Knowledge management in Medium-Sized Software Consulting Companies

An investigation of Intranet-based Knowledge Management Tools for Knowledge Cartography and Knowledge Repositories for Learning Software Organisations

Torgeir Dingsøyr

Knowledge Management in Medium-Sized Software Consulting Companies

An Investigation of Intranet-based Knowledge Management Tools for Knowledge Cartography and Knowledge Repositories for Learning Software Organisations

Submitted for the Partial Fulfillment of the Requirements for the Degree of Doktor Ingeniør

Department of Computer and Information Science
Faculty of Information Technology, Mathematics and Electrical Engineering
Norwegian University of Science and Technology

January, 2002

ISBN 82-7477-107-9

Henvendelser om denne boka rettes til:
Telefon: 22 85 30 30
Telefaks: 22 85 30 39
E-post:unipubforlag@sio.uio.no
Boka kan også bestilles på www.gnist.no

Omslag: Askim Grafix AS
Trykk og innbinding: GCSM AS, Oslo 2002

Layout: Hanne Holmesland

Unipub AS er et datterselskap av Akademika AS, som eies av Studentsamskipnaden i Oslo

Abstract

Companies that develop software have a pressure from customers to deliver better solutions, and to deliver solutions faster and cheaper. Many researchers have worked with suggestions on how to improve the development process; software process improvement. As software development is a very knowledge intensive task, both researchers and industry have recently turned their attention to knowledge management as a means to improve software development. This often involves developing technical tools, which many companies have spent resources on. But the tools are often not used in practise by developers and managers in the companies, and it is often unknown if the tools improve how knowledge is managed.

In order to build efficient knowledge management tools, we need a better understanding of how the tools that exist are applied and used in software development.

We present and analyse eight case studies of knowledge management initiatives from the literature. We found evidence of improved software quality, reduced development costs and evidence of a better working environment for developers as a result of these initiatives.

Further, we examine success criteria in knowledge management codification initiatives, based on Intranet tools in medium-sized software companies. We found four factors that we consider important: Having a culture for sharing knowledge, having a stable focus on knowledge management, developing knowledge management tools incrementally, and coupling knowledge management initiatives well to business goals. This research was based on participation with software companies in improvement projects.

In addition, we investigate how knowledge management tools are used for different purposes by different groups of users in two software consulting companies. They use tools both as support for personalization and codification strategies. The consulting companies are two medium-sized Norwegian companies with 40 and 150 employees, which work in

development projects that lasts from a few weeks to several years. We used semi-structured interviews with developers, project managers and managers, examined logs of tool usage, and company-internal minutes from development meetings, as well as handbooks, project plans and annual reports.

The frequency of usage varied between the two companies: in one, most employees used tools on a daily basis, whilst in the other, employees used tools weekly. We find that tools for codification are in use for transferring knowledge from projects in order to solve technical problems, get an overview of technical problem areas, avoiding rework in having to explain many people about the same technical solution, improving the employees' work situation by tips on better configuration of technical tools, and also for finding who knows what in the organisation. The tools for personalization are in use for searching for competence to solve technical problems, resource allocation, finding projects and external marketing, and for competence development. In all, we found a variety of uses of a variety of tools by several groups of employees in a company.

«The maturation of the information technology revolution in the 1990s has transformed the work process, introducing new forms of social and technical division of labor.»

Manuel Castells in The Rise of the Network Society

Contents

Acknowledgements

During the work on this thesis, I have benefited from a lot of communication with many people, who have provided inspiration and motivation. First of all, I would like to thank my supervisor, Reidar Conradi, who has commented on an enormous number of drafts through the last four years. Also, thanks to present and former members of the software engineering group at the Department of Computer and Information Science, NTNU, for providing a good work environment: Elisabeth Bayegan, Roxana Diaconescu, Monica Divitini, Ekaterina Prasolova-Førland, Letizia Jaccheri, Jens-Otto Larsen, Øystein Nytrø, Tor Stålhane, Sivert Sørumgård, Carl-Fredrik Sørensen, Marco Torchiano and Alf Inge Wang.

Another source of inspiration has been the possibility to participate in two research projects during my PhD work: The Software Process Improvement for Better Quality (SPIQ) and Process Improvement for IT-industry (PROFIT) projects. I would like to thank participants from Sintef Telecom and Informatics: Tore Dybå, Geir Kjetil Hanssen, Nils Brede Moe and Kari Juul Wedde (now Clustra) as well as from the University of Oslo: Erik Arisholm, Dag Sjøberg, Magne Jørgensen and project manager Tor Ulsund from Bravida Geomatikk. Thanks are also due to the contact persons from companies that participated in these projects, which provided a pragmatic research environment that greatly influenced the focus of this PhD. I am also grateful to the Norwegian Research Council who financed the PhD work.

A part of the work for the thesis was done as fieldwork in two companies, and I am deeply grateful to these companies and the contact persons and the people I interviewed for their willingness to share experience about their knowledge management efforts.

During the last phase of the work, I was able to stay at the Fraunhofer Institute for Experimental Software Engineering in Kaiserslautern, Germany. Many people deserve thanks for both social and professional inclusion. It was especially nice to be able to work on the COIN Experience Factory project with Björn Decker and Markus Nick, which made me see knowledge managment from another perspective. And I am very

grateful to Klaus-Dieter Althoff for organising the stay, and also to the rest of the Systematic Learning and Improvement group at the institute for numerous discussions.

Through the years that I have been working as a doctoral candidate, I have benefited greatly from discussions with students that I have supervised in project and diploma work: Arne Bakkebø, Terje Nygaard, Bjørgulv O. Sandanger, Thies Schrader, Torkel Westgaard, Helge Jenssen and Bjørn-Ovin Wivestad. In the early phase of thesis work, I also had many discussions with Bent Ingebretsen, who had written his masters thesis in this field.

There are also other people that I would like to thank for collaboration during the thesis work: Emil Røyrvik on the Kunne project at Sintef Technology Management for discussions and writings on skills management. Trond Knudsen at Norsk Regnesentral for letting me present an early version of research approach which provoked many new thoughts. Petter Gottschalk at the Norwegian School of Management BI, for giving me an early version of his book on knowledge management which directed me to new references. Also, I would like to thank Frank Maurer at the University of Calgary and Mirjam Minor at Humboldt Universität Berlin for organising guest lectures where some of the material in the thesis was presented and discussed.

I am further grateful to the people who have commented on parts of the thesis: Of course, Reidar Conradi as my supervisor, but also Stefan Biffl at the Technical University of Vienna, Magne Jørgensen at the University of Oslo, and Monica Divitini, Letizia Jaccheri, Roxana Diaconescu and Alf Inge Wang at the Department of Computer and Information Science at NTNU. I am also grateful to Terje Brasethvik at the Information Systems group at NTNU for a number of discussions on research topics and comments on drafts. Also thanks to Gavin Gaudet for tips on how to improve the oral English in the Empirical Investigations chapter. And thanks to Preben Randhol for help with scripts to handle usage logs.

Finally, I would like to thank family and friends for inspiraton and encouragement, and especially Sissel for her patience.

Trondheim, January 21st
Torgeir Dingsøyr

List of Figures

List of Tables

1 Introduction

This thesis is about how Intranet-based Knowledge Management Tools can be used to support what has been called a «Learning Software Organisation». An Intranet-based tool is a software program that provides help for software developers. We will define what we mean by a tool more precisely later.

Software development usually takes place in team-based projects where the participants work towards a shared goal. Many companies have problems with transferring what people learn in one project to other projects in the same company. Knowledge Management is a set of strategies and techniques to increase the transfer and use of different types of knowledge in a company or organisation.

We find many knowledge management tools and methods in companies and in the research literature, but most of the scientific work on tools is concentrating on technology to build such tools; on the structure of knowledge and technical work on retrieval mechanisms. Also, work on knowledge management methods is usually describing an ideal way of collecting and sharing knowledge, which is often difficult to reproduce in practise. There is little work on how tools and methods for knowledge management are actually applied in the software engineering domain. Also, many tools that are introduced in companies are abandoned later. This is often because they turned out not to be so useful as people thought before they were introduced.

We think that we would be able to design better tools and methods, if we knew more about how the existing tools are used - or why they are not used.

In this thesis we discuss how companies can improve their knowledge management by adjusting Intranet-based knowledge management tools, and thus become more of a learning organisation. We will base this discussion on an examination of tools and initiatives that are used in medium-sized companies that develop software. These medium-sized companies are four case companies in a prestudy, and a main and a contrast

case in a main study - as well as reports of knowledge management tools from the literature.

Now, we go on to define a problem outline for this thesis that will be further narrowed later and state the main contributions of this thesis. Then, we briefly state what main choices we have made for carrying out research. Further, we narrow the scope of this work, and finally give an overview of the structure of the thesis.

1.1 Problem Outline

In this thesis, we are interested in studying how tools for knowledge management are used in medium-sized companies that develop software. The specific tools are Intranet-based tools that companies have produced themselves. There are, however, many such tools, and we will only be concerned with Knowledge Repository and Library, and Knowledge Cartography Tools. We will introduce these types later. and argue why these are particularly interesting to examine.

The type of companies where we have studied this phenomenon is in medium-sized companies in Norway that develop software. By medium-sized we will mean companies with from 50 to 500 employees.

Many knowledge management tools are in use in the software industry. But there has been done little work on how these tools actually work in practise. Also, many research prototypes for knowledge management tools exist in the research literature. But not many of them has made it into industrial practise.

We are then asking the following research question:

- How can Intranet-based knowledge management tools be used in medium-sized software consulting companies to facilitate a «learning software organisation»?

This research question will be further discussed and elaborated after we have introduced more theory. We will also elaborate what we mean by a «Learning Software Organisation».

The critical reader might already now ask: But do these knowledge management tools help solve the problems that the software industry has (which will be described in the next chapter)? The answer is: we are not sure. But we think we need to know more about the tools in use, and about how they are used before we can begin to answer the question of whether they are solving problems or not, and of how cost-effective they are.

But is it really any use in studying such tools? The technology is changing so fast. When we have completed this study, the tools will be completely different! Although we think that developing tools for knowledge management is a long process, and the ones we will study are by no means «completed» - we still think it is important to study how they work, before moving on to something else. It has been claimed that it is a general problem in software engineering, that we do not systematically study the effect of technology and methods, before we jump on to newer technologies. We think it is a sound scientific task to analyse the impact of «new» tools. Yet, we acknowledge that the results might be a bit «old» when we finish.

Then, when we examine such tools, what is the relation between their usage and the potential improvement of the productivity or quality of the software that is developed? It is a long chain of events from the effects of a knowledge management tool, to this knowledge being learned and used by employees, which should then finally affect the quality of the developed software or the productivity of the software development team. We do not intend to show a causal relationship between these factors, but we think a knowledge management tool is one of many factors in a good work environment that can stimulate learning, creativity, and employee motivation, which will affect the quality of the output. But we limit ourselves here to study how knowledge management tools can be used, leaving more «hard measurements» for further work.

1.2 Claimed Contributions

The work in this thesis can be divided into four major phases, where we claim to have some contributions in each, related to the field of studying Learning Software Organisations by empirical methods. Some work in

the thesis has been published before, and we give references to these papers for each phase:

- Literature study: We present literature on knowledge management in software engineering, and have made a taxonomy of knowledge management tools based on findings from the literature. We have also surveyed existing case studies of how knowledge management tools are applied in companies that develop software, and present, and discuss these approaches. This work can be found in chapter 3 in the thesis, and in papers 1 and 8.
- Method for experience capture: We have contributed in developing a method to capture experience from completed software projects though a group process: lightweight postmortem reviews. This method is given as an example of experience capture methods in chapter 3, and is described in further detail in papers 2, 5 and 6.
- Four cases studies on knowledge management in software engineering companies: Here, we studied four companies that have applied different knowledge management initiatives, and discuss success factors. The cases are presented in chapter 5.1, and discussed in chapter 6. This analysis has also been published as paper 4.
- Deep case study and a contrast case: We examine further what kind of knowledge management tools that exist in two companies, and describe how different groups of users apply them. The cases are presented in chapter 5.2 and 5.3, and are discussed in chapter 6. Some of the work here on Skills Management has been published in papers 3 and 10.

We have further published paper 7 as a first discussion on the selected research topic and research questions in this thesis, that can be found in chapter 4. Finally, paper 9 gives a further description of knowledge management tools than the ones that can be found in chapter 3.

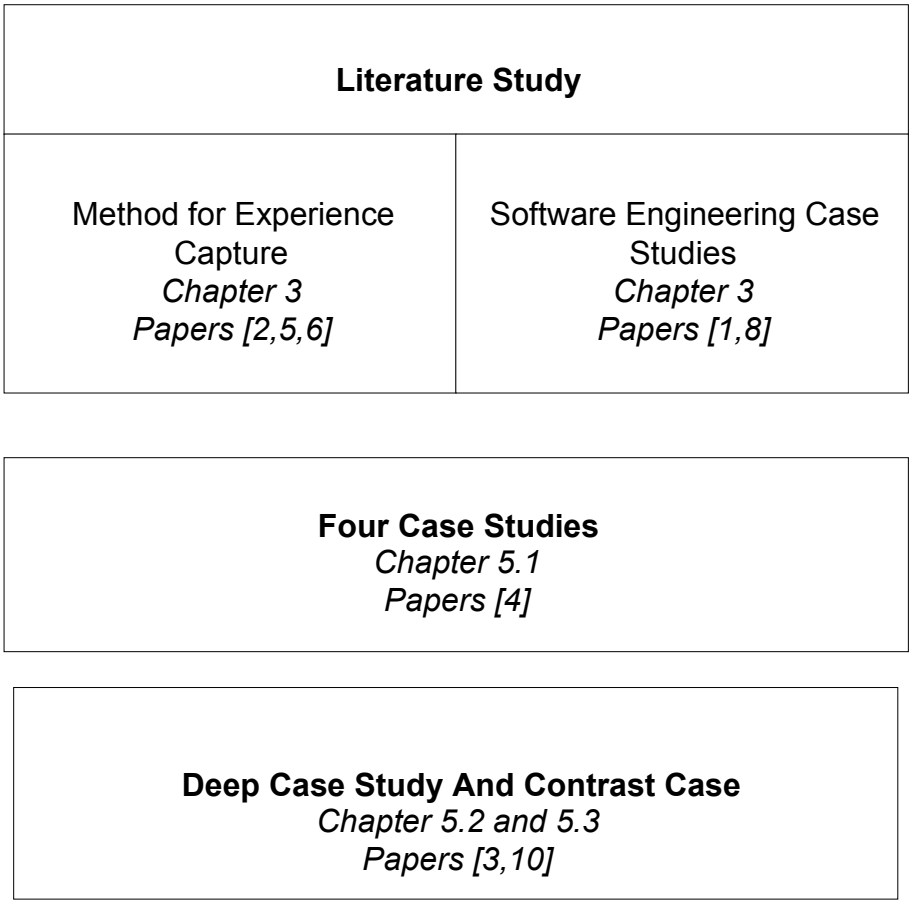

Figure 1.1: The main contributions in this thesis, with references to thesis chapters and published papers.

The following are the papers that has been published or are undergoing a publication process:

Journal articles

1. Dingsøyr, Torgeir, Conradi, Reidar: A Survey of Case Studies of Knowledge Management in Software Engineering, submitted to International Journal of Software Engineering and Knowledge Engineering. A previous version of this paper was published as paper 9.

2. Birk, Andreas, Dingsøyr, Torgeir, Stålhane, Tor: Postmortem: Never leave the project without it, submitted to IEEE Software, special issue on knowledge management in software engineering.

3. Dingsøyr, Torgeir, Djarraya, Hans Karim, Røyrvik, Emil: Managing Hard Skills: Findings from Practical Tool Use in a Software Consulting Company, submitted to IEEE Software, special issue on knowledge management in software engineering.

Book chapter

4. Dingsøyr, Torgeir, Conradi, Reidar: Knowledge Management Systems as a Feedback Mechanism in Software Development Processes: A Search for Success Criteria, submitted as a chapter to a book on Feedback and Evolution in the Software Process. A revised and extended version of: Conradi, Reidar and Dingsøyr, Torgeir (2000) Software experience bases: a consolidated evaluation and status report, Second International Conference on Product Focused Software Process Improvement, PROFES 2000, June 20-22, Oulu, Finland, Springer Verlag, vol. 1840, pp. 391 - 406.

Conference papers

5. Dingsøyr, Torgeir, Moe, Nils Brede and Nytrø, Øystein (2001) Augmenting Experience Reports with Lightweight Postmortem Reviews, Third International Conference on Product Focused Software Process Improvement, 10-13 September, Kaiserslautern, Germany, Springer Verlag, Lecture Notes in Computer Science, vol. 2188, pp. 167 - 181. Also published at the Norwegian Informatics Conference (NIK) 2001, Tromsø.

6. Stålhane, Tor, Dingsøyr, Torgeir, Moe, Nils Brede and Hanssen, Geir Kjetil (2001) Post Mortem - An Assessment of Two Approaches, EuroSPI, 10-12 October, Limerick, Ireland.

Workshop papers

7. Dingsøyr, Torgeir (2000) Focus for planned research: Knowledge Management for Software Process Improvement, The Ninth Nordic Workshop on Programming Environment Research, 28-30 May, Lillehammer, Norway.

8. Dingsøyr, Torgeir (2000) An evaluation of Research on Experience Factory, Workshop on Learning Software Organisations at the international conference on Product-Focused Software Process Improvement, Oulo, Finland, University of Oulu, VTT Electronics, Fraunhofer IESE, pp. 55 - 66.

9. Dingsøyr, Torgeir (2000) An Analysis of Process Support in Knowledge Management Tools for Software Engineering, Workshop on Flexible Strategies for Maintaining Knowledge Containers,14th European Conference on Artificial Intelligence, 20-25. August, Berlin, Germany, Humboldt-Universität zu Berlin, ECAI Workshop Notes, pp. 6 - 13.

10. Dingsøyr, Torgeir and Røyrvik, Emil (2001) Skills Management as Knowledge Technology in a Software Consultancy Company, Learning Software Organizations Workshop, 12 - 13 September, Kaiserslautern, Germany, Springer Verlag, Lecture Notes in Computer Science, vol. 2176, pp. 96-107.

1.3 Chosen Research Strategy

In researching the question outlined in section 1.1, we have chosen to investigate it in a real environment. That is, to go into a real organisation, and study tools in «vivo». We will discuss this further in the Research Methods and Design chapter. The main reasons for choosing to study real organisations, and doing case and field studies, are that:

- Many prototype knowledge management tools are already developed in research institutions, so the need for making more prototypes is small.
- Few studies exist on how knowledge management tools are used in software companies.

In software engineering, several environments have expressed the need for a more empirical basis of software engineering, promoting what has been called *empirical software engineering.*

In empirical software engineering, it is necessary to use different research methods than normally applied in software engineering. This is because we have no strict control of the environment. Also, in our case, there is

relatively little information to find about the usage of knowledge management systems in the research literature.

In studying organisations, we have used research methods that are common in social science, but not in technology-oriented disciplines such as software engineering. A common problem when using such methods is that: «technologists regard sociologists as, apparently, merely wishing to observe and give an account of what they observe, with no interest necessarily in this leading to social action». While on the other hand, «sociologists regard technologists as simply wanting plans of action to make their technology more 'effective'» (Low et al., 1996). Here, we hope that our proposed theory will be seen as a contribution to better understand the tools, and then be useful for anyone wanting to improve the design or usage of such tools later.

Much of the work in software engineering has been done in the spirit of modernity; with a rational view that the problems at hand can be solved if we just establish good enough work methods and tools. The search for a silver bullet (which will be discussed further in the next chapter) is evidence of such a view.

However, many people now have a more post-modern view of software development. That is, it is futile to «solve problems» related to organizational, human and technological factors by say technology alone. Instead of looking for a silver bullet, we can only hope to find a set of «weapons» - that will help us to reduce the impact of some problems as they appear to some people.

In fact, the whole idea of software «engineering» is questioned by some environments (see an interesting discussion on the engineering metaphor in (Bryant, 2000)). Engineering is often associated with words like «science», «mathematics», as well as «practical methods». But we could also see software development as a creative task (Glass, 1995), where for example improvisation (Dybå, 2000) is more important than rigour.

In this thesis we will adopt a subjectivist, or postmodern view, that different people might have different goals, and they do not necessarily always act in a pre-planned or even rational manner. In studying how people use knowledge management tools, we consider the software practitioners (or «community of practise») to be the true, skilled, experts

to judge what kind of tools they find useful or not. Therefore, we have opted for a research strategy with a close interaction with developers, project managers and management in the field. We will discuss this further in our chapter about research goals, method and design.

1.4 Research Context

The work which was performed in this thesis was a part of two larger research projects on software process improvement (SPI) which involved many Norwegian companies that develop software.

The Software Process Improvement for Better Quality (SPIQ) project aimed to increase the competitiveness of 12 participating Norwegian software companies, by creating an improvement environment in the companies, and introducing ideas from an American context, like the Experience Factory, and adopting it for small and medium-sized enterprises in Norway (Conradi, 1996). It also included pilot projects for improvement in companies, as well as discussion forums for issues related to process improvement. Further, it contained dissemination activities like conferences and the writing of a method handbook for process improvement in Norwegian (Dybå et al., 2000). This project lasted from 1997 to 1999.

This project was followed by the Process Improvement for IT industry (PROFIT) project, which focused more on software process improvement in companies with frequent changes in technology and market. Can such companies benefit from the same improvement initiatives as more stable organisations? This was one of the major questions in this project, which is still ongoing, and involved eight companies from the start. This project lasts from 2000 to 2002.

1.5 Scope

In this thesis, we are concerned with how Intranet-based tools are used for knowledge management in medium-sized organisations that develop software. We have thus limited the field of knowledge management to those processes that can be supported by computer tools, and specifically tools with a web-interface on a company-internal Intranet. We also

concentrate on a specific set of tools that will be discussed later. Further, we have limited the usage of these tools to the domain of software development and maintenance, and specifically in medium-sized companies, where most of the development is done «in-house», and where most of the staff spends much of their working day in front of a computer.

When we examined the knowledge management tools, we have only looked at how they are used. We have not looked at issued in developing such tools, and not on economical issues - whether they are cost-effective or not.

We have neither looked at specific tools for reusing code or other software artifacts, but at tools that operate on a higher level of abstraction. But these tools may be linked to code, like a system that help you solve problems by showing example code. In the companies where we have been working, reuse of code is usually organised through development of software libraries of «baseline products» that get input from all people in the organisation.

To introduce knowledge management as an «improvement» in an organisation is of course not without problems. What some people in the organisation see as «improvements» might be seen as «deteriorating» efforts by other people. For example, some employees might think that their knowledge is ignored by a company, because it is not included in a computer tool. This, and other political issues in deploying knowledge management tools are not issues that we will discuss here.

1.6 Structure of the Thesis

The structure of the rest of this thesis is as follows:

Chapter 2: Software Development; Problems and Remedies. In this chapter, we discuss what software development is about, and some of the challenges the field is concerned with. We also discuss some of the main improvement initiatives that have been in the field, and discuss one of them, namely knowledge management and learning organisations in more detail. Finally, we give an overview of research methods in software engineering.

Chapter 3: Knowledge Management: In General and in Software Engineering. Here we first discuss knowledge management in general, and then specifically its application in software engineering. We discuss terms like experience, information and knowledge, and other common terms in the knowledge management field, like organisational memory, corporate memory, and experience factory. We also examine how knowledge is transferred in an organisation, and introduce a knowledge management program as a strategy, a set of processes and a set of tools. We present case studies on knowledge management tools in companies that develop software, found in the literature.

Chapter 4: Research Goals, Method and Design. Here we further specify our research goals, using concepts from chapter 3. We list the topics of interest in the form of research questions. We present the research method that we selected, with arguments for why this approach is suiting the topics under study.

Chapter 5: Empirical Investigation. First, we present a prestudy of four case studies of knowledge management programs in Norwegian companies. Then, we present two companies where we did case studies, together with projects that we followed in each of them. We present the infrastructure for knowledge management that exist in the companies, and our findings on the usage of them.

Chapter 6: Discussion and Analysis. We discuss the findings from the literature, our prestudy and main study cases in light of the theory which is given in chapter 3.

Chapter 7: Conclusion and Further Work. We sum up the main findings from the discussion, and outline possible further work in the field of learning software organisations.

Appendix A: Interview guides - here we present the interview guides that was used in semi-structured interviews in the two companies in the main study.

Appendix B: Processed Usage Logs - Here, we list processed usage data from Knowledge Management Tools in Alpha, one of the main study companies.

This is a doctoral thesis, written for the research community. It is not the intention to come up with direct, practical aid for companies on how to improve their knowledge management, but more to bring forward theory about how knowledge management is used. This will hopefully make it into practise, but it is out of the scope to concentrate on that issue. That is the responsibility of the research field as a whole.

Reading this thesis requires knowledge of software engineering and specifically software process improvement, and what has been called learning software organisations (knowledge management in software engineering). It also requires knowledge of research methods in general.

2 Software Development; Problems and Remedies

Now, we first discuss what software development is about, and future trends. Then, we describe some of the main challenges for the software engineering field, and some solutions that have been suggested in the literature. Further, we present knowledge management and learning organisations as an interesting new field in improving software development practise, before we briefly discuss research methods in the software engineering domain: What methods have we got to scientifically examine the problems and remedies at hand?

2.1 Software development

To develop and maintain software is often referred to as «software engineering». One definition is that software engineering «is concerned with theories, methods and tools which are needed to develop software... for computers». It differs from other types of engineering because it is «not constrained by materials governed by physical laws or by manufacturing processes» (Sommerville, 1996). Of course, we also have constraints in software, for example due to manpower, organisation or skills.

With this definition of «software engineering» (see (Bryant, 2000) for an interesting discussion on the use of this word), we include everything from eliciting requirements for a software system from a customer, via specifying architectural details of the software system, to actual implementation in one or more programming languages. We also include activities to check or improve the quality of the software, like testing and inspection. Usually, the developers use a wide range of tools, from tools for handling documents to design tools and editors, compilers, debuggers and tools for version control and project management.

A common way to develop software, is to divide the work into several phases. A much referenced model of such phases is the waterfall model, which comes in many variants, but most include (Sommerville 1996):

- Requirements analysis and definition - to find what the software system will be used for, that is, find the requirements.
- System and software design - make decisions on technical issues, like software architecture, database design and user interface design.
- Implementation and unit testing - write, adapt or generate the actual code in a programming language, and test each program unit.
- Integration and system testing - check that the implementation fulfils the requirements.
- Operation and maintenance - enhance the software, or correct errors that are found during usage.

The sequential waterfall model works best when the requirements for the system are stable and easy to establish. For software where the requirements are largely unknown, or frequently changing, more incremental models for software development should be used.

To develop software is a typical example of what Peter Drucker has called «knowledge work»; where «value is (...) created by 'productivity' and 'innovation'» (Drucker, 1993). Knowledge is the only scarce resource in software development - not other «means of production» like computer hardware and software, office buildings or capital.

What is software likely to be in the future? A panel that constructed scenarios for software in the future saw some major trends in development and usage (Tellioglu and Wagner, 2000):

- Because you can charge more for a service than a product, software will more and more be seen as a service. This «implies new responsibilities for developers, particularly in light of the risk of software failure». For example, we might expect software companies to pay higher penalties if their software does not work.
- Developers often take many choices that the users have no influence on, but that will have an impact on how the software system can be used later. This will probably require developers to involve the users more in the technical development to a higher degree in the future.

- Software will interact more with users through natural forms, using speech and multimedia technology.
- Software will adapt to the users and their working styles.

In all, we can expect software to be more complex than today, and used in even more situations than today.

Other trends in software development that we have seen is that storage and memory has become practically free. Now the limitations are bandwidth. Some think that technical limitations will decrease in the future, and the limitations we will meet is in our own creativity and ability to design software solutions.

We have now given a broad overview of software development. Let us then go on to discuss some problems, which particularly are related to the quality of software, and the quality of the software development process.

2.2 Problems in Software Engineering: Overruns and Unfulfilled Requirements

To develop software is challenging. There are many examples of software projects that have failed. The much cited Standish report on software projects (1995) «shows a staggering 31.1% of projects will be cancelled before they ever get completed. Further results indicate 52.7% of projects will cost 189% of their original estimates. The cost of these failures and overruns are just the tip of the proverbial iceberg. The lost opportunity costs are not measurable, but could easily be in the trillions of dollars...». Now, we should be a bit careful in thinking that the percentages given in the report will be true for all projects, but they may give a good indication. The view that the software systems we use today are not very mature is also supported by the American «President's Information Technology Advisory Committee», that writes: «The Nation needs robust systems, but the software our systems depend on is often fragile. Software fragility is its tendency not to work properly - or at all. Fragility is manifested as unreliability, lack of security, performance lapses, errors and difficulty in upgrading» (Joy and Kennedy, 1999).

So what are the consequences of this kind of problems? In september 2001, the US bank Citigroup had severe problems as their «systems began crashing on Tuesday afternoon and nearly 24 hours later, officials of the largest US banking company were still scrambling to provide a reason»[1]. This affected 2000 ATMs and 750.000 online banking customers.

We can say that we have problems in software engineering that is related to low quality of software, and a long time to market. That is, developing a product can often take much more time than predicted, and this can be very critical in emerging markets like the Internet industry, where we saw companies offering a service first got all the attention of media and customers.

But before continuing - what do we mean by quality when we speak of software? Many define quality simply as «satisfied customers», but we have several other types of «quality», for example in requirement specifications, we can talk of syntactic quality (that the requirements are stated syntactically correct) and semantic quality (that the meaning of the requirement is correct) (Krogstie, 2001). Also, different user groups of software systems can have different perceptions of quality (See (Wong and Jeffery, 2001)).

So why are there so many problems related to software development projects? Software is an immaterial product, and it can be difficult to get an overview of a total program system, which can be millions of lines of code, to identify all possible error sources. Also, a very small defect might have a lot of influence in critical systems, like the European Space Agency's Ariane 5 satellite launcher, that ended in a failure in 1996. About 40 seconds after initiation, the launcher «veered off its flight path, broke up and exploded» according to the report by the inquiry board (Lions, 1996). The error was «caused by an internal variable related to the horizontal velocity of the launcher exceeding a limit which existed in the software». Thus, just a few lines of code that was lacking, had severe consequences - a loss of around 312 million Euro. We could say that this was a process error, in that the module that failed was not tested under the right conditions.

---

1 Article «Citigroup struggles with growing pains», Financial Times, Friday 7th of September, 2001.

Other problems can be that the communication between the end- users and the software developers is lacking, or that project management is difficult in an environment where a small bug can take a very long time to correct, and where it is often difficult to estimate the schedule and amount of remaining work.

Numerous examples of problems in software development projects can be found in popular books like Crash - Learning from the World's worst Computer Disasters (Collins and Bicknell, 1997) and Software Runaways (Glass, 1998).

After listing all these problems that exist in software and its development, you may ask: are all software systems that bad? Of course, it is not so, there are a lot of software projects that deliver software that is highly usable and working. Robert Glass has argued that the software failures are the exception rather than the trend (Glass, 2000) - «we tend to focus on the unusual things that go wrong because they're more interesting or important than the run-of-the-mill things that go right». Glass argues that we should not use a word like «crisis» to describe the software development field when we know of so many well-working systems. The main reason for this argument is that problems in software are used to motivate a lot of research - which should be able to stand on it's own feet.

We acknowledge that there have been more writings about the failures than the successes in software engineering projects, and that the situation might not be as bad as it looks. But as the reports we have cited earlier show, there is at least quite a lot of processes and projects that could improve, although it is not right to use a word like «crisis».

2.3 Suggested Solutions: Is There A Silver Bullet?

There has been much discussion in the software engineering community about finding a «silver bullet» to end the problems, or at least reduce the impact of them. Several solutions have been suggested to improve the way software is developed. Some have tried to change the way software is produced; the «process», some by introducing new programming languages, and other have worked with supporting tools to assist in devel-

opment. The goal is usually to increase productivity and/or the quality of the software that is made.

There are a lot of factors involved in developing software; we have outlined a set of major factors in Figure 2.1.

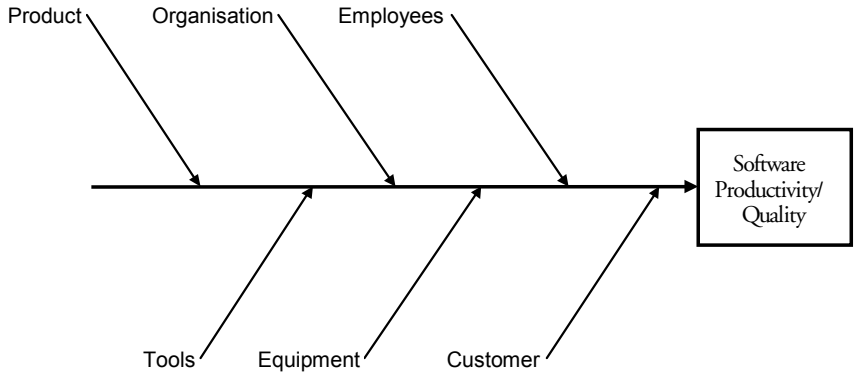

Figure 2.1: Some factors that influence productivity and quality in software development.

Clearly, software development depends on the software product to be developed, but also on the way software development is organised. (Organisation includes what kind of work processes that are used, and how communication in projects is organised). Other factors are: the skills and attitudes of the individual developers, what computer tools that are used, as well as what physical equipment (such as computers). And it depends on the need of, and the relations to the customer.

But let us go back to the suggested remedies for improving the productivity and quality of software. The outcome of several of these initiatives was summed up in an article in Communications of the ACM (Glass, 1999). Claims of «order of magnitude» improvements were evaluated, on different «technologies» like:

- Structured techniques - using structured analysis, design and programming.
- Fourth generation programming languages (application generation).
- Computer Aided Software Engineering - tools to support software engineering, mainly in analysis and design.

- Formal methods - formal specification and verification of software.
- Cleanroom methodologies - methods for removing software defects.
- Process models - descriptions of appropriate processes in software engineering.
- Object-oriented technology - to find «objects» in the problem to be solved, and use those in generating software solutions.

Many of the technologies show promising results, but there are few scientific studies that evaluate how the different technology and methods actually work. Also, in some studies that claim improvement, the improvement technology is confused with other changes, like changes in the programming language. So there is still a need for more systematic studies on how these technologies work in practise.

If we look at the improvement initiatives that include organizational as well as technical aspects, we find a subfield of software engineering named «software process improvement». The idea here is to change work practice to be more effective or predictive, or to develop software with higher quality. The underlying idea is that the way you produce software affects the final product. Within this field, we find what we can describe as two different positions: One that imposes «top-down standardization» to increase the quality of software, and one that imposes more «bottom up» quality improvement initiatives. To give a further overview of this area, we also present «software process modelling», which has provided important contributions to this field, and finally the topic that interests us the most; improving software by stimulating learning: «Knowledge Management and Learning Organisations».

Let us first present the standardization approach, then Total Quality Management and Process Modelling before we briefly discuss these approaches. Note that the two first approaches may require a «champion» in the organisation that is seeking improvement: A person who initiates and follows up improvement efforts - either in the top or in the bottom of the organisation.

2.3.1 Improving through Standardization; The Capability Maturity Model

The motivation here is, like standardization in other fields, that if we develop software in a more well-defined and predictable way, the resulting software will also be of higher quality, and it will be easier to reach goals on cost, schedule and quality for a software development project. In a statistical sense, we can say that we aim to reduce the variance in cost and quality between different teams and projects, in other words to reduce the risk for severe overruns. The most known standardization approach in the software engineering field, is the Capability Maturity Model (Humphrey, 1989) developed at the Software Engineering Institute at Carnegie Mellon University. This is a framework to evaluate the «maturity» of software developing companies, where companies that produce software in a very planned and documented way is considered to have a «higher maturity level» than other companies. The framework divides companies into five levels (Paulk et al. 1995, Pfleeger, 1998):

1. Initial - software development is done in an ad-hoc fashion, with little control on effort spent or remaining, and on the quality of the software.
2. Repeatable - inputs and outputs of different parts of the development process are defined, such as budget, schedule, resources that will be used, as well as functionality. Each project has a standard.
3. Defined - activities to produce the software («processes») are documented and standardized. The organisation has a standard.
4. Managed - measures of process and product quality make it possible to find problems and assess effects of possible solutions. The risk of overruns is reduced.
5. Optimizing - new tools and techniques are tested to find how they work before they affect processes and products, and possible faults are discovered before they appear. This means to be more efficient.

Some have criticized this way of improving because the model is not suited to the everyday problems of most software developing companies. It was originally developed for software contraction in the defence industry, but has since been applied in other fields, like in the telecommunication industry. However, CMM is not very tailorable to the situation a company might be in, it only focuses on the more managerial aspects of the development process, and does not consider that most software

companies have to sell their product in a market. So one critique of this model, like with most other forms of standardization, is that a company can get a very high score without really doing well in the market. This is similar to being «ISO-certified» to make useless life jackets of concrete - as long as the production process is well documented. Anyway, many of these issues, especially in the lower levels of the CMM are issues that most companies will benefit from, like more systematic version management and project planning.

2.3.2 Involving everyone in Improvement; Total Quality Management and the Quality Improvement Paradigm

Another position in the software engineering community is to try to improve, but focus more on the specific needs of a company in its market situation. This is based on thoughts from Total Quality Management, TQM (Deming, 2000, Pascale, 1991), which has been a popular improvement strategy the last twenty years. Some important aspects of this approach has been (Neerland, 2000): organized improvement, involvement of every employee, increased customer support, improved performance and integration of activities. «Quality is everyone's responsibility», and «quality is satisfied customers» are slogans that gives a good description of TQM, and several technologies have been developed to help all employees in a company to focus on quality.

A central idea in TQM is to learn from the activities that you do in a company. For this purpose, and for improving your performance from what you learn, the *plan-do-check-act* cycle is a structured way of working. The idea is to first *plan* an improvement or change activity, then *do* it, then *check* whether you reached the intended goals, and finally *act*; make changes to work processes in order to do better the next time, based on what you have learned.

This kind of feedback-loop is also used in software engineering under the name Quality Improvement Paradigm (Basili, 1985), developed at the NASA Software Engineering Laboratory. Here, we find six steps to apply in improvement work: 1) characterise the environment, 2) set goals, 3) choose process, 4) execute, 5) analyse, and 6) package. This is a further breakdown of the steps from total quality management, but the fo-

cus in the quality improvement paradigm has traditionally been on gathering quantitative data. A technique for focusing data collection is called the Goal Question Metric method (van Solingen and Berghout, 1999). Here, you start by defining some goals, like «to improve the quality of the user interface», then go on to find some questions that can give you an answer to whether you have reached you goal or not, like «how often does the user interface crash?» or «how much time does users spend to familiarize with the user interface?», and then you finally decide on some metrics to define the data to collect. In our example, this could be «number of user interface system crashes per 100 hours», or «average time before users claim to master the interface». When a company wants to measure how a new method or tool performs over time, the goal question metric method can be a valuable support as a part of a quality improvement program.

2.3.3 Making Work Practice Explicit, and Automate it

Yet other people have been working on process-sensitive tools to define and support the software process that the developer is supposed to follow, for example by assisting in tasks like planning and organisation. A software process can be defined as «the coherent set of policies, organizational structures, technologies, procedures, and artefacts that are needed to conceive, develop, deploy and maintain a software product» (Fuggetta, 2000).

An important step here is to find how software is actually developed in a company (to elicit and define the process model), and then to design tools that support this way of development (enact in the development process). Of course, it is also possible here to change the way processes are carried out in order to make development simpler or focus more on quality aspects in the development.

2.3.4 Summary and Discussion

Now, we have seen some different perspectives on what software process improvement can be. All the views we have presented can overlap in normal improvement activities in software companies, and the different fields loan from each other. In the Capability Maturity Model, for example, work processes should be documented when you reach level two.

This documentation is the expertise of the software process modelling field. Such process models can also be a prerequisite in more bottom-up improvement initiatives involving measurement, as in the Quality Improvement Paradigm. And finally, in a computer-supported knowledge management tool which we will discuss in the next section, can be beneficial to «tag» knowledge about issues to existing «processes» in a company.

Note that most of these improvement strategies has their main goal as «optimising» how work is done, in the spirit of scientific management. Another approach to be more productive is not to try to optimise, but to «improvise» (Dybå, 2000). We will come back to this point later.

2.4 Knowledge Management and Learning Organisations

Another recent improvement «trend» has been knowledge management, which is also related to creating «learning organisations», in software engineering: «learning software organisations».

Objectives of knowledge management might be «to make the enterprise act as intelligently as possible to secure its viability and overall success» (Wiig, 1997). If we look more into knowledge management, we find that some important aspects are (Wiig, 1995):

- Survey, develop, maintain and secure the intellectual and knowledge resources of the enterprise.
- Determine the knowledge and expertise required to perform work tasks, organize it, make the requisite knowledge available, «package it» and distribute it to the relevant points of action.
- Provide (...) a knowledge architecture so that the enterprise's facilities, procedures, guidelines, standards, examples, and practices facilitate and support active Knowledge management as part of the organization's practices and culture.

An example of a knowledge management tool is the COIN Experience Factory, an Intranet tool developed in the Fraunhofer Institute for Experimental Software Engineering (Tautz, 2000). This tool allows researchers to search in a database of experience gathered from previous

projects. This experience has been gathered through in-depth interviews with project participants, and then structured according to topic.

We will discuss the term knowledge management in depth in chapter 3.

Another holistic approach, which includes organisations and technology in improvement, has been to create «learning organisations». A learning organisation is «an organisation skilled at creating, acquiring, and transferring knowledge, and at modifying its behaviour to reflect new knowledge and insight» (Garvin, 1993). George Huber gives some advice on what managers can do to make their organisations more «learning» (Huber, 1996):

- Learn from experience - systematically capture, store, interpret and distribute relevant experience gathered from projects; and also to investigate new ideas by carrying out experiments.
- Learn by watching and listening - make people act as «sensors» to learn on behalf of the organisation by participating in communities and reading relevant information.
- Using a computer-based organisational memory - to capture knowledge obtained from experts to spread it through the organisation.

A research area that is linked to organizational learning is research on «communities of practise» as a basis for learning. Etienne Wenger writes: «learning is an issue of sustaining the interconnected communities of practise through which an organization knows what it knows» (Wenger, 1998).

In the much-cited book on learning organisations, The Fifth Discipline (Senge, 1990), we find further characteristics of learning organisations: the ability of «systems thinking» - to see more than just parts of a system. This often means to involve people in an organisation to develop a «shared vision», some common grounds that make the work meaningful, and also serve to explain aspects that you yourself do not have hands-on experience in. Another way of improving communication in an organisation is to work on «mental models» that support action, «personal mastery»; that people make use of their creativity and abilities. And finally «group learning» - to enhance dialogue and openness in the organisation.

Many researchers use the word «best practise» in relation to knowledge management - implying that the aim is to transfer the «best practise» from some people to the whole organisation. We think that this is both difficult and unwanted in most organisations: To «capture» a «best practise» and make it applicable in different settings, you need to capture the context, and this makes it expensive. We also think that such descriptions of process models can better serve as «good examples» that is not necessarily the «best» way to develop software.

Another misconception about knowledge management is that it is synonymous to «reuse» or «replication». This can be a part of knowledge management, but then more as «getting to know which artefacts that are reusable», and probably modify it to suit a specific need than to re-apply an old piece of work.

2.4.1 Why is Knowledge Management a Good Approach?

After having seen some different possible solutions to some of the common problems in software engineering, why would we suggest another one like knowledge management? Let us first discuss why this approach is relevant for software companies, and why it is interesting as a research topic.

Our main argument why knowledge management is a good solution to common problems in software engineering is that software development is knowledge-intensive work, and knowledge-intensive work can be improved by managing knowledge better. We claim that software engineering is knowledge-intensive because:

1. To develop software requires deep technical knowledge in many specific domains.
2. The required knowledge is changing because of technological changes, and because the market wants new solutions.

So, it requires knowledge both to do a good job, and also to cope with rapid changes in both technology and needs in an application domain. Then we reach our second step in the argument: Knowledge intensive-work can be improved by managing knowledge better, because:

1. Work that requires knowledge can be done better if you know that the knowledge is relevant and up to date, which requires learning.

2. To ensure that you learn relevant knowledge, it is best to learn from you own environment, which is the essence of knowledge management. This also means that you «try to make the best out of the resources you have available already».
3. To improve knowledge work, we need a holistic approach with both technical and organisational aspects. People learn better when they are motivated to do so.
4. Focusing on managing knowledge will activate local knowledge that exists in a company.

Some knowledge is easier to transfer to others if it is written down, like in a (possibly) formal document. Frederik Brooks writes about this in his book The Mythical Man-Month about software development, where he recommends that «no matter how small the project, however, the manager is wise to begin immediately to formalize at least mini-documents to serve as his database» (Brooks, 1995).

Of course, many companies are interested in having knowledge from employees written down - to make it easier to replace the employees if they leave for another company, or another position internally. This is an issue that can make normal employees sceptic to knowledge management, as this can reduce their «value» in the company. However, we can also expect the contrary to be the case: that employees that are good at sharing knowledge with others become even more valuable for a company than before.

We think that knowledge management is a promising set of methods and tools, that could help knowledge workers in performing their job better, and that will probably be used in many different occupations in the future. It seems that the last years' focus on knowledge management has made a business climate for learning, and even learning «on the job». The field of knowledge management is also a truly interdisciplinary arena, where many communities including artificial intelligence, organisational development, software engineering, pedagogy and psychology meet.

Many software developers have long workdays and stress because of the complexity of software development and short time limits. We see knowledge management as one way to make the workday better for developers, by giving a better overview of the work situation, as well as helping people to be more effective.

Knowledge management is a field dominated by a lot of hype and a mixture of theory and technology from different research fields. It can be difficult to understand the different knowledge management initiatives. Especially in software engineering, where technology from artificial intelligence and software development meet with theories from consulting companies, the word «knowledge management» can have a lot of meanings. We think there is a great need to clarify which approaches exist in this domain, and relate different theories and technologies to each other.

The field of knowledge management has been criticised as the next «fad to forget people» (Swan et al., 1999). It has been criticised on four points: (we will introduce the words codify and tacit knowledge later) «overstating the codifiability of knowledge», «overemphasise the utility of new ITs for improving organizational performance improvements», making «unjustified assumptions about the willingness of employees to use such IT systems», and finally not seeing that «the codification of tacit knowledge into formal systems may generate its own pathology; the informal and locally situated practises that allow the firm to cope with uncertainty, may become rigidified by the system». We agree with this criticism in that it is vital to keep in mind what knowledge management is used for, and that it is neither wanted or economical to «downskill» all activities in a company by writing down required knowledge. We view knowledge management as a field that is open to combine lessons learned from social science as well as technology fields.

A problem with many previous studies of knowledge management in software engineering is that the work relies on work descriptions and «methods» that you find in manuals in the companies. But as it has been pointed out in ethnographic studies, there are often a lot of differences between how people really work, and how the organisation describes the work that is done (Brown and Duguid, 1991). Therefore, we think it is a great need for more empirical studies in this field.

Why did we choose to study knowledge management in the software engineering domain?

The «science» or «art» of software development is relatively immature, and so there is a huge potential for learning, as we have seen from the introductory sections. Also, software development is a large industry,

where much effort is spent on improvement initiatives. Another reason for choosing this domain, is that people who work in software engineering are used to make use of computerized tools - they often spend most of their workday in front of a computer, either in their office, or at a customer. These people will probably be the first to use tools that will reach a wider audience later, both because of the high computer use, but also because this group of people are likely to write new software to help in their own work. Therefore, it is particularly interesting to study the use of knowledge management tools in this domain.

Many of the improvement initiatives in the previous sections have been tried out in real software companies. But, as we will describe later, we are mostly interested in medium-sized companies. So why this fascination with medium-sized companies? First, many companies belong to this category (see Fayad et al., 2090)), and many more such companies appear every year. Second, there have not been many studies of process improvement efforts in this type of companies, and especially efforts related to knowledge management initiatives. This might be due to that very few of these companies have an own research and development department that can cooperate in studies with academia. It is interesting to see what kind of technology and organisation that exists in this type of companies, as they are usually very «lean» and can not afford to develop large internal systems as for larger companies. A third reason is that we have actually participated in research projects with this type of companies, which we will describe later.

Another point is that medium-sized companies quite often use different technical solutions that larger ones. Many large companies use Lous Notes as a tool for knowledge management. The consultancy company Ernst & Young in the UK claims to have more than 1 million documents available on their Intranet in addition to their 5.000 internal Lotus Notes databases (Ezingeard et al., 2000). Medium- sized companies will certainly not have systems of this size, and will probably also use more low-cost solutions like Intranets than larger companies.

But could we not also have looked at small companies? After all, there are more of them than the medium-sized ones. We could have looked at them as well, but think that they do usually not have such a great need for computer systems to share knowledge, because the people working in the company can communicate directly much easier. In general, we

think computer-based tools needs an environment of a certain size to be effective.

2.5 Research Methods in Software Engineering

Three types of validation methods in software engineering are discussed in an article by (Zelkowitz and Wallace, 1998): observational, historical and controlled. We will refer to these as research methods rather than validation methods, as they can be used also for constructing research results and not just for validating research hypothesis. We now present each of these set of methods from Table 2.1, and also add action research as a fourth group. We place special emphasis on the case study as this is a method that we will be using later, and describe a way of data collection in case studies called ethnography, and a method for data analysis called grounded theory.

Table 2.1: Validation Methods for Software Engineering (Modified from (Zelkowitz and Wallace, 1998))

Category	Validation method	Description
Observational	Project	Collect developmend data.
	Case study	Monitor projects in depth.
	Assertion	Use ad hoc validation techniques.
	Field study	Monitor multiple projects.
Historical	Literature	Examine previously published studies.
	Legacy	Examine data from completed projects.
	Lessons	Examine qualitative data from completed projects.
	Static analysis	Examine structure of the product.
Controlled	Replicated	Use different approaches.
	Synthetic	Replicate one factor in laborarory setting.
	Dynamic	Examine developed product.
	Simulation	Use developed product in a simulation.
Action Research	-	Cooperate with industry to reach improvement goals.

2.5.1 Historical Research Methods

Historical methods use secondary or indirect data sources, for example from projects that are already finished. One historical method is literature search, where scientific papers that are publicly available are analysed. Another method is to study documents from previously completed projects, like program source code and documentation. This method is called to study legacy data. Other documents for study could be lessons learned documents, which are often reported after finishing larger projects. A last method is static analysis, where we look at a finished (software) product.

2.5.2 Controlled Research Methods

Controlled methods imply that we have control over important aspects of the environment where the study is performed. A study is replicated if it is performed several times with changing assumptions to see the impact of different variables. If it is difficult to perform a study in a normal environment, we can make an approximation of the environment and

perform a synthetic environment experiment. If the subject under study is a software product, we can use dynamic analysis to test the product under changing conditions. Another method would be to use simulation - to evaluate a product in a simulated environment. For a wider discussion on experiments in software engineering, see (Wohlin et al., 2000).

2.5.3 Action Research

We could also add a new category that we do not find in Zelkowitz and Wallace's overview, which is relying both on observation and on taking a more active part in forming results, namely action research. This is defined as (Greenwood and Levin, 1998) «social research carried out by a team encompassing a professional action researcher and members of an organization or community seeking to improve their situation. Action research promotes broad participation in the research process and supports actions leading to a more just or satisfying situation for the stakeholders. Together, the professional researcher and the stakeholders define the problems to be examined, cogenerate relevant knowledge about them, learn and execute social research techniques, take actions, and interpret the results of actions based on what they have learned».

This is a kind of democratic research view that gives credit to the knowledge that people in normal work have, and also ensures that the research is relevant for the software industry (Avison et al., 1999).

Potential problems with this kind of research is that it can easily be biased, in that everyone is interested in reaching the goals that are set up. Thus, we do not know if the same results would be achieved with another set of researchers, or with other people from the company, or with another company in the same situation. But this kind of research is a way to get interaction with companies in a way that would not be possible if it was not so much in the company's interest.

2.5.4 Observational Research Methods

By observational research methods we mean to collect information about our subject of study without strict control over the environment. As in most research, we have to decide what type of information or data to collect, and a proper way to collect it. Data collection methods can be

through written questionnaires, visual observation, interviews, written reports, logs, etc. We can further distinguish between different observational studies when looking at how the research material is collected. If we simply study a project with no special efforts to gather data, and no interference, we call it project monitoring. If the researchers are more involved in deciding what information should be collected, we call it a case study. If there is no strong distinction between the subjects participating in the study and the researchers, we call it an assertion. This type of studies would increase the possibilities of biased results. If we collect data from several projects, we call it a field study.

Note that this usage of the word «field study» differs from normal usage in social science, where a field study would imply that the researchers actually spend time on working in the field.

2.5.5 A Further Description of The Case Study

A definition of a case study is: «an empirical inquiry that (Yin, 1994):

- Investigates a contemporary phenomenon within its real-life context, especially when
- The boundaries between phenomenon and context are not clearly evident.»

To gather data for a case study, we have several options (see (Seaman, 1999) for an overview of qualitative methods in empirical studies of software engineering). We can use a method called ethnography, which we will first present, and then we present another method for analysing the data called grounded theory.

Ethnography - Using a Variety of Data Sources in a Company Setting

Ethnography is a method for collecting information for a case study which has been used by anthropologists. It is described as «the art and science of describing a group or culture. The description may be of a small tribal group in an exotic land or a classroom in middle- class suburbia. The task is much like the one taken on by an investigative reporter, who interviews relevant people, reviews records, weighs the credibility of one person's opinion against another's, looks for ties to special interests and organizations, and writes the story for a concerned public and for professional colleagues. A key difference, however, is that

whereas the journalist seeks out the unusual - the murder, the plane crash, or the bank robbery - the ethnographer writes about the routine, daily lives of people» (Fetterman, 1998). The main element of ethnographic research is the fieldwork: the researcher should get into the environment that she is intending to study, and gradually start to collect data. Another key element in ethnography is to rely on multiple data sources: Participant observation, questionnaires, interviews, projective techniques, videos, pictures and written material.

The analysis in ethnography is usually concentrated around triangulation - to set different sources of information up against each other, to find patterns of thought or behaviour in the community of study. Other methods include drawing organizational charts, making flowcharts of processes that happen, setting up matrices to compare and contrast data, and to use statistics.

Ethnography has been used to some extent within the Computer Science subfield of Information Systems, as well as in Software Engineering. Some researchers have used ethnography and discourse analysis to investigate how quality procedures are applied by practitioners (Sharp et al., 2000). Perry et. al. report on how software developers actually spend their time in a large company (Perry et al., 1994), which contradicts the waterfall model. Others have written about applying ethnographic methods in the construction of information systems and to analyse the development itself (Beynon-Davies, 1997). This paper also gives a good introduction to ethnography.

Grounded Theory - Building Inductive Theory from Field Data

Grounded theory is one type of qualitative research, which is relying heavily on the data that is collected. It aims to use the data for building theoretical constructions unlike other methods more used in natural sciences where the data is used to evaluate hypothesis that we have. A lot of emphasis is put on the process of coding, that is «The analytic processes through which data are fractured, conceptualised, and integrated to form theory» (Strauss and Corbin, 1998). By data, we here mean transcripts of interviews, observational field notes, videos, journals, memos and other written or pictorial information. Some coding procedures are «open coding» - where information is analysed to find central ideas - «concepts» - which is then used to form theory. The analysis can be

word-by-word, sentence-by-sentence or on a more abstract level to find concepts in complete documents. Another much used technique is to apply «axial coding» - to place concepts along an axis. An example from Strauss and Corbin are how teenagers describe the impact of drugs, from «getting stoned» to «not having any effect».