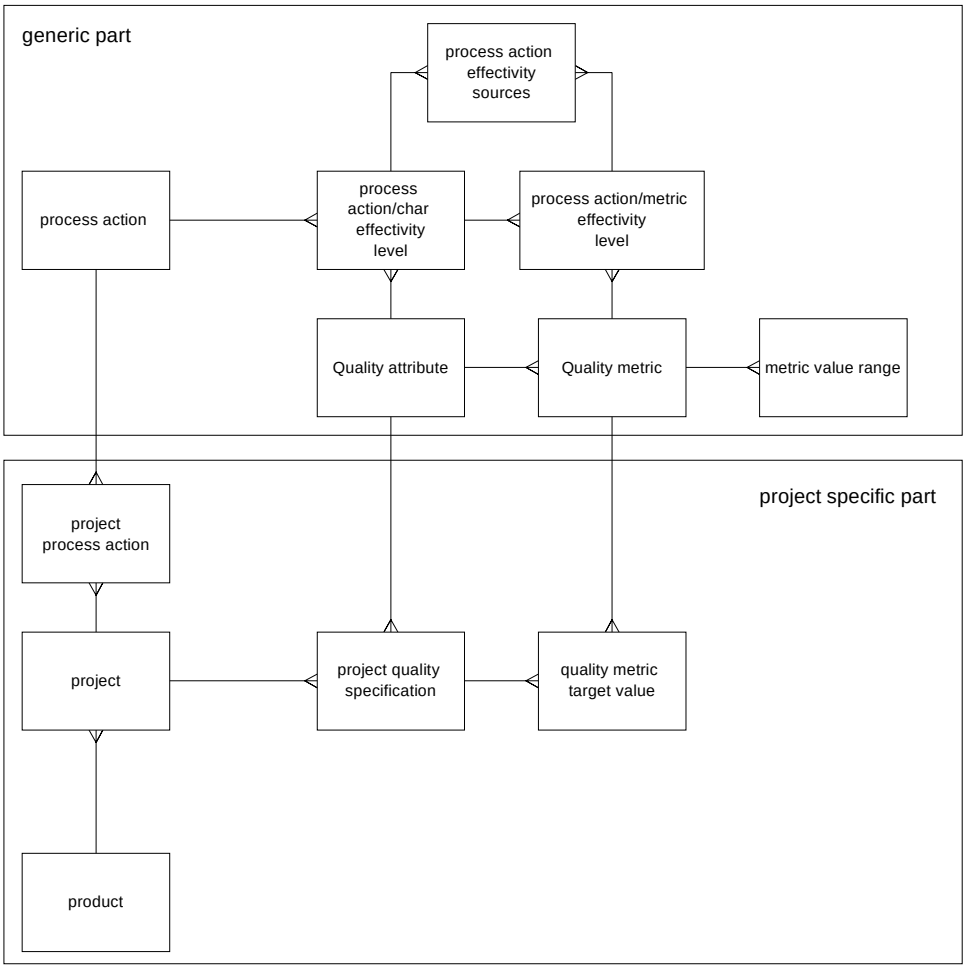


Figure 2: RPM experience base entity-relationship diagram

The generic part supports the storage and updating of information which can be used and re-used for specific projects. The generic part is about the process actions that can be taken for the projects and the knowledge about the effectivity of these process actions.

The project specific part maintains information about projects and the selected process actions for these projects. This part also contains the selected quality for the projects.

The entity-relationship diagram of Figure 2 contains several entities:

- Product: the embedded product that is being developed
- Project: the set of resources, objectives and time that intend to develop a product
- Project quality specification: the quality specification for the product of a specific project
- Quality metric target value: the goal for a metric to be realised in the project

- Quality metric: the data collection procedure for a certain measurement of a certain aspect of a product quality attribute
- Quality attribute: A sub-division of product quality, e.g. Functionality or Usability
- Metric value range: the range along which a metric can have its values
- Project process action: used to establish an n:m relation between a project and a process action
- Process action: the action taken in the development process to achieve an explicit effect on product quality
- Process action characteristic effectivity level: the relationship between a process action and its impact on a product quality attribute
- Process action effectivity metric level: used to establish an n:m relation between Quality metric and Process action characteristic effectivity level
- Process action effectivity sources: the literature or practical source on which the relationship between process action and product quality attribute is based

Functional Specification

The process actions and quality attributes system consists of separate system functions

- storage of project information
- storage of quality attributes and metrics
- storage of process action information
- storage of the effectivity of process actions on quality attributes
- analysis of the resource data
- preparation of report materials

In the following sections the purpose of each system function, as well as its input, output and behaviour will be presented.

Storage of project information

The purpose of the system function ‘storage of project information’ is to store and maintain data about development projects. The reason for maintaining this information is enabling the assignment of process actions to a specific project. The input of this function are the characteristics of certain projects. The function has no output. The function will enter the characteristics of the projects in the database.

Storage of quality attributes and metrics

The purpose of the system function ‘storage of quality attributes and quality metrics’ is to store and maintain data about the quality attributes and quality metrics which are used and re-used for the requirements of a project. The inputs of this function are the quality attributes and metrics. The function has no output. The function will enter the quality attributes and metrics in the database.

Storage of process action information

The purpose of the system function ‘storage of process action information’ is to store and maintain data about the process actions that can or have to be taken to improve the quality attributes of a certain product within a certain project. The inputs of this function are the details of process actions. The function has no output. The function will enter the details of a process action in the database.

Storage of the effectivity of process actions on quality attributes

The purpose of the system function ‘storage of the effectivity of process actions on quality attributes’ is to store the estimate effectivity of a certain process action on a certain quality attribute. This data will later be used to select process actions for certain projects. As input this function has the knowledge gained about the process actions. The function will store this data in the database. This data is maintained and updated if necessary

Select project process actions

The purpose of the system function ‘select project process actions’ is to give the user a possibility to select certain process actions for certain projects. The objective is to give the user an interactive way to select the project process actions. In a way the user has to play with the process actions and see their results on the selected quality profile with the quality attributes to select them.

The function must work in such a way that the selection of process actions is easy and based on the results of the process action known in the database.

Preparation of report materials

The purpose of the system function ‘preparation of report materials’ is to provide the possibility to output the information that is stored in the database in reports enabling users to understand the data.

References

Agarwal, R., Krudys, G., Tanniru, M., ‘Infusing learning into the information systems organisation’, European Journal of Information Systems, No. 6, pp. 25-40, 1997.

Aken, J.E. van, ‘Management science as design science: the regulative and the reflective cycle’ (In Dutch), Bedrijfskunde, 66 (1), pp 16-26, 1994.

Albrecht, A.J., Gaffney, J.E., ‘Software function, source lines of code, and development effort prediction: A software science validation’, IEEE Transactions on software engineering, Vol. 9, No. 6, pp. 639-648, November 1983.

Alford, M., Lawson, J., ‘Software Requirements Engineering Methodology’, RADC-TR-79-168, US Air Force Rome Air Development Center, June 1979.

Amabile, T.M., ‘How to kill creativity’, Harvard Business Review, September/October, pp. 77-87, 1998.

Analoui, F., Training and transfer of learning, Avebury, 1993.

Anderson, J.R., Cognitive psychology and its implications, Freeman and company, 1990.

Argyris, C, Schön, D.A., Organizational learning: a theory of action perspective, Addison-Wesley, 1978.

Argyris, C., Reasoning, Learning and Action, Jossey-Bass Publishers, 1982.

Argyris, C., On Organizational Learning, Blackwell Publishers, 1993.

Ayas, K., Design for learning for innovation, Eburon Publishers, Delft - The Netherlands, 1997.

Bach, J., ‘The immaturity of the CMM’, The American Programmer, September 1994.

Basili, V.R., Weiss, D.M., ‘A methodology for collecting valid software engineering data’, IEEE Transactions on Software Engineering, SE-10(6):728 - 738, November 1984.

Basili, V.R., Selby, R.W., Huthchens, D.H., ‘Experimentation in Software Engineering’, IEEE Transactions on Software Engineering, Vol. 12, No. 7, pp. 733-743, July 1986.

Basili, V.R., Rombach, H.D., ‘The TAME project: Towards Improvement Oriented Software Environments’, IEEE Transactions on Software Engineering, Vol. 14, No. 6, pp. 758-773, June 1988.

Basili, V.R., ‘The experimental paradigm in software engineering’, In: Experimental Software Engineering Issues, Springer Verlag, LNCS#706, 1993.

Basili, V.R., Caldiera, C., Rombach, H.D., ‘Experience Factory’, Encyclopaedia of Software Engineering (Marciniak, J.J., editor), Volume 1, John Wiley and Sons, pp. 469 - 476, 1994a.

Basili, V.R., Caldiera, C., Rombach, H.D., ‘Goal/Question/Metric Paradigm’, Encyclopaedia of Software Engineering (Marciniak, J.J., editor), Volume 1, pp. 528-532, John Wiley and Sons, 1994b.

Bass, L., Kazman, R., ‘Architecture-based development’, SEI Technical Report, CMU/SEI-99-TR-007, 1999.

Bemelmans, T.M.A., Bestuurlijke informatiesystemen en automatisering (In Dutch), Kluwer Bedrijfsinformatie, 1998.

Benyon, D., Skidmore, S., ‘Towards a tool kit for the system analyst’, The computer journal, pp 2-7, 1987.

Bicego, A., Khurana, M., Kuvaja, P., 'Bootstrap 3.0 - Software Process Assessment Methodology', Proceedings of the SQM'98, 1998.

Bicego, A., Derks, P., Kuvaja, P., 'Product focused process improvement: Experiences of applying the PROFES improvement methodology at Dräger', Proceedings of the 1999 EuroMicro Conference, 1999.

Birk, A., Järvinen, J., Solingen, R. van, 'A validation approach for product-focused process improvement', Proceedings of the 1st PROFES conference, Oulu, Finland, June 22-24, VTT Symposium series, 1999.

Boehm, B.W. Software engineering economics, Prentice-Hall International, New Jersey, 1981.

Boehm, B.W., 'A spiral model of software development and enhancement', In: Software Engineering Project Management, editor: R.H.Thayer, pp 128-142, IEEE CS, 1987.

Briand, L.C., Differding, C.M., Rombach, H.D., 'Practical guidelines for measurement based process improvement', ISERN Report 96-05, 1996.

Brinkemper, S., 'Method engineering: engineering of information systems development methods and tools', Information and Software Technology 38, Elsevier, pp. 275-280, 1996.

Brombacher, A.C., Reliability by design, John Wiley and Sons, 1992.

Brombacher, A.C., et al., 'Systematic failures in safety systems: How to analyse; how to optimise', Proceedings of the ISA Chicago conference, October 1996.

Brombacher, A.C., Steinz, H.C., Volman, H.P.J., 'Safety and reliability assessment of products and organisations', ISA Expo98, TECHNOLOGY UPDATE, Volume 2, The International Conference and Exposition for Advancing Measurement and Control Technologies, Products and Services, Houston, Texas, October 19 -22, 1998.

Brooks, F.P., The mythical man month: essays on software engineering, Addison-Wesley, 1975.

Bush, M.E., Fenton, N.E., 'Software Measurement: A conceptual framework', Journal of Systems and Software, Vol. 12, pp. 223-231, 1990.

Card, D., Glass, R., Measuring software design quality, Prentice-Hall, 1990.

Cavano, J.P., McCall, J.A., 'A framework for the measurement of software quality', Proceedings of the software quality and assurance workshop, 1978.

Clements, P., Bass, L., Kazman, R., Abowd, G., 'Predicting software quality by architecture level evaluation', Proceedings of the Fifth International Conference on Software Quality, Austin, Texas, October, 1995.

Clements, P.C., Northtrop, L.M., 'Software Architecture: An executive overview', SEI Technical Report, CMU/SEI-96-TR-003, 1996.

Cook, C.R., Roesch, A., 'Real-time software metrics', Journal of Systems and Software, Vol. 24, pp. 223-237, 1994.

Cook, T.D., Campbell, D.T., Quasi-experimentation, Rand McNally, Chicago, 1979.

Crosby, B.P., Quality is free, McGrawHill, New York, 1979.

Daskalantonakis, M.K., 'Achieving higher SEI levels', IEEE Software, July 1994.

Davis, A.M., Bersoff, E.H., Comer, E.R., 'A strategy for comparing alternative software development life cycle models', IEEE Transactions on Software Engineering, Vol. 14, No. 10, October 1988, pp 1453-1461, IEEE CS, 1988.

Davis, A., Dandashi, F., Reynolds, P., 'Identifying and Measuring Quality in software requirements specification', Proceedings of the first international software metrics symposium, IEEE CS, pp 141-152, 1993.

Davis, G.B., 'Strategies for information requirements determination', IBM Systems Journal, Vol. 21, No. 1, 1982.

Davis, G.B., Olson, M.H., Management Information Systems: Conceptual foundations, structure and development, McGraw-Hill, 1985.

DeMarco, T., Controlling software projects, Yourdon Press, New York, 1982.

Deming, W.E., Out of the crisis, MIT Center for advanced engineering study, 1986.

Dion, R., 'Process Improvement and the corporate balance sheet', IEEE Software, July 1993.

Downes, V.A., Goldsack, S.J., Programming embedded systems with ADA, Prentice-Hall International, 1982.

Entwistle, N., Styles of Learning and Teaching, John Wiley & Sons, 1981.

Erens, F.J., The Synthesis of Variety: Developing product families, KPMG, 1996.

Faulk, S.R., 'Software Requirements: A Tutorial', Software Engineering Editors: Dorfman and Thayer, IEEE pp 82-103, 1996.

Fenton, N.E., Pfleeger, S.L., Software Metrics: A rigorous and practical approach, Thomson Computer Press, 1996.

Finkelstein, A., Kramer, J., Nuseibeh, B., Goedicke, M., 'Viewpoints: a framework for integrating multiple perspectives in system development', International journal on software engineering and knowledge engineering, pp. 31-58, February 1992.

Gal, R., Genuchten, M., 'Release the embedded software: the electronics industry in transition', International Journal of Technology Management, Vol. 12, No. 1, 1996.

Galbraith, J., Organization design, Addison-Wesley, Reading (Mass.), 1977.

Garlan, D., Perry, D., Guest editorial opening statement, IEEE Transactions on Software Engineering, April 1995.

Garvin, D.A., 'What does product quality really mean?', Sloan Management Review, Vol.26, nr. 1, 1984.

Garvin, D.A., Managing Quality, The Free Press, MacMillan Inc., 1988.

Garvin, D.A., 'Building a learning organisation', Harvard Business Review, July-August, pp. 81-91, 1993.

Gentleman, W.M., 'If software quality is a perception, how do we measure it?', NRC (nat. research counsel of Canada) number 40149, pp. 336-345, 1994.

Genuchten, M. van, Towards a software factory, Ph.D. Thesis Eindhoven University of Technology, 1991.

Gibbs, W.W., 'Software's Chronic Crisis', Scientific American, pp. 86-95, September 1994.

Gilb, T., Principles of software engineering management, Addison-Wesley, 1994.

Gillies, A.C., Software quality: Theory and management, Chapman & Hall, 1992.

Glass, R.L., Software Creativity, Prentic Hall, 1995.

Goguen, J.A., Linde, C., 'Techniques for Requirements Elicitation', Proceedings of the International Symposium on Requirements Engineering, pp. 152-164, IEEE, 1993.

Goldenson, D.R., Herbsleb, J.D., 'After the appraisal: A systematic survey of process improvement, its benefits and factors that influence success', SEI-95-TR-009, Software Engineering Institute, 1995.

Gomaa, H., 'The impact of prototyping on software system engineering', In: System and Software Requirements Engineering, Dofrman and Thayer (eds.), IEEE, pp 543-552, 1990.

Goodman, P., Practical implementation of software metrics, McGraw-Hill Publishers, London, 1993.

Grady, R.B., Caswell, D.L., Software Metrics: Establishing a company-wide program, Prentice-Hall, 1987.

Grady, R.B., Practical software metrics for project management and process improvement, Prentice-Hall, 1992.

Grady, R.B., 'Successfully applying software metrics', IEEE Computer, pp 18-25, September 1994.

Hall, T., Fenton, N., 'Implementing software metrics - the critical success factors', Software Quality Journal, No. 3, pp. 195-208, 1994.

Halstead, M., Elements of Software Science, Elsevier, 1977.

Hamann, D., Järvinen, J., Birk, A., Pfahl, D., 'A product process dependency definition method', Proceedings of the Euromicro 98 workshop on software process and product improvement, Västerås, Sweden, August 1998a.

Hamann, D., Järvinen, J., Oivo, M., Pfahl, D., 'Experience with explicit modelling of relationships between process and product quality', Proceedings of the 4th European Software Process Improvement Conference, Monte Carlo, December 1998b.

Hanani, M.Z., Shoval, P., 'A combined methodology for information systems analysis and design based on ISAC and NIAM', Information Systems 11, pp 245-253, 1986.

Hatley, D., Pirbhai, I., Strategies for real-time specification, Dorset House, New York, 1987.

Hatton, L., 'Automated incremental improvement of software product quality: a case history', In: Software Quality: Assurance and Measurement A Worldwide perspective, editors: Fenton, N., Whitty, R., Iizuka, Y., Thomson Computer Press, 1995.

Hayes, W., Zubrow, D., 'Moving on up: Data and experience doing CMM-based process improvement', CMU/SEI-95-TR-008, 1995.

Heemstra, F.J., How expensive is software (In Dutch), Kluwer, 1989.

Heemstra, F.J., Kusters, R.J., Trienekens, J.J.M., 'From quality requirement factor to quality factor: an end-user based method', Proceedings of the 6th ESCOM conference, pp. 18.1-18-19, May 1995.

Hetzel, B., 'The sorry state of software practice measurement and evaluation', In: Software Quality: Assurance and Measurement A Worldwide perspective, editors: Fenton, N., Whitty, R., Iizuka, Y., Thomson Computer Press, 1995.

Huber, G.P., 'Organisational learning: the contributing processes and the literatures', Organization Science, Vol. 2, No. 1, pp. 88-115, February 1991.

Humphrey, W. S., Managing the software process, SEI series in software engineering, Addison-Wesley, 1989.

Humphrey, W.S., Snyder, T.R., Willis, R.R., 'Software process improvement at Hughes aircraft', IEEE Software, 8(4), pp. 11-23, July 1991.

Hutjes, J.M., Buuren, J.A. van, The case-study: Strategies for qualitative research (In Dutch), Boom, 1992.

IEEE Standards Collection, Standard for requirements specification, 1994.

ISO 9000-3, Quality Management and Quality Assurance Standards - Part 3: Guidelines for the application of ISO 9001 to develop, supply install and maintain software, 1997.

ISO 9001, Quality Systems - Model for Quality Assurance in design, development, production, installation and servicing, International Standards Organisation, 1994.

ISO 9126, Information Technology, Software product evaluation: Quality characteristics and guidelines for their use, International Organisation for Standardisation, 1991.

ISO 9126, Information Technology, Software product Quality - Part I: Quality model, International Organisation for Standardisation, FCD 1998.

ISO 14598, Software Product Evaluation, International Organisation for Standardisation, 1996.

ISO 15504, Information Technology - Software Process Assessment - Part 2: A Reference Model for Process and Product Capability, Technical Report Type 2, International Organisation for Standardisation, 1998.

ITIL, CCTA, Information Technology Infrastructure Library, HMSO Publishing Centre, London, 1987.

Jelinek, M., Institutionalizing Innovation, Praeger, 1979.

Juran, J.M., Gryna, F.M., Juran's quality control handbook, McGraw-Hill, 1988.

Kan, S.H., Basili, V.R., Shapiro, L.N., 'Software Quality: An overview from the perspective of total quality management', IBM Systems Journal, Vol. 33, No. 1, pp. 4-19, 1994.

Karjalainen, J., Makarainen, M., Komi-Sirvio, S., Seppanen, V., 'Practical Process Improvement for Embedded Real-Time Software', Quality Engineering, vol. 8, no. 4, pp. 565-573, 1996.

Kidder, L., Judd, C.M., Research methods in social relations, Holt, Rinehart & Winston, 1986.

Kolb, D.A., Experiential Learning, Prentice Hall, 1984.

Kotonya, G., Somerville, I., 'Requirements engineering with viewpoints', Software Engineering Journal, pp 5-18, January 1996.

Kündig, A.T., 'A note on the meaning of embedded systems', In: Embedded Systems: New approaches to their forma description and design, Lecture notes in computer science #284, Springer-Verlag, 1986.

Kusters, R.J., Solingen, R. van, Trienekens, J.J.M., 'User-perceptions of embedded software quality', Proceedings of the Eighth international workshop on software technology and engineering practice (STEP'97), pp. 184-197, London, July 14-18, 1997.

Kusters, Solingen, Trienekens, 'Identifying embedded software quality: two approaches', Quality and Reliability Engineering International, Wiley, Nov/Dec 1999.

Kuvaja, P., Bicego, A., 'BOOTSTRAP: a European assessment methodology', Software Quality Journal, 3(3), pp. 117-128, 1994.

Lammers, C.J., et al., Comparing Organisations (In Dutch), Het Spectrum, 1997.

Latum, F. van, Oivo, M., Hoisl, B., Ruhe, G., 'No improvement without feedback: experiences from goal oriented measurement at Schlumberger', Proceedings of the 5th European Workshop on Software Process Technology (EWSPT96), Nancy, France, Lecture Notes in Computer Science #1149, Springer Verlag, pp. 167-182, October 1996.

Latum, Solingen, Oivo, Rombach, Hoisl and Ruhe, 'Adopting GQM-based measurement in an industrial environment', IEEE Software, Jan/Feb 1998.

Leeuw, A.C.J. de, Organisations, management, analysis, design and change: a systems perspective (in Dutch), van Gorcum, 1986.

Leite, J.C.S.P., 'A survey on requirements analysis', Department of information and computer science, University of California, 1987.

Malouin, J.L., Landry, M., 'The mirage of universal methods in systems design', Journal of applied system analysis, pp 47-62, 1983.

March, J.G., 'Exploration and exploitation in organizational learning', Organization Science, Vol. 2, No. 1, pp. 71-87, February 1991.

McGarry, F., Pajerski, R., Page, G., Waligora, S., Basili, V., Zelkowitz, M., 'Software Process Improvement in the NASA Software Engineering Laboratory', Technical Report, CMU/SEI-94-TR-22, Software Engineering Institute, 1994.

Mintzberg, H., 'Structure in fives: designing effective organizations', Prentice-Hall International Editions, Englewood Cliffs, New Jersey, ISBN 0-13-854191-4, 1983.

Möller, K.H., Paulisch, D.J., Software Metrics: A practitioner's guide to improved product development, Prentice-Hall, 1993.

Nevis, E., DiBella, A., Gould, J., 'Understanding organisations as learning systems', Sloan Management Review, Winter 1995.

Nissen, H., Jeusfeld, M., Jarke, M., Zemanek, G., Huber, H., 'Technology to Manage Multiple Requirements Perspectives', IEEE Software, March 1996.

Nonaka, I., Takeuchi, H., The knowledge-Creating Company, Oxford University Press, New York, 1995.

Oivo, M., Quantitative management of software production using object-oriented models, VTT Publications, 1994.

Oivo, M., Bicego, A., Kuvaja, P., Pfahl, D., Solingen, R. van, The PROFES methodology book and User Manual, Http://www.ele.vtt.fi/profes/, 1999.

Paulisch, D.J., Carleton, A.D., 'Case Studies of software process improvement measurement', IEEE Computer, pp. 50-57, September 1994.

Paulk, M.C., Curtis, B., Chrissis, M.B., Weber, C.V., Capability Maturity Model for Software, Version 1.1. SEI-CMU-93-TR-24, Software Engineering Institute, 1993.

Paulk, M.C., Curtis, B., Chrissis, M.B., Weber, C.V., Key practices of the Capability Maturity Model, Version 1.1. SEI-CMU-93-TR-25, Software Engineering Institute, 1993.

Pfleeger, S.F., Software Engineering: the production of quality software, McMillan Publishing, New York, 1991.

Pfleeger, S.L., Rombach, H.D., 'Measurement based process improvement', IEEE Software, pp. 9-11, July 1994.

Popper, K.R., The logic of scientific discovery, Hutchinson, London, 1968.

Punter, T., Lemmen, K., 'The MEMA model: towards a new approach for method engineering', Journal of Information and Software Technology 38, pp 295-305, 1996.

Putnam, L.H., Myers, W., Measures for Excellence: Reliable software on time, within budget, Yourdon press - Prentice Hall, New Jersey, 1992.

Quint, Specifying Software Quality (In Dutch), Kluwer Bedrijfswetenschappen, 1991.

Rae, H., Hausen, Robert, Ph., Software Product Evaluation, McGraw-Hill, 1995.

Rees, J.R. van, 'The method does not work' (In Dutch), Informatie 24 pp 81-93, 1982.

Renkema, T.J.W., Investing in the information infrastructure: guidelines for decision making in organisations (In Dutch), Ph.D. thesis Einhoven University of Technology, The Netherlands, 1996.

Robert, Ph. SCOPE: Final report, Scope Consortium, 1994.

Rooijmans, J., Aerts, H., Genuchten, M. van, 'Software Quality in Consumer Electronic Products', IEEE Software, pp. 55-64, January 1996.

Royce, W.W., 'Managing the development of large software systems: concepts and techniques', Proceedings of the WESCON, August 1970.

Rumbaugh, J., 'Getting started: using use cases to capture requirements', Journal of object oriented programming, pp 8-12, September 1994.

Sassenburg,H., Matser, G., Kazil, P., 'Software Process Improvement: Why and when?' (In Dutch), Informatie, 38, July/August, 1996.

Senge, P.M., The fifth discipline: The art and practice of the learning organisation, New York, Doubleday, 1990.

Senge, P.M., 'The leader's new work: Building learning organisations', Sloan Management Review, pp. 7-23, Fall 1990.

Senge, P.M., Roberts, C., Ross, R.B., Smith, B.J., Kleiner, A., The fifth discipline fieldbook: Strategies and tools for building a learning organization, NB-publishing, London, 1994.

Shaw, M., 'Prospects for an engineering discipline of software', IEEE Software, November 1990.

Shepperd, M.J., Software Engineering Metrics, Volume 1: Measures and Validations, McGraw-Hill, 1993.

Shepperd, M.J., Ince, D., Derivation and Validation of Software Metrics, Clarendon Press, 1993.

Siddiqi, J., Shekaran, M.C., 'Requirements engineering: the emerging wisdom', IEEE Software, March 1996.

Slooten, C. van, Situated methods for systems development, Ph.D. thesis, Technical University Twente, The Netherlands, 1995.

Soerjoesing, S.P., Product-Process Dependency Modelling: An investigation between software engineering practices and software product quality, Delft University of Technology and Schlumberger RPS, 1998.

Soerjoesing, S.P., Software Quality Improvement through Product-Process Dependency Modelling: Experiences with product-process dependency models embedded in the PROFES approach to software process improvement as applied in Tokheim, Delft University of Technology and Tokheim, 1999.

Sol, E.J., 'Embedded software: vision, paradigm shifts, figures, and consequences for companies in the electronic business, Xootic Magazine, Eindhoven University, March 1995.

Solingen, R. van, Rodenbach, E., 'Embedded software for petrol stations' (In Dutch), Maandblad Informatie, pp. 36-43, July/August, 1996.

Solingen, R. van, Uijtregt, S. van, 'Field-Testing of Embedded Software Products: Handling Paradoxes in Practice', Proceedings of the 3rd ENCRESS Conference, Chapman and Hall, ISBN 0412802805, 1997.

Solingen, R. van , Berghout, E., Kooiman, E., 'Assessing feedback of measurement data: Relating Schlumberger RPS practice to learning theory', Proceedings of the 4th International Software Metrics Symposium (Metrics'97), Albuquerque, November 5-7, IEEE CS, pp. 152-164, 1997.

Solingen, R. van, Berghout, E.W., Latum, F. van, 'Interrupts: Just a minute never is', IEEE Software, September/October 1998.

Solingen, R. van, Berghout, E.W., 'The Goal/Question/Metric Method: A practical guide for quality improvement of software development', http://www.gqm.nl/, McGraw-Hill Publishers, ISBN 0077095537, 1999.

Solingen, R. van, Derks, P., Hirvensalo, J., 'Product focused SPI in the embedded systems industry: Experiences of Dräger, Ericsson and Tokheim', VTT Symposium Series, Proceedings of the 1st PROFES Conference, Oulu, Finland, June 22-24, 1999a.

Solingen, R. van, Kusters, R., Trienekens, J., Uijtregt, A. van, 'Product focused software process improvement (P-SPI): Concepts and their application', Quality and Reliability Engineering International, Wiley, Nov/Dec 1999b.

Stark, G., Durst, R.C., Vowell, C.W., 'Using Metrics in Management Decision Making', IEEE Computer, pp. 42-48, September 1994.

Stevens, R., Paltu, M., Subject to Requirements, Computing, September 1994.

Strien, P.J. van, Practice as science (In Dutch), van Gorcum, Assen, 1986.

Swieringa, J., Wierdsma, A.F.M., On the way to a learning organisation: on learning and education in organisations (in Dutch), Wolters Noordhoff Management, 1990.

Taramaa, J., Khurana, M., Koll, R., Kuvaja, P., Lehtonen, J., Oivo, M., Rodenbach, E., Seppänen, V., Solingen, R. van, 'Detailed specification of specific embedded systems characteristics', PROFES public report, 1997.

Thayer, R.H., Dorfman, M., Software Requirements engineering, IEEE Computer Society, 1997.

Thayer, R.H., Thayer, M.C., 'Software Requirements Engineering Glossary', In: Software Requirements engineering, Editors: Dorfman and Thayer, IEEE Computer Society, 1997.

TickIT, Guide to software quality management systems, construction and certification using EN 29001, TickIT Office, London, 1995.

Trienekens, J.J.M., Time for Quality: Working towards better information systems (in Dutch), Ph.D. Thesis, Thesis Publishers, Amsterdam, 1994.

Trienekens, J., Veenendaal, E. van, Solingen, R. van, Punter, T., Zwan, M. van der, Software quality from a business perspective: Directions and advanced approaches, Kluwer Bedrijfsinformatie, 1997.

Uijtregt, A. van, Product focused software process improvement: Integrating SPI and SPQ approaches into a quality improvement method for RPS, Master Thesis Eindhoven University of Technology, 1998.

Ulrich, D., 'Intellectual capital = competence x commitment', Sloan Management Review, pp. 15-26, Winter 1998.

Veld, J. in 't, Analysis of organisational problems: An application of thinking in systems and processes (In Dutch), Stenfert Kroesse, 1975.

Verschuren, P., Doorewaard, H., Designing a research (In Dutch), Lemma, 1995.

Weggeman, M., Knowledge Management (In Dutch), Scriptum Management, 1997.

Weinberg, G.M., Quality Software Management: Vol. 1: Systems Thinking, Dorset House Publishing, New York, 1992.

Weinberg, G.M., Quality Software Management: Vol. 2: First-Order Measurement, Dorset House Publishing, New York, 1993.

Wenger, E., Towards a theory of cultural transparency: elements of a social discourse of the visible and the invisible, Institute for research on learning, Hanover, 1990.

Wohlwend, H., Rosenbaum, S., 'Schlumberger's software improvement program', IEEE Transactions on software engineering, 20(11), pp. 833-839, 1994.

Wrycza, S., 'The ISAC driven transition between requirements analysis and ER conceptual modelling', Information Systems 15, pp 603-614, 1990.

Yeh, H., Software Process Quality, McGraw-Hill, 1993.

Yeh, R.T., Ng, P.A., 'Software Requirements - A Management Perspective', In: Systems and Software Requirements Engineering, Editors: Dorfman and Thayer, IEEE, 1990.

Yin, R.K., Case Study Research: Design and methods, Sage, 1994.

Zuse, H., Software Complexity: Measures and methods, De Gruyter, Berlin, 1991.

Zwaan, A.H. van der, 'From case to case: discovery or validation? On the incomplete utilisation of case-studies' (In Dutch), Nobo, 1998.

Samenvatting (Summary in Dutch)

Het belang van software in de hedendaagse maatschappij neemt nog immer toe. Alom bekende software toepassingen zijn tekstverwerkers, internet browsers, e-mail, boekhoudkundige pakketten, ERP toepassingen enz. Voor deze voorbeelden is het duidelijk dat men met software werkt. Er is echter een veel groter toepassingsgebied van software, waarbij software is ‘ingepakt’ in een (elektronisch) product. Deze zogenaamde ‘embedded producten’ met daarin ‘embedded software’ worden steeds meer toegepast en komt men op steeds meer plaatsen tegen in de maatschappij. Voorbeelden van embedded producten zijn: autoradio’s, televisies, (mobiele) telefoons, wasmachines, kopieerapparaten, benzinepompen, horloges, en dergelijke. Embedded producten zijn er in alle soorten en maten, worden door zeer veel verschillende gebruikers gebruikt, en variëren sterk in prijs en in toepassing.

De kwaliteit van dergelijke producten en dus ook van de daarin verwerkte embedded software is een relevant onderwerp van onderzoek. De maatschappij is inmiddels behoorlijk afhankelijk van embedded producten. Welzijn, geluk, de financiële situatie en soms zelfs het leven van mensen worden beïnvloed door de kwaliteit van deze embedded software. Echter kwaliteit is zeer divers interpreteerbaar en veel moeilijker te meten dan andere belangrijke aspecten van producten zoals de prijs en de levertijd van het product. Daarnaast is kwaliteit altijd een afweging tegen diverse aspecten: een betere kwaliteit kost meer en het duurt langer voordat een goed kwalitatief product is uitontwikkeld.

Het onderzoek dat besproken wordt in dit proefschrift poogt een oplossing te geven voor de moeilijkheid om producten van goede kwaliteit te ontwikkelen. Daartoe wordt een conceptueel model ontwikkeld waarmee organisaties hun embedded software ontwikkelproces zodanig in kunnen richten dat dit proces zo goed mogelijk bijdraagt aan de gewenste productkwaliteit. Deze aanpak is voornamelijk gebaseerd op de aanname dat productkwaliteit bepaald wordt door de volgorde en mate waarin ontwikkelactiviteiten en maatregelen worden uitgevoerd, kortom de kwaliteit van het product is afhankelijk van de kwaliteit van het proces. Daartoe dient voor ieder product een specifiek proces ontworpen of samengesteld te worden dat op de meest effectieve én efficiënte wijze bijdraagt aan de kwaliteit van het product.

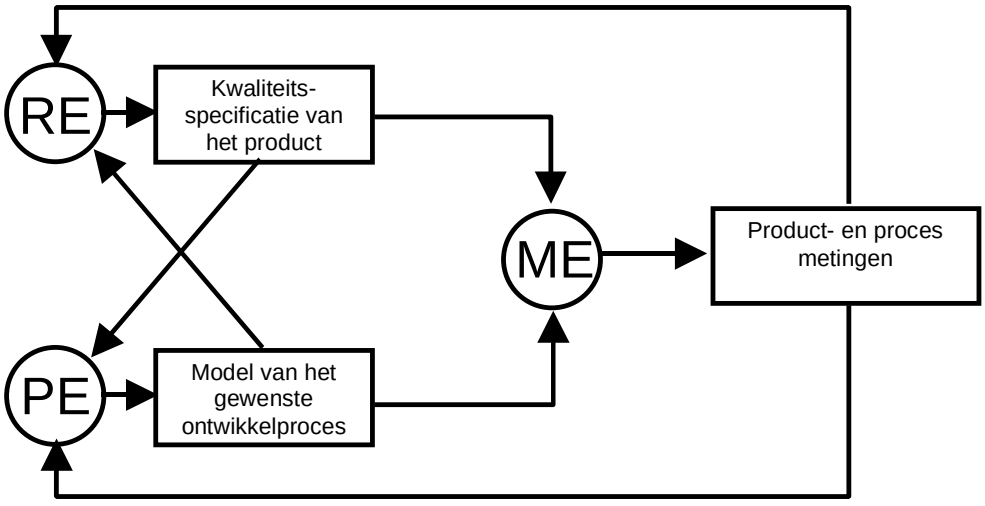
Op basis van beschikbare literatuur en industriële ervaringen op het gebied van software product- en proceskwaliteit en methoden voor SPI (Software Process Improvement), is in dit boek een analyse gemaakt van de sterke en zwakke punten van bestaande SPI methoden voor toepassing binnen het embedded product domein. Een methode voor

productgerichte procesverbetering dient deze sterke punten te behouden en een groot deel van de zwakke punten (indien mogelijk) op te lossen. De belangrijkste tekortkomingen van bestaande SPI-methoden welke in dit onderzoek zijn aangepakt zijn: het onvoldoende expliciet maken van productkwaliteitsbehoeften van gebruikers, het onvoldoende meten aan zowel het product als aan het proces, het onvoldoende meten van opbrengsten en kosten van procesverbetering, het onvoldoende flexibele en het abstracte niveau waarop procesverbetering wordt beschreven en het onvoldoende specifiek ondersteunen van individuele projecten.

Op basis van deze tekortkomingen is in dit onderzoek een conceptueel model ontwikkeld voor productgerichte procesverbetering. Dit model onderscheidt drie belangrijke gebieden waarop werk verricht dient te worden:

- Requirements engineering (RE), waarin kwaliteitseisen van directe en indirecte gebruikers worden verzameld en 'vertaald' worden middels een generieke ontwerptaal naar een (meetbare) kwaliteitsspecificatie van het product;
- Proces engineering (PE), waarin procesmaatregelen worden geselecteerd en een ontwikkelproces wordt samengesteld dat er specifiek op gericht is om zo optimaal mogelijk de gewenste productkwaliteit te realiseren;
- Measurement programme engineering (ME), waarin een aantal proces- en productmetrieken wordt verzameld en geanalyseerd met het doel te evalueren of de uitgevoerde procesmaatregelen de verwachte effecten opleveren en of de uiteindelijke productkwaliteit overeenkomt met de kwaliteitsspecificatie.

Dit conceptuele model is afgebeeld in Figuur 1.



Figuur 1: Conceptueel model voor productgerichte procesverbetering (RPM)

Het conceptuele model geeft de drie werkgebieden weer en hun onderlinge relaties. Met name het onderhandelingsspel tussen requirements engineering en proces engineering is hierin duidelijk weergegeven. Het is namelijk zelden het geval dat een kwaliteitsspecificatie van een product in één stap kan worden opgesteld. Er vindt altijd een onderhandelingsronde plaats waarin kwaliteitseisen ten opzichte van elkaar worden afgewogen (geprioritiseerd) en waarin wordt nagegaan in welke mate het ontwikkelproces in staat is aan deze eisen te voldoen, tegen welke kosten en doorlooptijden. Deze onderhandeling leidt tot een specificatie van gewenste productkwaliteit en een model van het gewenste ontwikkelproces dat geacht wordt deze kwaliteit te kunnen realiseren. Daarnaast wordt een meetprogramma opgestart dat middels proces- en productmetingen terugkoppeling verzorgt aan requirements engineering over de mate waarin het product voldoet aan de eisen, en aan proces engineering over de mate waarin genomen maatregelen ook de gewenste effecten hebben.

Het model is gebaseerd op een belangrijke voorwaarde: namelijk dat de bijdragen van individuele procesmaatregelen aan productkwaliteit bekend zijn. Helaas is dat vaak niet het geval. Hiervoor zijn een aantal oorzaken. Ten eerste zijn de effecten van procesmaatregelen niet constant en verschillen per situatie. Een maatregel zoals bijvoorbeeld gestructureerd testen kan in het ene project een grote invloed op productbetrouwbaarheid hebben, maar in een ander project kan deze bijdrage minimaal zijn. Ten tweede wordt er in de praktijk onvoldoende aan specifieke maatregelen gemeten om te kijken wat hun effecten zijn. En ten derde is software ontwikkeling een nog zeer jonge discipline waarin men blijkbaar nog niet toe is aan het meten van de effecten van procesmaatregelen. Onderzoek richt zich met name op het bedenken van nieuwe maatregelen zonder goed te kijken naar hun effecten of naar de effecten van reeds bestaande maatregelen.

Dit tekort aan generieke kennis over effecten van procesmaatregelen op productkwaliteit wordt in dit onderzoek aangepakt door aan te nemen dat dergelijke kennis vooral situationeel is en dat daarom de situationele effecten *geleerd* dienen te worden. Leren kan op verschillende manieren geschieden, echter is het vergelijken van ‘verwachte effecten’ met ‘gemeten effecten’ al een eerste leercurve die expliciet in het conceptuele model van Figuur 1 zit verwerkt. Daarnaast is tijdens dit onderzoek een uitgebreide analyse van leertheorie uitgevoerd om te kijken in hoeverre dergelijke theorie handvatten aan kan bieden om de leereffecten van procesverbetering te kunnen verhogen. De belangrijkste uitkomsten van dit onderzoek zijn een aantal ‘learning enablers’: criteria die aangeven hoe het leren in organisaties bevorderd kan worden. Daarnaast is een uitbreiding van het conceptuele model voorgesteld waarin naast ‘single-loop leren’ ook ‘double-loop leren’ wordt onderscheiden. Single-loop leren richt zich voornamelijk op het leren binnen een huidige situatie, wat al in het model van Figuur 1 zit geïntegreerd. Daarnaast moet ook

‘double-loop’ geleerd worden, wat betekent dat gekeken wordt of de huidige werkwijze nog wel ideaal is en of er geen rigoureuze wijzigingen nodig zijn. Daartoe wordt een apart leerproces onderscheiden dat zowel op basis van interne als externe informatie bepaalt of de huidige set van procesmaatregelen nog afdoende is, of dat er moet worden geïnvesteerd in nieuwe maatregelen. Een voorbeeld van double-loop leren is bijvoorbeeld de ervaring van een ontwikkelgroep die er achter komt dat in een (nieuwe) situatie het uiteindelijke systeemgebruik volstrekt onbekend is en dat daarom het normaal gebruikte ‘watervalmodel’ vervangen dient te worden door een meer ‘iteratief’ ontwikkelproces.

Dit conceptuele model voor productgerichte procesverbetering is echter van een hoog abstractieniveau en toepassing in praktijksituaties wordt nog onvoldoende operationeel gemaakt binnen het model. Daartoe is een set van praktijkrichtlijnen opgesteld op basis van literatuur en praktijkervaringen. Deze richtlijnen geven aan ‘wat’ er gedaan dient te worden binnen de drie werkgebieden van het conceptuele model. Deze richtlijnen geven niet aan ‘hoe’ dat dient te gebeuren omdat ervaring heeft geleerd dat dit grotendeels afhankelijk is van de specifieke situatie (organisatie of project) waarin het RPM model wordt gebruikt.

Het conceptuele model en de daarbij behorende praktijkrichtlijnen zijn binnen dit onderzoek gebruikt in twee bedrijven: Schlumberger RPS/Tokheim en Dräger Medical Technology. Schlumberger RPS/Tokheim ontwikkelt embedded producten voor benzinepompstations zoals betaalterminals, kassasystemen en benzinepompen. Dräger Medical Technology ontwikkelt medische apparatuur zoals beademingsapparatuur, anesthesie-apparatuur en patiënt-monitorsystemen. Binnen Schlumberger RPS/Tokheim is een longitudinale casestudie uitgevoerd waarin drie geneste cases waren opgenomen. Voor ieder van de drie belangrijkste embedded producten van dit bedrijf is een productgericht procesverbeterprogramma opgezet en uitgevoerd. Daarnaast is binnen Dräger Medical Technology een casestudie uitgevoerd om te kijken in hoeverre de ontwikkelde methode generiek toepasbaar en succesvol is.

De ervaringen in deze casestudies leiden tot de conclusie dat het RPM model en de daarbij behorende richtlijnen een goede aanpak beschrijven voor productgerichte procesverbetering voor embedded software ontwikkeling. De aanpak bleek voldoende flexibel om in alle vier de toepassingen een zodanig verbeterprogramma op te zetten dat dit aansloot bij de productkwaliteitsdoelen van de desbetreffende ontwikkelprojecten. In alle vier de gevallen heeft dit geleid tot aanzienlijke verbeteringen van de productkwaliteit. In geen van de vier casestudies is het verbeterprogramma mislukt, wat duidt op een generieke toepasbaarheid van de beschreven aanpak. De ontwikkelaars en managers van de ondersteunde ontwikkelprojecten waren enthousiast over de aanpak en waardeerden met name het doelgerichte en het project specifieke karakter van de aanpak.

Analyse van de kosten en opbrengsten van het toepassen van het conceptuele model en de richtlijnen in de casestudies toont aan dat de kosten relatief laag zijn en de opbrengsten noemenswaardig. De metingen toonden aan dat de ontwikkelprojecten minder dan 1% van hun tijd in het verbeterprogramma hoeven te investeren en dat de totale inspanning voor een verbeterprogramma beperkt blijft tot twee maanden (340 uur). De directe voordelen die konden worden gemeten, waren reeds ruim voldoende om de kosten te dekken. Daarnaast waren met name de projectmanagers erg enthousiast over de vele indirecte voordelen, zoals meer expliciete aandacht voor kwaliteit, een betere samenwerking met de kwaliteitsafdeling en een groter kwaliteitsbewustzijn bij de ontwikkelaars.

De hoofdconclusie waarmee het boek afsluit is drieledig.

- Ten eerste wordt geconcludeerd dat het creëren van productkwaliteit geen technisch of algoritmisch proces is, maar vooral een onderhandelingsproces. Onderhandelingen tussen (indirecte en directe) gebruikers en de afweging van de kwaliteitseisen tegen kosten, doorlooptijden, risico's en haalbaarheid zijn essentieel voor het verkrijgen van een kwalitatief goed product. Een dergelijk onderhandelingsproces blijkt erg complex en moeizaam in elkaar te zitten, wat waarschijnlijk ook een reden is dat de meeste projecten dit onvoldoende uitvoeren. De consequenties daarvan voor productkwaliteit moge duidelijk zijn.
- Ten tweede is gebleken dat het verbeteren van productkwaliteit hulpmiddelen behoeft. Deze hulpmiddelen hoeven echter helemaal niet complex te zijn. Eenvoudige hulpmiddelen zoals beschreven in dit boek kunnen goed helpen bij het expliciet maken van kwaliteitsbehoeften, het afwegen van dergelijke behoeften en het inzichtelijk maken van kwaliteit zowel van het product als van het proces. Een aantal voorbeelden van dergelijke hulpmiddelen worden in dit boek besproken en zijn met succes toegepast in vier industriële procesverbeterprogramma's.
- Ten derde heeft dit onderzoek aangetoond dat productkwaliteit tot stand komt in projecten. Daarom horen methoden voor procesverbetering vooral aandacht te besteden aan het projectniveau. Vele bestaande methoden voor procesverbetering richten zich echter met name op de 'organisatie' als geheel en missen daarmee de concrete verbeteringen die op het gebied van productkwaliteit te behalen zijn.

About the author

Rini van Solingen was born on September 2, 1971 in Middelburg, the Netherlands. He graduated from high school at the Oranje Nassau College in Zoetermeer in 1989. He received a Master of Science degree in Technical Informatics (Computer Science) from Delft University of Technology in July 1995. His graduation assignment was carried out at Schlumberger Retail Petroleum Systems in Bladel, under the supervision of Dr. Egon Berghout, Prof. Bas Brussaard and drs. Frank van Latum. During his student years he also worked as a software engineer and courseware designer in several projects at RPA, a software development and training institute in The Hague.

In September 1995, Rini returned to work for Schlumberger RPS and Tokheim as a senior software quality engineer leaving in September 1999. During this period he worked as a process and product quality improvement consultant in development projects, and as project member in several national and international co-operation projects. In one of these projects: the Esprit project 23239: PROFES, he worked as work-package manager under project manager Prof. Markku Oivo. In this role Rini was responsible for the co-ordination of the methodology application experiments in the industrial companies: Dräger Medical Technology, Ericsson LMF, and Tokheim.

In parallel with his industrial work Rini started his Ph.D. research at the faculty Technology Management at Eindhoven University of Technology in September 1995, under the supervision of Prof. Theo Bemelmans, Prof. Aarnout Brombacher, Prof. Rob Kusters and Dr. Ir. Jos Trienekens. In addition to the Ph.D. thesis, Rini's research has led to over 40 publications in international journals and conference proceedings. Furthermore, his industrial and university achievements led to the publication of the McGraw-Hill book: 'The Goal/Question/Metric Method: A practical guide for quality improvement of software development' (http://www.gqm.nl), which he co-authored with Egon Berghout.

On January 3, 2000, Rini started his new assignment as head of the 'Quality and Process Engineering' department at the Fraunhofer Institute for Experimental Software Engineering in Kaiserslautern, Germany, under the leadership of Prof. Dieter Rombach. Rini lives in Kaiserslautern, together with his wife Patricia.

Index

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Z