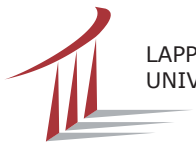


LAPPEENRANTA
UNIVERSITY OF TECHNOLOGY

Ossi Taipale

OBSERVATIONS ON SOFTWARE TESTING PRACTICE

Thesis for the degree of Doctor of Science (Technology) to be presented with due permission for public examination and criticism in the Auditorium of the Student Union House at Lappeenranta University of Technology, Lappeenranta, Finland, on the 26th of October, 2007, at noon.

Acta Universitatis
Lappeenrantaensis
276

Supervisors Professor Kari Smolander
Laboratory of Information Processing
Department of Information Technology
Lappeenranta University of Technology
Finland

Professor Heikki Kälviäinen
Laboratory of Information Processing
Department of Information Technology
Lappeenranta University of Technology
Finland

Reviewers Dr Ita Richardson
Department of Computer Science and Information Systems
University of Limerick
Ireland

Professor Markku Tukiainen
Department of Computer Science and Statistics
University of Joensuu
Finland

Opponent Professor Per Runeson
Department of Communication Systems
Lund University
Sweden

ISBN 978-952-214-428-7
ISBN 978-952-214-429-4 (PDF)
ISSN 1456-4491

Lappeenrannan teknillinen yliopisto
Digipaino 2007

Abstract

Ossi Taipale
Observations on Software Testing Practice
Lappeenranta, 2007
81 p.

Acta Universitatis Lappeenrantaensis 276
Diss. Lappeenranta University of Technology
ISBN 978-952-214-428-7, ISBN 978-952-214-429-4 (PDF)
ISSN 1456-4491

This thesis investigates factors that affect software testing practice. The thesis consists of empirical studies, in which the affecting factors were analyzed and interpreted using quantitative and qualitative methods.

First, the Delphi method was used to specify the scope of the thesis. Secondly, for the quantitative analysis 40 industry experts from 30 organizational units (OUs) were interviewed. The survey method was used to explore factors that affect software testing practice. Conclusions were derived using correlation and regression analysis. Thirdly, from these 30 OUs, five were further selected for an in-depth case study. The data was collected through 41 semi-structured interviews. The affecting factors and their relationships were interpreted with qualitative analysis using grounded theory as the research method. The practice of software testing was analyzed from the process improvement and knowledge management viewpoints. The qualitative and quantitative results were triangulated to increase the validity of the thesis.

Results suggested that testing ought to be adjusted according to the business orientation of the OU; the business orientation affects the testing organization and knowledge management strategy, and the business orientation and the knowledge management strategy affect outsourcing. As a special case, the complex relationship between testing schedules and knowledge transfer is discussed. The results of this thesis can be used in improving testing processes and knowledge management in software testing.

Keywords: software testing, process improvement, knowledge management, survey method, grounded theory method.

UDC 004.415.53 : 65.012.1 : 005.94

Acknowledgements

It has been a privilege to work with great people, companies, and research institutes. I will try to express my gratitude to those people and organizations that have supported me during these years.

Most of all, I want to thank my financers, Tekes (the Finnish Funding Agency for Technology and Innovation, project numbers 40155/04, 40191/05, and 40120/06) and its employees Pekka Yrjölä and Eero Silvennoinen, the companies ABB, Capricode, Delta Enterprise, Metso Automation, Outokumpu, Siemens, and SoftaTest, and the research institutes Helsinki University of Technology, Lappeenranta University of Technology, Tampere University of Technology, and VTT. Without the financial support of Tekes, the participating companies, and the research institutes, this research project would not have been possible.

My supervisors (Professors Kari Smolander and Heikki Kälviäinen) have not spared their efforts. I thank you Kari for your professional guidance, inspiration, and friendship. I thank you Heikki for providing reliable project management, useful advice, and a good working environment.

I want to thank my research team (Olli Hämäläinen, Katja Karhu, Minna Perttula, and Tiina Repo). I am grateful to you for your contribution to this research.

The work of the external reviewers of this thesis, Dr Ita Richardson and Professor Markku Tukiainen, is gratefully acknowledged.

I would also like to thank the steering group of this research project. I appreciate Tiina Kauranen for her professional help in editing the language of this thesis.

Many thanks to my nearest colleagues Sami Jantunen and Uolevi Nikula.

Thank you, my wife Liisa and my children Paula and Olli, for supporting me during this work.

Lappeenranta, 8 May, 2007

Ossi Taipale

List of publications

I. Taipale, O., K. Smolander, H. Kälviäinen (2005). "Finding and Ranking Research Directions for Software Testing", Proceedings of the 12th European Software Process Improvement and Innovation Conference (EuroSPI), 9-11 November 2005, Budapest, Hungary, Lecture Notes on Computer Science 3792, Springer Verlag, pp. 39-48.

II. Taipale, O., K. Smolander, H. Kälviäinen (2006). "Cost Reduction and Quality Improvement in Software Testing", Proceedings of the Software Quality Management Conference (SQM), 10-12 April 2006, Southampton, UK, BCS.

III. Taipale, O., K. Smolander, H. Kälviäinen (2006). "A Survey on Software Testing", Proceedings of the 6th International SPICE Conference on Process Assessment and Improvement (SPICE), 4-5 May 2006, Luxembourg, SPICE User Group, pp. 69-85.

IV. Taipale, O., K. Smolander, H. Kälviäinen (2006). "Factors Affecting Software Testing Time Schedule", Proceedings of the Australian Software Engineering Conference (ASWEC), 18-21 April 2006, Sydney, Australia, Australian Computer Society, IEEE, pp. 283-291.

V. Taipale, O., K. Smolander (2006). "Improving Software Testing by Observing Practice", Proceedings of the 5th ACM-IEEE International Symposium on Empirical Software Engineering (ISESE), 21-22 September 2006, Rio de Janeiro, Brazil, IEEE, pp. 262-271.

VI. Taipale, O., K. Karhu, K. Smolander (2007). "Observing Software Testing Practice from the Viewpoint of Organizations and Knowledge Management", Proceedings of the 1st International Symposium on Empirical Software Engineering and Measurement (ESEM), 20-21 September, 2007, Madrid, Spain, IEEE.

VII. Taipale, O., K. Karhu, K. Smolander (2007). "Triangulating Testing Schedule Over-runs from Knowledge Transfer Viewpoint", Lappeenranta University of Technology, Research Report 104, Finland, pp. 1-35.

VIII. K. Karhu, O. Taipale, K. Smolander (2007). "Outsourcing and Knowledge Management in Software Testing", Proceedings of the 11th International

Conference on Evaluation and Assessment in Software Engineering (EASE), 2-3 April 2007, Keele University, Staffordshire, UK, BCS.

In this thesis, these publications are referred as *Publication I, Publication II, Publication III, Publication IV, Publication V, Publication VI, Publication VII, and Publication VIII.*

Symbols and abbreviations

Agile	Agile software development
AHP	Analytic-hierarchy-process
alpha	Cronbach's coefficient alpha
AM	Agile Modeling
ANSI	American National Standards Institute
ANTI	Name of the research project
ASD	Adaptive Software Development
ATLAS.ti	Name of the qualitative analysis software
CASE	Computer-Aided Software Engineering
CBD	Component-Based Software Development
CBSE	Component-Based Software Engineering
CMM	Capability Maturity Model
COBOL	Common Business-Oriented Language
COTS	Commercial Off-The-Self
Delphi	Name of the research method
df	Degree of freedom
DSDM	Dynamic Systems Development Method
Excel	Name of the spreadsheet software
F	F-test
FDD	Feature-Drive Development

Fortran	Formula Translation/Translator
GAAP	Generally Accepted Accounting Principles
ICT	Information and Communications Technologies
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical & Electronics Engineers
ISD	Internet-Speed Development
ISO	International Organization for Standardization
LB	Like Best technique
Likert	Likert scale
MDD	Model-Driven Development
MES	Manufacturing Execution Systems
N	Number of
NATO	North Atlantic Treaty Organization
OO	Object-oriented
OOD	Object-oriented Design
OU	Organizational Unit
PCA	Principal Component Analysis
PP	Pragmatic Programming
QA	Quality Assurance
R	Coefficient of multiple correlation
R^2	Coefficient of determination
RAD	Rapid Application Development
R&D	Research and Development
SEI	Software Engineering Institute

Sig.	Significance
SME	Small or Medium-sized Enterprise
SP	Structured Programming
SPI	Software Process Improvement
SPICE	Software Process Improvement and Capability dEtermination
SPL	Software Product Line
SPSS	Statistical Package for the Social Sciences
SQA	Software Quality Assurance
SUT	System Under Test
t	t-test
Tekes	Finnish Funding Agency for Technology and Innovation
TMM	Testing Maturity Model
TPI	Test Process Improvement
TTCN	Testing and Test Control Notation
UML	Unified Modeling Language
U Model	Software Development Technologies testing model
VoIP	Voice over Internet Protocol
XP	eXtreme Programming

Contents

1 Introduction

Applications of information and communications technologies (ICT) have penetrated many areas of industry and every day life. The created systems are larger and more critical than ever before. The size and the criticality of the systems among other things emphasize software testing. Kit (1995) states that the systems we build are even more complex and critical, and more than 50% of development efforts is frequently focused on testing.

The research problem of this thesis was derived from Osterweil's (1997) key objectives of software engineering: "software engineering has as two of its key objectives the reduction of costs and the improvement of the quality of products". Software testing as a part of software engineering (ACM et al. 2004) also strives for the reduction of the costs and the improvement of the quality of the products. This thesis studies the question of how to concurrently reduce testing costs and improve software quality. This requires that we first analyze factors that affect the practice of software testing. Understanding the affecting factors and their relationships enables us to develop improvement hypotheses for software testing.

In this thesis the practice of software testing is empirically analyzed from the process improvement and knowledge management viewpoints. Sommerville et al. (1999) introduce the concept of viewpoints to software processes meaning that the observed process is subject to each person's interpretation, or viewpoint. In this thesis the viewpoints are used in a wider context, meaning that software testing is observed and interpreted from the above-mentioned viewpoints. Process improvement and knowledge management were selected as the viewpoints based on the results of the preliminary study. In the preliminary study, experts from industry and research

institutes ranked research issues in software testing. The results of the preliminary study, *Publication I,* showed that the viewpoints process improvement and knowledge management could contain important factors that affect concurrent testing cost reduction and software quality improvement.

Osterweil (1997) writes that processes play a key role in concurrent cost reduction and quality improvement. He emphasizes concurrent cost reduction and quality improvement. Software process improvement (SPI) is considered as one of the central means to make both development and testing processes more effective (Osterweil 1997).

On the other hand, SPI is not free of problems. SPI activities result in organizational changes, which are difficult to implement. Abrahamsson (2001) notes that two-thirds of all organizational change efforts have failed or fallen short of expectations. He emphasizes commitment from all organizational levels because without commitment to SPI, the initiative will most likely fail. Human aspects are important in seeking development and testing efficiency. Osterweil (2003) suggests that the development of a software product is actually the execution of a process by a collection of agents some of which are human, and some of which are tools. Cohen et al. (2004) emphasize that the result of testing ultimately depends on the interpersonal interactions of the people producing the software.

John et al. (2005) state that human and social factors have a very strong impact on software development endeavors and the resulting system. This is in line with the study of Argote and Ingram (2000), who state that there is a growing agreement that organizational knowledge explains the performance of organizations. According to Nonaka (1994), organizational knowledge is created through a continuous dialogue between tacit and explicit knowledge. Tacit knowledge is, for example, embedded in employees. Explicit knowledge is documented and transferable. Knowledge management is regarded as the main source of competitive advantage for organizations (Argote & Ingram 2000; Aurum et al. 1998; Spender & Grant 1996).

Knowledge transfer between development and testing, especially in the earlier phases of the software life cycle, is seen to increase efficiency. Both Graham (2002) and Harrold (2000) emphasize the need to integrate earlier phases of the development process with the testing process. In the same way, modern software development methods (e.g. agile software development) integrate software development and testing. Knowledge transfer is a part of knowledge management. Argote and Ingram (2000) define knowledge transfer in organizations as the process through which one unit (e.g. group, department, or division) is affected by the experience of another. The transfer of knowledge (i.e. routine or best practices) can be observed through changes in the knowledge or performance of recipient units. According to Szulanski (1996), the transfer of organizational knowledge can be quite difficult to achieve. This is because

knowledge resides in organizational members, tools, tasks, and their sub-networks and, as Nonaka and Takeuchi (1995) show, much knowledge in organizations is tacit or hard to articulate.

The special objective of this thesis is to understand the factors that affect concurrent testing cost reduction and software quality improvement. The practice of software testing is described by affecting factors and their relationships. This understanding enables us to generate improvement hypotheses from selected viewpoints.

In this thesis, both quantitative and qualitative methods were applied and the empirical results were triangulated to improve the validity of the thesis. High abstraction level constructs were used because using detailed level constructs might have led to too complicated a description of the practice of software testing. According to the results of the preliminary study, the affecting factors and their relationships were analyzed from the process improvement and the knowledge management viewpoints. Describing the practice of software testing at a high abstraction level is important because, for example, comparing methods, tools and techniques of software testing requires a framework for testing.

The thesis is divided into two parts, an introduction and an appendix including eight scientific publications. In the introduction, the research area, the research problem, and the methods used during the research process are introduced. The appendix contains eight publications. Seven of them have gone through a scientific referee process and *Publication VII* is in the process. The detailed results are given in the publications.

The first part (introduction) contains six chapters. Chapter 2 introduces software testing, viewpoints of the thesis, and describes the history of software testing. Chapter 3 describes the research problem and subject, the selection of the research methods, and the research process. In Chapter 4, the included publications are summarized. Chapter 5 combines the implications of this thesis for the practice and research. Finally, Chapter 6 of the introduction summarizes the whole thesis, lists its contributions, identifies possible limitations, and suggests topics for further research.

2 Software testing and the viewpoints of the thesis

The definition of software testing used in this thesis was adopted from Kit (1995). According to him, software testing consists of verification and validation. By testing, we try to answer two questions: are we building the product right and are we building the right product?

The research problem can be evaluated from different viewpoints. The research work started with the selection of the viewpoints for this thesis. Process improvement and knowledge management were selected as the viewpoints according to the results of the preliminary study. This selection enabled us to concentrate research resources on the issues that respondents evaluated as important.

Software testing and software development are closely related because, for example, approaches, methods, tools, technologies, processes, knowledge, and automation of software development affect testing and vice versa. The 1960s can be regarded as the birth of modern software engineering, when the NATO Science Committee organized software engineering conferences in 1968 and 1969 (Naur & Randell 1969). The 1970s can be regarded as the birth of modern software testing, when Myers published the book, "The Art of Software Testing" (Myers 1976).

The history of software testing offers many examples on, for example, how new development approaches, methods etc. have affected testing. Software testing and its relation to software development is discussed, among others, by Boehm (2006), Jones (2000), Pyhäjärvi et al. (2003), Wheeler & Duggins (1998), Whittaker & Voas (2002), and Whittaker (2000).

2.1 What is software testing?

The literature contains many definitions of software testing. According to Heiser (1997), testing is any technique of checking software including the execution of test cases and program proving. In IEEE/ANSI standards, software testing is defined as:

> (1) The process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component, IEEE standard 610.12-1990 (1990).
>
> (2) The process of analyzing a software item to detect the difference between existing and required conditions (that is, bugs) and to evaluate the features of the software items, IEEE standard 829-1983 (1983).

Further, the IEEE/ANSI 610.12-1990 standard (1990) gives a specification for a test:

> (1) An activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component.
>
> (2) To conduct an activity as in (1).
>
> (3) IEEE standard 829-1983 (1983): A set of one or more test cases, or
>
> (4) IEEE standard 829-1983 (1983): A set of one or more test procedures, or
>
> (5) IEEE standard 829-1983 (1983): A set of one or more test cases and procedures.
>
> The definitions use the term "test case" that is specified as a set of inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specified requirement, IEEE standard 610.12-1990 (1990).

The definition of software testing used in this thesis was adopted from (Kit 1995):

> Testing is verification and validation.

Software testing was defined in this thesis by verification and validation because the definition links software testing neither to any specific software development method nor to a specific life cycle model. Also verification and validation are defined in standards, detectable in the software testing practice, and specified in many quality systems. Verification and validation contain many of the activities of Software Quality

Assurance (SQA) (Pressman 2001). In the following, the contents of verification and validation are discussed.

2.1.1 Verification and validation

Verification ensures that software correctly implements a specific function and it answers the question: are we building the product right? The IEEE standard 610.12-1990 (1990) gives a definition for verification:

> Verification is the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.

Basic verification methods include inspections, walkthroughs, and technical reviews. Checklists are used as the tools of verification. Checklists consist, for example, of requirements, functional design, technical design, code, and document verification checklists. Aurum el al. (2002) describe software inspection methods. Runeson and Thelin (2003) introduce Sample-Driven Inspections. The idea is to select a subset of documents to inspect.

Validation ensures that software that has been built is traceable to customer requirements. It answers the question: are we building the right product? The IEEE standard 610.12-1990 (1990) defines validation:

> Validation is the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.

Validation consists of (1) developing tests that will determine whether the product satisfies the users' requirements, as stated in the requirements specification and (2) developing tests that will determine whether the product's behavior matches the desired behavior as described in the functional specification. Validation activities can be divided into unit testing, integration testing, usability testing, function testing, system testing, and acceptance testing. Runeson (2006) notes that the practices of unit testing vary between companies. In this thesis, unit testing is understood as testing of the smallest unit or units and it is done by developers (Runeson 2006).

Testing methods can be divided into black-box and white-box testing methods. Requirement-based and function-based tests use black-box testing, and technical specification-based tests use white-box testing. In black-box testing, the test cases are derived from the requirements and the functions of the system. The internal structure of the software does not affect test cases. White-box testing is used when the test cases are derived from the internal structure of the software. Validation contains both black-

box and white-box testing methods. Runeson et al. (2006) analyzed existing empirical studies on defect detection methods. Their recommendation is to use inspections for requirements and design defects, and to use validation methods for code.

According to Kit (1995), black-box testing methods for requirements and function-based tests include, for example, equivalence partitioning (identification of equivalence classes and test cases), boundary-value analysis (special case of equivalence partitioning, special interest in boundaries), error guessing (guessing based on intuition and experience), cause-effect graphing (systematic approach to transform a natural-language specification to a formal-language specification), syntax testing (systematic method to generate valid and invalid input to a program), state transition testing (an analytical method using finite-state machines to design tests for programs), and a graph matrix (a representation of a graph to organize the data). White-box testing methods for technical specification-based tests include, for example, statement coverage (each statement is executed at least once), decision (branch) coverage (each decision takes on all possible outcomes at least once), condition coverage (each condition in a decision takes on all possible outcomes at least once), and path coverage (all possible combinations of condition outcomes in each decision occur at least once) (Kit 1995).

Testing can be categorized as functional or structural. In functional testing, test cases are being formed on the basis of specification, and black-box testing is applied. In structural testing, test cases are based on implementation, and white-box testing is applied. Testing can also be categorized as static or dynamic. Static testing includes reviews, walkthroughs, inspections, audits, program proving, symbolic evaluation, and anomaly analysis. Dynamic testing includes any technique that involves executing the software (Heiser 1997).

2.2 The viewpoints of this thesis

Process improvement and knowledge management were selected as the viewpoints of this thesis to concentrate research resources on the issues which experts in the preliminary study evaluated as the most important. Osterweil (1997) defines software processes: “Software processes are software too, suggest that software processes are themselves a form of software and that there are considerable benefits that will derive from basing a discipline of software process development on the more traditional discipline of application software development. Processes and applications are both executed, they both address requirements that need to be understood, both benefit from being modeled by a variety of sorts of models, both must evolve guided by measurement, and so forth.”

Processes are modeled by identifying affecting factors and their relationships. Karlström et al. (2002) use the "analytic-hierarchy process" (AHP) method in rating SPI factors. Factors were identified by a qualitative study and the relationships between factors were also identified. In this chapter, affecting factors that are collected from literature are discussed. Factors affecting testing processes include, for example, the involvement of testing in the development process, the influence of complexity on the testing processes, risk-based testing, testing of software components, outsourcing in testing, and the business orientation of an organizational unit (OU).

Graham (2002) emphasizes the early involvement of testing in the development process, such as testers developing tests for requirements that developers analyze. The involvement of testing in the development process is a complicated issue because the processes glide in parallel. Baskerville et al. (2001) have noticed that software developers run their testing or quality assurance in parallel with other development phases, which results in mutually adjusted processes.

The complexity of testing increases as a function of the complexity of the system under test (SUT). Recent changes in software development include, for example, that systems are larger, they operate on various platforms, and include third-party software components and Commercial Off-The-Self (COTS) software. Increasing complexity of SUTs affects testing processes. Salminen et al. (2000) discuss the strategic management of complexity and divide complexity into four components: environmental, organizational, product, and process complexity.

A trade-off between the scope and schedule of testing affects testing processes and the contents of testing. For example, the risk-based testing approach defines the contents of testing especially in the case of a shortage of resources. The idea of risk-based testing is to focus testing and spend more time on critical functions (Amland 2000).

Component-based development and testing emphasize reuse, and this affects testing processes. Families of similar systems that are differentiated by features are called Product Lines (Northrop 2006). Northrop (2006) addresses strategic reuse as a part of the Software Product Line (SPL) architecture. Torkar and Mankefors (2003) surveyed reuse and testing in different types of communities and organizations. They found that 60% of developers claimed that verification and validation were the first to be neglected in cases of time shortage during a project. This finding on reuse shows that the testing of software components, COTS software, and third party software must be improved before component-based software systems can become the next generation mainstream software systems.

Both design and testing can be outsourced, which affects testing processes. The literature of software testing expresses both advantages and disadvantages of outsourcing (Kaner et al. 1999). Dibbern et al. (2004) have conducted a comprehensive outsourcing survey and an analysis of the literature.

The business orientation and business practices of an organization may cause variation in testing processes. Sommerville (1995) classifies software producers into two broad classes according to their software products: producers of generic products and producers of customized products. In this thesis, Sommerville's (1995) classification was complemented and widened with a finer granulation, and we added a purely service-oriented dimension "consulting and subcontracting" at the other end. In general, OUs can be positioned along a continuum that starts from a purely service oriented OU and ends at a purely product oriented OU, as illustrated in Figure 1.

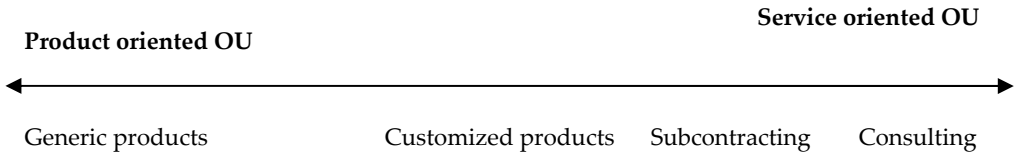


Figure 1. Business orientation

Knowledge management was selected as another viewpoint of the thesis. Nonaka and Takeuchi (1995) state that knowledge and its creation widely affect the competitiveness of an organization. Knowledge is recognized as the principal source of economic rent and competitive advantage (Argote & Ingram 2000; Spender & Grant 1996). According to Edwards (2003), knowledge is central in software engineering and knowledge management is connected to different tasks of software engineering.

Knowledge can further be divided into explicit and tacit knowledge. The objective of knowledge management is to ensure that the right people have the right knowledge at the right time (Aurum et al. 1998). According to Hansen et al. (1999), knowledge management strategies consist of codification and personalization strategies, as illustrated in Figure 2. In a codification strategy, knowledge is codified (explicit) and available in, for example, databases. In a personalization strategy, knowledge is tacit and embedded in employees.

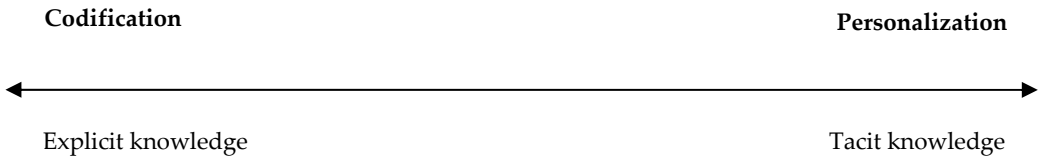


Figure 2. Codification and personalization

Knowledge is further associated with knowledge transfer, which is discussed in the literature, for example by (Becker & Knudsen 2003; Cohen et al. 2004; Conradi & Dybå 2001; Szulanski 1996). Becker and Knudsen (2003) discuss barriers to and enablers of knowledge transfer and divide knowledge transfer into intra-firm and inter-firm transfer. According to them, intra-firm knowledge flows take place within an organization from management to employees (vertical) or between colleagues (horizontal). Inter-firm knowledge flows may lead to downstream (with customers) or upstream (with suppliers, universities and other organizations) knowledge flows (vertical or horizontal), that is, between organizations in competitive interaction.

Szulanski (1996) explored the internal stickiness of knowledge transfer and found that the major barriers to internal knowledge transfer were knowledge-related factors such as the recipient's lack of absorptive capacity, causal ambiguity, and an arduous relationship between the source and the recipient. Conradi and Dybå (2001) studied formal routines to transfer knowledge and experience and concluded that formal routines must be supplemented by collaborative, social processes to promote effective dissemination and organizational learning. Cohen et al. (2004) have noticed that the physical distance between development and testing may create new challenges for knowledge transfer. They found in exploring the organizational structure that testers and developers ought to co-locate. When testers and developers worked in separate locations, communication, as well as personal relationships, was impaired, unlike when both groups worked in close proximity.

The knowledge management strategy affects knowledge transfer. Knowledge is transferred in a personalization strategy through personal interaction. If the knowledge has many tacit elements, transferring it may need many transactions (Nonaka 1994). Codified information is reusable, but creating codified knowledge is expensive because of codification methods and tools (Foray 2004). According to Foray (2004), it is no longer necessary to develop knowledge internally, for it can be bought; this effect is at the root of the growing trend toward outsourcing in many industries.

Testing tasks can be divided into scripted testing and exploratory testing. Scripted testing consists, for example, of running test cases and reporting (Tinkham & Kaner 2003). Extreme scripted testing enables testing automation. In exploratory testing, a tester actively controls test planning, runs the tests, and uses the gained information in planning better tests (Bach 2003). According to Tinkham and Kaner (2003), all testers use exploratory testing to a certain extent. Scripted testing emphasizes explicit knowledge and exploratory testing emphasizes tacit knowledge, as illustrated in Figure 3.

Figure 3. Scripted and exploratory testing

2.3 The history of software testing

The history of software testing connected with the history of software engineering helps to understand how the practice of testing has evolved. In the following, the events that have affected testing are highlighted.

In the 1950s, testing was regarded as debugging that was performed by developers. Testing focused on hardware. Program checkout, debugging, and testing were seen as one and the same. In the late 1950s, software testing was distinguished from debugging and became regarded as detecting the bugs in the software (Kit 1995). The phrases "make sure the program runs" and "make sure the program solves the problem" described software testing in late fifties (Hetzel & Gelperin 1988).

During the 1960s, testing became more significant because the number, cost, and complexity of computer applications grew (Boehm 2006). Programming languages became more powerful. Compiling programs was difficult and time consuming because of the lack of personal compilers (Whittaker & Voas 2002). The code and fix approach became common (Boehm 2006). Faulty programs were released if there was no time to fix them. "If you can't fix the errors, ignore them" (Baber 1982). During this era, the infrastructure of development and testing was improved because of powerful mainframe operating systems, utilities, and mature higher-order languages such as Fortran and COBOL (Boehm 2006).

The era from 1970 to 1979 can be regarded as the birth of modern software testing. During this era, coding became better organized and requirements engineering and design were applied (Boehm 2006). The Structured Programming (SP) movement emerged. The "Formal methods" branch of Structured Programming focused on program correctness, either by mathematical proof or by construction via a "programming calculus" (Boehm 2006). Myers (1976) defined testing as the process of executing a program with the intent of finding errors. Cases of "fully tried and tested" software were found to be unusable (Baber 1982). The focus of testing was more code-centric than quality centric (Whittaker & Voas 2002).

During the 1980s, Computer-Aided Software Engineering (CASE) tools, development of standardization, capability maturity models (CMM), SPI, and object-oriented (OO) methods affected the development of software testing. The idea of CASE tools was to create better software with the aid of computer assisted tools (Whittaker & Voas 2002). This development led to testing tools and automation (Berner et al. 2005; Dustin et al. 1999; Poston 1996) and improved software testing techniques (Beizer 1990). Testing tools and automatic testing appeared during the decade (Mantere 2003).

During the 1980s, various standards and capability maturity models were created. The Software Engineering Institute (SEI) developed software CMM (Paulk et al. 1995). The software CMM content was largely method-independent, although some strong sequential waterfall-model reinforcement remained (Boehm 2006). A similar International Organization for Standardization (ISO) ISO-9001 standard for quality practices applicable to software was concurrently developed, largely under European leadership (Boehm 2006). Quality systems were created based on capability maturity models and standards. Quality systems defined SQA and further testing. IEEE/ANSI standards were published for software test documentation, IEEE standard 829-1983 (1983), and for software unit testing, IEEE standard 1012-1986 (1986) (Hetzel & Gelperin 1988). Osterweil's paper "Software Processes are Software Too" (Osterweil 1987) directed the focus on SPI. SPI improved productivity by reducing rework (Boehm 2006). The SPI approach connected development and testing processes. During the 1980s, the way of thinking changed in software testing. The purpose of testing was no longer to show that the program has no faults but to show that the program has faults (Hetzel & Gelperin 1988).

During the 1990s, the most influential factors included OO methods, SPI, capability maturity models, standards, and automatic testing tools. According to Boehm (2006), OO methods were strengthened through such advances as design patterns, software architectures and architecture design languages, and the development of the Unified Modeling Language (UML) (Jacobson et al. 1999). Dai et al. (2004) explain, for example, how to proceed from design to testing using UML description.

OO methods, SPL architecture (Northrop 2006), reusable components etc. caused a transition from the waterfall model to models providing more concurrency, such as evolutionary (spiral) models. Concurrent processes were emphasized during the 1990s because OO methods and evolutionary software life cycle models required concurrent processes and the sequential processes of the waterfall model were no longer applicable (Boehm 2006). According to Whittaker and Voas (2002), better technical practices improved software processes: OO methods, evolutionary life cycle models, open source development, SPL architecture etc. motivated reuse-intensive and COTS software development. During the 1990s emerged Component-Based Software Engineering (CBSE) and COTS components (Whittaker & Voas 2002). The evolution of

software engineering emphasized testing of OO systems and components, and regression testing because of repetitive testing tasks. The application of CMM led to developing a Testing Maturity Model (TMM) (Burnstein et al. 1996). TMM described an organization's capability and maturity levels in testing. Testing was recognized as more important, which led to the development of testing tools (Kit 1995). The automatic completion of test cases was an important issue because of the growing number of needed test cases (Katara 2005).

OO systems introduced new fault hazards and affected testing. The testing methods of procedural programming are almost all applicable when testing OO systems, but the testing of OO systems creates new challenges (Binder 2001). New fault hazards of OO languages are listed in Table 1.

Table 1. New fault hazards of OO languages according to Binder (2001)

Dynamic binding and complex inheritance structures create many opportunities for faults due to unanticipated bindings or misinterpretation of correct usage.
Interface programming errors are a leading cause of faults in procedural languages. OO programs typically have many small components and therefore more interfaces. Interface errors are more likely, other things being equal.
Objects preserve state, but state control (the acceptable sequence of events) is typically distributed over an entire program. State control errors are likely.

In the 21st century, agile methods have formed the prevailing trend in software development. The continuation of the trend toward Rapid Application Development (RAD) and the acceleration of the pace of changes in information technology (internet related software development), in organizations (mergers, acquisitions, startups), in competitive countermeasures (national security), and in the environment (globalization, consumer demand patterns) have caused frustration with heavyweight plans, specifications, and documentation and emphasized agile software development (Boehm 2006). Agile methods have offered solutions for light-weight software development (Whittaker & Voas 2002).

According to Abrahamsson et al. (2003), the driving force to apply agile methods comes from business requirements, such as lighter-weightiness, fast reaction time, and tight schedules. Software development projects that apply agile methods react fast to changes in business and technology. Agile methods fit especially small and fast reacting software development teams and projects where the schedule is short, requirements change often, the criticality of the products is under average, and when it is important to publish the product before competitors.

Agile software development emphasizes value-prioritized increments of software. According to Boehm (2006), the value-based approach (Value-Based Software

Engineering (VBSE) also provides a framework for determining which low-risk, dynamic parts of a project are better addressed by more lightweight agile methods and which high-risk, more stabilized parts are better addressed by plan-driven methods. Such syntheses are becoming more important as software becomes more product-critical or mission-critical while software organizations continue to optimize on time-to-market (Boehm 2006).

The comparison of traditional and agile software development is derived from Nerur et al. (2005). The comparison is summarized in Table 2.

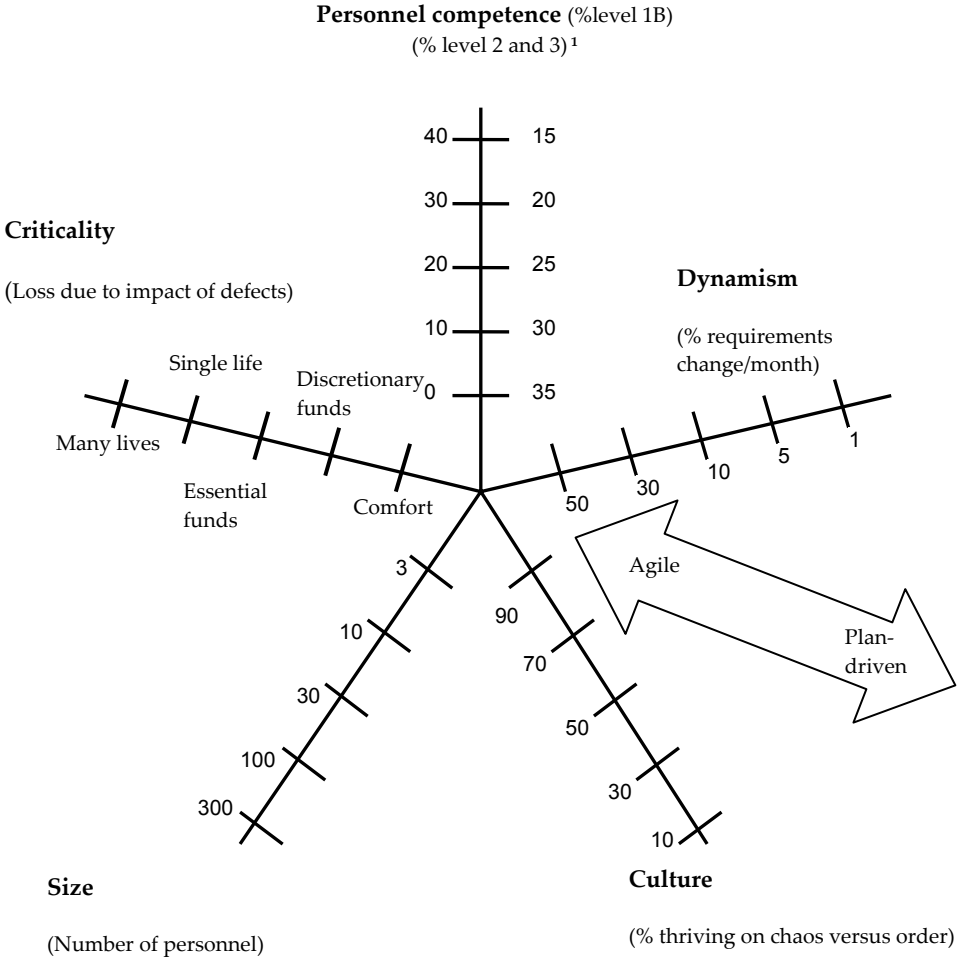
Table 2. Traditional versus agile software development according to Nerur et al. (2005)

	Traditional	**Agile**
Fundamental Assumptions	Systems are fully specifiable, predictable, and can be built through meticulous and extensive planning.	High-quality, adaptive software can be developed by small teams using the principles of continuous design improvement and testing based on rapid feedback and change.
Control	Process centric	People centric
Management Style	Command-and-control	Leadership-and-collaboration
Knowledge Management	Explicit	Tacit
Role Assignment	Individual - favors specialization.	Self-organizing teams – encourages role interchangeability.
Communication	Formal	Informal
Customer's Role	Important	Critical
Project Cycle	Guided by tasks or activities.	Guided by product features.
Development Model	Life cycle model (waterfall, spiral, or some variation)	The evolutionary-delivery model
Desired Organizational Form/Structure	Mechanistic (bureaucratic with high formalization)	Organic (flexible and participative encouraging cooperative social action)
Technology	No restriction	Favors OO technology

Agile methods contain numerous methods and the knowledge of their suitability and usability is insufficient. Agile methods include, for example, Adaptive Software Development (ASD) (Highsmith 2000), Agile Modeling (AM) (Ambler 2002), the Crystal Family (Cockburn 2000), the Dynamic Systems Development Method (DSDM) (Stapleton 1997), Extreme Programming (XP) (Beck 2000), Feature-Drive Development (FDD) (Palmer & Felsing 2002), Internet-Speed Development (ISD) (Baskerville et al.

2001; Cusumano & Yoffie 1999), Pragmatic Programming (PP) (Hunt & Thomas 2000), and Scrum (Schwaber & Beedle 2002).

The applicability of agile and plan-driven methods depends on the nature of the project and the development environment. Boehm and Turner (2003) have developed a polar chart that distinguishes between agile methods and plan-driven methods (Figure 4). The five axes of the polar chart represent factors (personnel, dynamism, culture, size, and criticality) that according to them make a difference between these two approaches.



[1]Levels 1B, 2, and 3 describe Cockburn's Three Levels of Software Understanding. The higher levels 2 and 3 express expertise. Cockburn's scale is relative to the application's complexity. A developer might be a Level 2 in an organization developing simple applications, but a Level 1A in an organization developing high complexity applications.

Figure 4. Agile versus plan-driven methods (Boehm & Turner 2003)

According to Nerur et al. (2005), agile methods favor OO technology, tacit knowledge management, and informal communication. Agile software development affects testing processes and knowledge management. For example, the XP method (Beck 2000) contains a process where acceptance test cases are implemented before programming. Also XP requires an automated testing environment.

After the turn of the millennium, Component-Based Software Development (CBD) and reuse have formed another trend in software development. The objective of the CBD is to save development time and costs, and produce higher quality software because of tested components. A central point is work avoidance through reuse (Boehm 2006). Components consist, for example, of COTS, third party, and open source components. Components include, for example, methods, classes, objects, functions, modules, executables, tasks, subsystems, and application subsystems. CBD is expanding rapidly. It is commonly claimed (e.g. Cai et al. 2005) that component-based software systems are the next generation mainstream software systems. The use of components is expected to shorten development time (Brown 2000).

COTS and open source software components support the rapid development of products in a short time. The availability of COTS systems is increasing (Boehm 2006; Whittaker & Voas 2002). The COTS integration and testing challenges increase because COTS vendors differentiate their products. Enterprise architectures and Model-Driven Development (MDD) offer prospects of improving the compatibility of COTS by increasing pressure on COTS vendors to align with architectures and participate in MDD (Boehm 2006).

Testing products of MDD leads to model-based testing (Katara 2005). MDD and model-based testing offer tools to improve the quality and compatibility of component-based systems. Model-based testing is discussed, among others, by Pretschner et al. (2005). The growing use of COTS, open source, and third party components emphasizes testing of, for example, components, interfaces and integrations. The use and testing of components is discussed, for example, by Cai et al. (2005) and Voas (1998). Testing of component-based systems is challenging. Boehm (2006) notes that components are opaque and difficult to debug. They are often incompatible with each other due to the need for competitive differentiation. They are uncontrollably evolving, averaging about 10 months between new releases, and generally unsupported by their vendors after 3 subsequent releases.

2.4 Summary

In this thesis, software testing, i.e. verification and validation, is evaluated from the process improvement and knowledge management viewpoints. According to the history of software testing, the evolution of testing has followed the pace of changes in software engineering. This is natural because software testing is a part of software engineering. Component-based software systems (e.g. Cai et al. 2005) and agile software development seem to be rising trends in software development. In testing, this evolution means testing of component-based systems and testing with agile software development in addition to testing systems based on plan-driven methods. Both CBSE and agile software development emphasize the development of testing processes and knowledge management because they affect the testing process contents and knowledge management. CBSE requires, for example, testing of components. Agile software development requires, for example, the implementation of test cases before programming and emphasizes tacit knowledge (Nerur et al. 2005).

3 Research goal and methodology

To approach the research problem, how to concurrently reduce testing costs and improve software quality, it was further decomposed into sub-problems and discussed in respective publications. The objective of the first sub-problem was to specify the viewpoints of the thesis. The objective of the other sub-problems was to identify affecting factors and derive improvement hypotheses by analyzing the research subject from selected viewpoints with quantitative and qualitative methods.

Software testing and related software development in organizations formed the research subject. The standards ISO/IEC 12207, Software Life Cycle Processes (2001), ISO/IEC 15504-5, An Exemplar Process Assessment Model (2004), and ISO/IEC 15504-1, Concepts and Vocabulary (2002) explained how we initially understood the research subject. In this thesis, we use the terms software development and testing. The process contents of these terms were adopted from the ISO/IEC 12207 (2001) and 15504 (2004) standards. The objective was to select internationally accepted standards that explain the state-of-the-art of software development and testing. The research process consisted of three phases: preliminary study (viewpoints of the thesis), quantitative studies, and qualitative studies.

In the selection of the research methods, the objective was to find the best method to approach the research subject. For the preliminary phase of the thesis, we selected a Delphi derivative research method (Schmidt 1997). The survey method (Fink & Kosecoff 1985) was used as the research method in the quantitative phase of the thesis and the grounded theory method (Strauss & Corbin 1990) served as the research method in the qualitative phase of the thesis.

3.1 The research problem

The research problem arose from author's own observations in the software development projects and that according to literature more than 50 % of development efforts is frequently focused on testing (Kit 1995). The research problem, how to concurrently reduce testing costs and improve quality, was decomposed into sub-problems. The specification of the viewpoints of the thesis (sub-problem 1) was needed to specify the scope of the thesis. Sub-problems 2 and 3 were used in the quantitative analysis of the software testing practice. Sub-problems 4 (quantitative analysis) and 7 (qualitative analysis) concerned the emergent special question of how testing schedule over-runs are associated with knowledge transfer between development and testing. Sub-problems 5, 6, and 8 were used in the qualitative analysis of the software testing practice. Objectives of the individual studies in this thesis were derived from the specified sub-problems. Sub-problems, objectives of the studies, and the respective publications are listed in Table 3.

Table 3. Decomposition of the research problem

Sub-problem	Objective of the study	Publication
1. Which are the viewpoints of the thesis?	Specification of the viewpoints of the thesis.	*Publication I*
2. Which factors reduce testing costs and improve software quality?	Identification and decomposition of factors that affect testing cost reduction and software quality improvement.	*Publication II*
3. What are the cross-sectional situation and improvement needs in software testing?	Current situation and improvement needs in software testing.	*Publication III*
4. How is knowledge transfer between development and testing associated with testing schedule over-runs? (emergent special question)	Statistical analysis of the factors affecting the software testing time schedule.	*Publication IV*
5. How to improve software testing efficiency from the SPI viewpoint?	Analysis of the practice of software testing from the process improvement viewpoint.	*Publication V*
6. How to improve software testing efficiency from the knowledge management viewpoint?	Analysis of the practice of software testing from the knowledge management viewpoint.	*Publication VI*
7. How is knowledge transfer between development and testing associated with testing schedule over-runs? (emergent special question)	Analysis of the factors affecting the software testing time schedule.	*Publication VII*
8. What is the association between testing outsourcing and knowledge management?	Analysis of the associations between testing outsourcing and knowledge management.	*Publication VIII*

3.2 Research subject and the selection of the research methods

This thesis uses a mixed methods approach consisting of quantitative and qualitative analyses. Combining quantitative and qualitative analyses is based on methodological pluralism. Methodological pluralism means that there is no one "correct" method of science but many possible methods (Hirschheim 1985). In the following, the background that led to the selection of methodological pluralism is explained.

According to Hirschheim (1985), systems that consist of computers and users are, fundamentally, social rather than technical. Thus, the scientific paradigm adopted by the natural sciences is appropriate to such systems only insofar as it is appropriate for the social sciences (Hirschheim 1985). In the selection of the research methodology and paradigm, the above sentence from Hirschheim was used. In this thesis, both technical and social aspects are considered, but without emphasizing the technical or social aspect of the systems. According to Punch (1998), the term "social science" refers to the scientific study of human behavior. "Social" refers to people and their behavior, and to the fact that so much of that behavior occurs in a social context. "Science" refers to the way that people and their behavior are studied.

In methodological pluralism, no single scientific method is placed above other methods, but different scientific methods are used to increase the validity of the study. In this thesis, the scientific methods used consisted of the Delphi, survey, and grounded theory methods. Understanding and interpretation used in the qualitative phase of the thesis included features from hermeneutics. In hermeneutics, understanding and interpretation are essential. Hermeneutical thinking has the same philosophical base as phenomenology, which attempts to capture the essence of things. Positivism and phenomenology are often presented as opposite research philosophies. Positivism is often associated with quantitative study, and phenomenology with qualitative study. Pather and Remenyi (2004) state that positivism or logical positivism, interpretivism or qualitative research, and critical research form the three principal tenets of dominant methodological paradigms in research. According to Järvinen (1999), positivism tends to explain and predict what happens in the social reality by searching for regularities and causal relations between the factors of phenomena. According to interpretivism, the social world can be understood only through the point of view of those individuals who participated in the activities being studied. The critical perspective assumes that knowledge is based on social and historical practices.

Pather and Remenyi (2004) describe critical realism as a combination of positivism, interpretivism, and critical research. According to them, it has become increasingly

obvious that none of these approaches is superior to the others and the best results are achieved by a combination of all of these.

3.2.1 The research subject

In this thesis, the ISO/IEC 12207 and 15504 standards describe how we initially understood and approached the research subject, software testing and related software development in organizations. The ISO/IEC standards were selected because they offered both process and assessment models. The Process Assessment Model 15504-5 (ISO/IEC 2004) uses ISO/IEC 12207 (ISO/IEC 2001) as the Process Reference Model. Also, the ISO/IEC 15504-5 standard (ISO/IEC 2004) was under preparation and we had the possibility to use it afresh.

The life cycle model was derived from the standard ISO/IEC 12207, Software Life Cycle Processes (2001). The assessment model was derived from the standard ISO/IEC 15504-5, An Exemplar Process Assessment Model (2004). The organizational unit (OU) was selected as an assessment unit and it was derived from the standard ISO/IEC 15504-1, Concepts and Vocabulary (2002). The standard ISO/IEC 15504-1 (2002) specifies an OU as a part of an organization that is the subject of an assessment. An OU deploys one or more processes that have a coherent process context and operates within a coherent set of business goals. An OU is typically part of a larger organization, although in a small organization, the OU may constitute the whole organization. The reason to use an OU instead of a company as an assessment unit was that we wanted to normalize the effect of the company size to get comparable data. Another reason to use an OU as an assessment unit was that in a larger company OUs can have different process contents and different business goals.

3.2.2 The selection of the research methods

A Delphi derivative research method (Schmidt 1997) was selected as the research method in the preliminary phase of the thesis because the Delphi method can be used in finding good arguments about an ongoing process. Collecting arguments was the motivation to use the Delphi method in this study. Also, the method generated insights into why respondents view certain issues as being more important than others.

The survey method described by Fink and Kosecoff (1985) was selected as the research method in the quantitative phase of the thesis because a method for gathering information from interviewees was needed, and a survey is a method of collecting information from people about their feelings and beliefs. Surveys are most appropriate when information should come directly from people (Fink & Kosecoff 1985). In a survey, information is collected in a standardized form from a group of persons. According to Pfleeger and Kitchenham (2001) a survey is a comprehensive

system for collecting information to describe, compare or explain knowledge, attitudes and behavior.

The grounded theory research method outlined by Glaser and Strauss (1967) and later extended by Strauss and Corbin (1990) was selected for the qualitative phase of the thesis. Grounded theory was selected because of its ability to uncover the issues from the practice under observation that may not have been identified in earlier literature (Glaser & Strauss 1967).

In a case study, detailed information is collected from a single case or from a small group of related cases. According to Eisenhardt (1989), the theory built from a case study is often novel, testable, and empirically valid. According to Strauss and Corbin (1990), the grounded theory method uses a systematic set of procedures to develop an inductively derived grounded theory about a phenomenon.

The results of the quantitative and qualitative analyses were triangulated to increase the validity of the thesis. The principle of triangulation means that more than one method, observer or data set is used in a study to complement each other and to verify the findings (Denzin 1978). According to Seaman (1999), the combination of quantitative and qualitative methods is usually more fruitful than either in isolation, because statistical relationships found between the quantitative variables can also be checked against qualitative data and vice versa. Paré and Elam (1997) emphasize the mixed method study and write about method triangulation that qualitative data can be used to develop or suggest theoretical arguments which could then be strengthened (or weakened) by quantitative support.

The philosophical baselines can be joined to the empirical study as illustrated in Figure 5. The description is adopted from Hirsjärvi et al. (1997), and our selections are added.

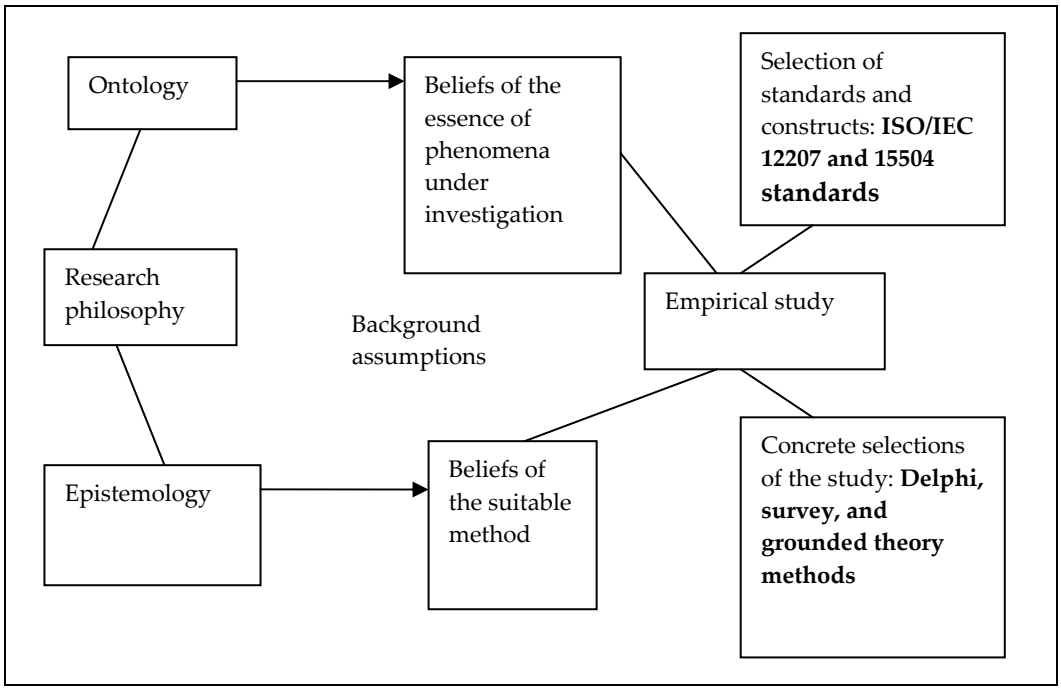


Figure 5. Research philosophy in empirical study according to Hirsjärvi et al. (1997) with our selections

3.3 Research process

The research process was divided into three phases. In the preliminary phase of the thesis, a Delphi derivative method was applied, the quantitative phase of the thesis used the survey method, and the grounded theory method was applied in the qualitative phase of the thesis. The research process together with the phases of the thesis is expressed in Figure 6.

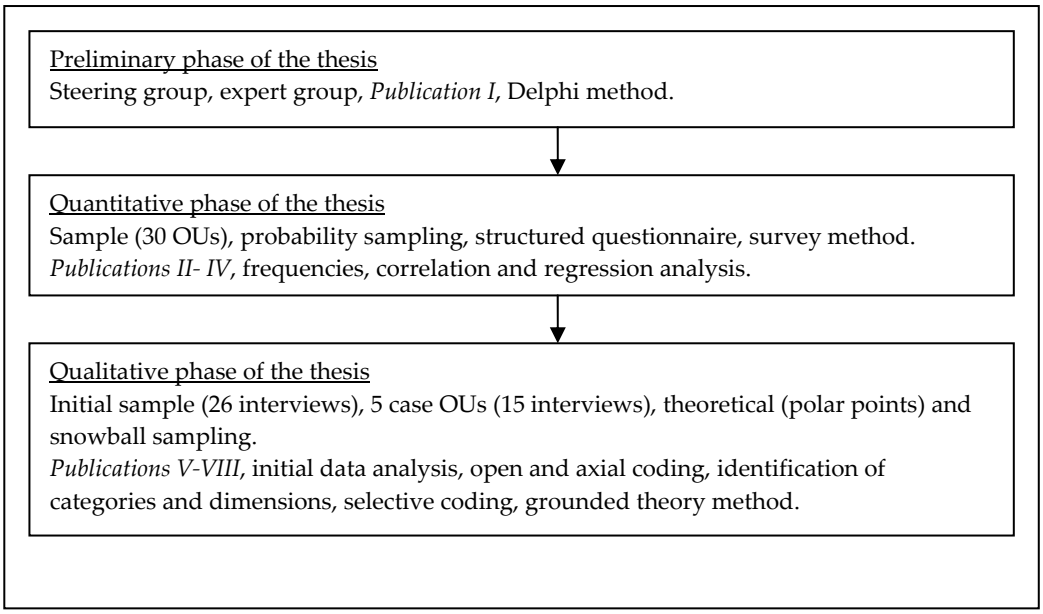


Figure 6. Research process and phases

3.3.1 Delphi method in the preliminary phase of the thesis

During the preliminary phase of the thesis, experts evaluated the relative importance of research directions in software testing using the Delphi method (Schmidt 1997). The objective of the preliminary phase of the thesis was to specify the viewpoints of the thesis. The research process and the results of the preliminary phase of the thesis are reported in *Publication I.*

The phases of the preliminary study were adopted from the Delphi derivative survey method developed by Schmidt (1997). The original Delphi method (Dalkey 1969; Dalkey & Helmer 1963) was developed at the Rand Corporation in the 1950s. The objective of the Delphi method is to achieve the most reliable consensus of opinion of a group of experts. This is accomplished via a series of questionnaires and controlled opinion feedback. Dalkey and Helmer (1963) claim the Delphi method to be more conducive to independent thought than some more conventional uses of experts.

The Delphi method was selected to this study because it is suitable for group decisions, and the objective here was to identify important issues in software testing from experts working together in group meetings. Successful experiences with Schmidt's Delphi derivate method contributed to our choice to apply the Delphi method. Schmidt's Delphi derivative method was successfully used in a study of identifying software project risks performed by Keil & al. (1998). In this study, experienced software project managers identified and ranked the most important risks

in software projects. Not only the risk factors and their relative importance were identified, but also new insights into why some risks are seen as more important than others were gained. Ranking rounds were repeated until either the panelists reached a strong consensus or the consensus did not change from one round to the next. The Delphi survey is designed to elicit opinions from a panel of experts through iterative, controlled feedback. According to Schmidt (1997), in order to help the reader to determine the degree of confidence to place in the findings, the total number of issues generated in the beginning and the strength of their support should be reported, the panel of experts must be well described, and the number of panelists for each round must be reported.

Data collection and analysis in the preliminary phase

The Delphi derivative method applied requires a panel for ranking the issues. The steering group of the research project formed the panel where the important issues of software testing were ranked. The panel (steering group) consisted of representatives of the participating industry and research institutes. In addition, an expert group was assembled.

Schmidt (1997) mentions three phases of data collection in the Delphi method: the discovery of issues, determining the most important issues, and ranking the issues. In the preliminary study, Schmidt's "consolidating the data" was added as a separate phase.

In the first phase (discovering the issues), the data was collected from literature and expert interviews in order to capture as many issues as possible. According to Schmidt (1997), this maximizes the chance to discover the most important issues.

The second phase – consolidating the list – was conducted in three steps. Firstly, larger entities were formed from the issues of the list by researchers. Then this list was introduced to the first panel. New issues were also accepted onto the list. Finally, corrections were made based on the feedback from the first panel. In consolidation the list was compressed from 22 to 9 issues.

In the third phase, the list was given to an expert group for rating to determine the most important issues. The expert group consisted of experts who deal with problems of testing daily. The expert group represented a wider cross-section of the software engineering industry than the steering group. The objective of the expert group was to generate information for the panel by rating the issues of software testing. The experts were independently asked to rate the issues on the list on the Likert scale of 1 to 5 and the results were summed up. This process was similar to the one described by Morris et al. (1998).

The fourth phase (ranking the research issues) consisted of a panel which voted the three most important issues on the list, eliminating the rest. The paring method was adopted from Keil et al. (1998). The panel reached a consensus and selected the winning issue.

3.3.2 Survey method in the quantitative study

In the quantitative phase of this thesis, the survey method (Fink 2003; Fink & Kosecoff 1985) was used. The survey method is used to gather information about feelings, motivations, plans, beliefs, and personal, educational, and financial background. Tools for gathering such information are usually questionnaires or interviews (Fink & Kosecoff 1985). The results of the quantitative analyses are published in *Publications II-IV*.

The survey method was selected as the quantitative research method in identification and decomposition of factors that affect testing cost reduction and software quality improvement (*Publication II*) because open-ended questions offered an insight into why people believe what they do (Fink & Kosecoff 1985). The current situation and improvement needs in software testing were described in *Publication III*. The survey was the natural choice as the research method, because this study explores the cross-sectional situation in software testing. Exploring the cross-sectional situation is the basic form of a survey study (Fink & Kosecoff 1985). The statistical analysis of the factors affecting the software testing schedule was described in *Publication IV.* The survey method offered an efficient tool for data collection. Also successful experiences with software testing surveys, for example, (Ng et al. 2004; Torkar & Mankefors 2003) contributed to our choice to apply the survey as the quantitative research method.

According to Pfleeger and Kitchenham (2001), activities of the survey process include:

1. Setting specific, measurable objectives.
2. Planning and scheduling the survey.
3. Ensuring that appropriate resources are available.
4. Designing the survey.
5. Preparing the data collection instrument.
6. Validating the instrument.
7. Selecting participants.
8. Administering and scoring the instrument.
9. Analyzing the data.
10. Reporting the results.

Interviews can be categorized as structured, theme-based, or open interviews. In the quantitative analysis, structured interviews were used. They were based on a questionnaire consisting of scale-based, multiple choice, and open (open-ended) questions.

The survey questions can be categorized into scale-based questions (e.g. the Likert scale), multiple choice questions, and open questions. The instrument consisted of structured and open questions, instructions, and a database. Multiple choice questions and scale-based questions (Likert scale) were statistically analyzed. According to Fink (2003), answers to open questions may be difficult to compare and interpret but they can provide more detailed information than closed questions.

The reliability and validity of the survey instrument ought to be ensured (Litwin 1995). Fink (2003) writes that a reliable survey instrument is consistent and a valid one is accurate. Fink (2003) emphasizes piloting; a pilot test is an opportunity to try out an instrument well before it is made final. The questionnaire was piloted with three OUs and four private persons. In addition, the members of the project's steering group commented on the form. According to Fink (2003), a survey's internal consistency, or homogeneity, is the extent to which all the items or questions assess the same skill, characteristic, or quality. Internal consistency or homogeneity of multi-item questions was measured by using Cronbach's coefficient alpha (Cronbach 1951), the average of all the correlations between each item and the total score. It was calculated to determine the extent of homogeneity.

According to Fink (2003), a sample is a portion or subset of a larger group called a population. A good sample is a miniature version of the population of which it is a part – just like it, only smaller. The population of this study consisted of OUs that develop and test technical software for automation or telecommunication domains. We applied the standard ISO/IEC 15504-1 (ISO/IEC 2002) to guarantee that the processes are comparable. The standard specifies a process as a set of interrelated activities, which transform inputs into outputs. We judged the comparability of processes by process outputs. We selected OUs of a high technical level producing real time software, the criticality of which was above average.

Sampling methods can be divided into probability sampling and nonprobability sampling. According to Fink (Fink 2003), probability sampling provides a statistical basis for saying that a sample is representative of the study or target population. One form of probability sampling is systematic sampling where, for example, every other or every third item from the population list is selected for the sample.

Data collection

In the quantitative phase of the thesis, a validated survey instrument for data collection and analysis was needed, but such an instrument was not available in the literature. The questionnaire (Appendix II) was based on the process assessment and improvement models, the Software Process Improvement and Capability dEtermination (SPICE) (Emam et al. 1998) described in the ISO/IEC 15504-5 standard, Information Technology – Process Assessment, an Exemplar Process Assessment Model (ISO/IEC 2004) and Test Process Improvement (TPI) (Koomen & Pol 1999), which is a model for testing process assessment and improvement. The TPI model was used as a reference that describes the best practices. Standards ISO/IEC 12207, Information Technology – Software Life Cycle Processes (ISO/IEC 2001) and ISO/IEC 15504 served as the basis for developing the research instrument. The constructs of the instrument were derived from the ISO/IEC 12207 and ISO/IEC 15504 standards, excluding business orientation that was derived from Sommerville (1995), items of communication and interaction (knowledge transfer) that were derived from Suchman (1989), and problems of the testing environment that were taken from software project risks analyses (Boehm 1991; Keil et al. 1998; Standish 1994; Wallace & Keil 2004) and modified to the testing environment. The terms "communication and interaction" and "knowledge transfer" have been used as synonyms in this thesis. The term communication and interaction (*Publications I to V*) was changed to knowledge transfer (*Publications VI to VIII*) because the term knowledge transfer is more compact and more widely used in the literature. Further, knowledge transfer and know-how are discussed as a part of knowledge management.

The sample contained 40 industry experts from 30 OUs. The survey instrument consisted of a structured questionnaire. Respondents were interviewed face-to-face. A few open questions were placed at the end of the questionnaire. The classification of the open answers was planned in advance by creating initial classes. The questionnaire was planned to be answered during the interview to avoid missing answers because they make the data analysis complicated. All the interviews were tape-recorded. The selection from the population to the sample was based on probability sampling. The OUs were in random order in our database and every second organization was selected. Baruch (1999) states that the average response rate for self-assisted questionnaires is 55.6%, and when the survey involves top management or organizational representatives the response rate is 36.1%. In this case, a self-assisted, mailed questionnaire would have led to a small sample. For these reasons, it was rejected and personal interviews were selected.

One person did all of the interviews to minimize the bias caused by different interviewers. Only two OUs refused to give an interview. This was because they felt that the questions concerned classified information. In addition, three OUs were

rejected because they did not fit the population criteria in spite of the source information, and three OUs were excluded because it was impossible to fit the interview into the respondent's schedule. Therefore, the response rate was all in all 79%.

Data analysis

The survey instrument was built by using Excel spreadsheet software. The answers of the interviewees were directly collected into Excel tables and further transformed to SPSS software (2004) format. Statistical analysis was implemented by using SPSS software. The statistical analysis included the calculation of frequencies including basic statistics (e.g. number of occurrences, mean, geometric mean, median, mode, and standard deviation), visual representation of statistics, estimation of the reliability of the multi-item constructs, and correlation and regression analyses.

The objective of the first part of the quantitative phase of the thesis (*Publication II)* was to evaluate from the testing point of view the factors that affect cost reduction and software quality improvement. Both the format of the open-ended questions and the classification of the answers were based on the like best (LB) technique adopted from Fink & Kosecoff (1985). According to the LB technique, respondents were asked to list at least one but no more than three points they considered the most efficient. The results revealed the relative importance of factors that affect software testing efficiency and the decomposition of the affecting factors.

The objective of the second part of the quantitative phase of the thesis (*Publication III)* was to reveal the current situation and improvement needs in software testing. The statistical analysis included the calculation of frequencies (e.g. number of occurrences, mean, geometric mean, median, mode, and standard deviation) and visual representation (e.g. XY scatter, line, column, and bar graphs, and pie and radar charts). In this study we used the mean as the measure of the "central tendency" because the data was collected using an interval scale (Fink & Kosecoff 1985) and it obeyed normal distribution.

The objective of the third part of the quantitative phase of the thesis (*Publication IV)* was to identify and model constructs associated with over-runs in software testing schedules. Constructs concerning communication and interaction were measured as multi-item constructs. The reliability of the multi-item constructs was estimated by using the Cronbach alpha (1951). This process was similar to the one described by Dybå (2000). The Cronbach coefficient alpha expresses the degree to which items in a scale are homogeneous. The calculation was performed using the SPSS software (2004). The alpha expresses the mean reliability coefficient estimate for all possible ways of splitting a set of items in two (Cronbach 1951). Nunnally (1978) states that the coefficient alpha sets an upper limit to the reliability of tests constructed in terms of a

domain-sampling model. If it proves to be very low, either the test is too short or the items have very little in common. The data of the items was collected using an interval scale. The constructs were calculated as the means of the four items which still improved the normal distribution of the constructs.

In the correlation and regression analysis, a Pearson correlation matrix was first calculated between the variables to estimate the constructs associated with testing schedule over-runs. The significance of correlations was calculated at levels 0.05 and 0.01 (2-tailed). In the regression model, the independent variable groups consisted of communication and interaction constructs, product orientation, and problems of the testing environment (control variables). Keeping the testing schedule was selected as the dependent variable. In the regression analysis, stepwise linear regression was used as the variable selection method. R Square statistics were used to express the proportion of the variance of the dependent variable explained by the independent variables. The statistical significance of the model was estimated by the analysis of the variance. Regression coefficients and the significance of the effect of the independent variables on the dependent variable were calculated. The collinearity analysis revealed that the variables were independent. The tolerance values were above 0.2 for all variables, which means that there is no evidence of multicollinearity between the variables.

3.3.3 Grounded theory method in the qualitative study

In the qualitative phase of this thesis, grounded theory (Strauss & Corbin 1990) was used. The baseline in the qualitative research is to describe real life. The research subject is studied as comprehensively as possible. The objective of a qualitative study is rather to find or reveal facts than to prove existing theorems. Strauss and Corbin (1990) define qualitative research as any kind of research that produces findings not arrived at by means of statistical procedures or other means of quantification. Tesch (1990) divides qualitative research methods into four main groups concerning the research interest: the characteristics of language, the discovery of regularities, the comprehension of the meaning of text/action, and reflection. The results of the qualitative analyses are published in *Publications V-VIII*.

Tesch (1990) categorizes grounded theory in the field where the research interest is in the discovery of regularities. The grounded theory among the types of qualitative research is described in Figure 7.

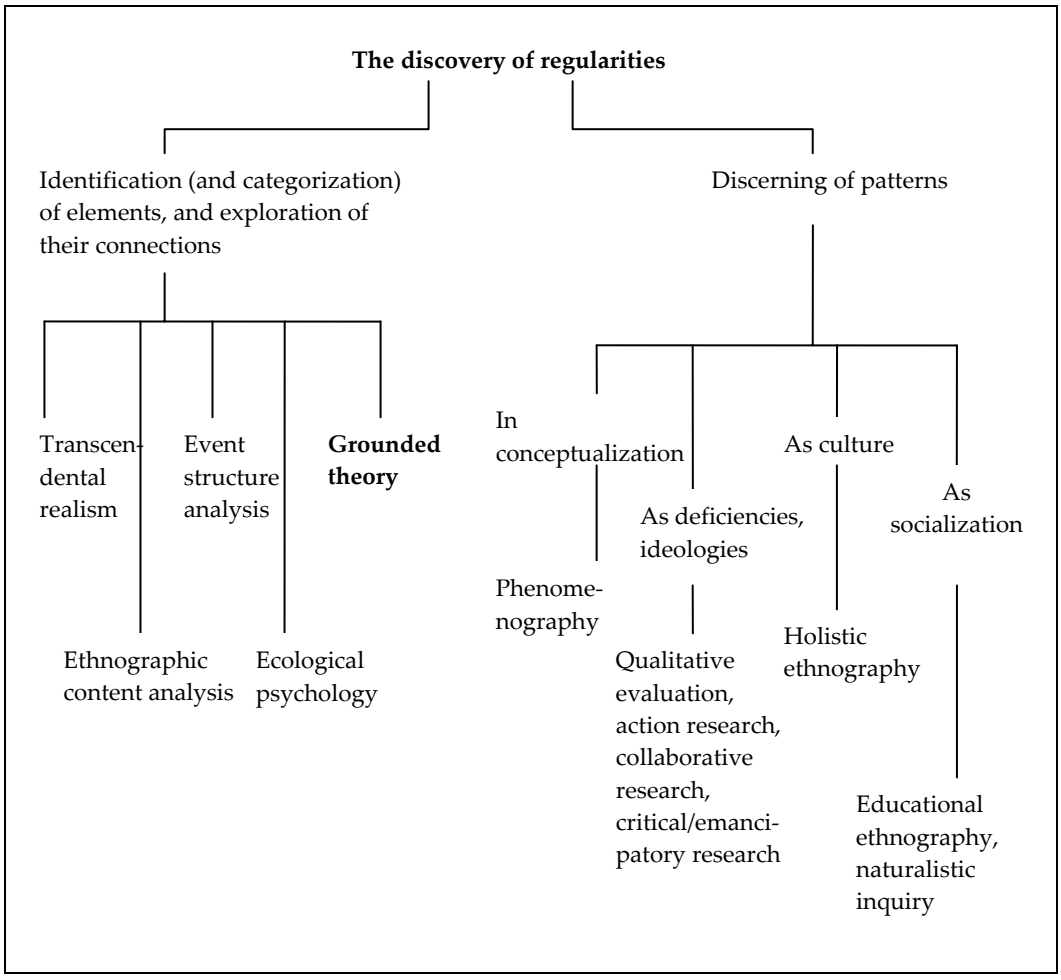


Figure 7. Grounded theory among the types of qualitative research according to Tesch (1990)

In analyzing the practice of software testing (*Publications V, VI, and VIII*) and in analyzing factors that affect the software testing schedule (*Publication VII*), grounded theory was selected as the qualitative research method because it enables the identification of affecting factors and their relationships by grounding observations on the data. Successful experiences with the grounded theory method (Carter & Dresner 2001; Paré & Elam 1997; Smolander et al. 2005) contributed to our choice to apply the method.

Data collection

The beginning of a qualitative (interpretive) study includes the definition of a research problem, possible a priori constructs, the selection of cases, and the crafting of instruments and protocols for data collection (Eisenhardt 1989). The quantitative analysis preceded the qualitative analysis meaning that some candidates or a priori constructs, such as business orientation, were available. According to Eisenhardt (1989), a priori constructs can help to shape the initial design of the research. Inductive theory building research should, however, have no theory and no hypotheses to test.

For the case study, we selected five OUs from among the thirty OUs interviewed during the quantitative phase of the thesis. The sampling was theoretical (Paré & Elam 1997) and the cases were chosen to provide examples of polar types (Eisenhardt 1989), which means that the cases represent different types of OUs, such as different line of business, different size of the company, and different operation. Theoretical sampling (Glaser & Strauss 1967) describes the process of choosing research cases to compare with other cases. The goal of theoretical sampling is not the same as with probabilistic sampling. The researcher's goal is not a representative sample of all possible variations, but gaining a deeper understanding of the analyzed cases and identifying concepts and their relationships. Theme-based questionnaires (Appendix III) served as the instruments for data collection.

The study included four theme-based interview rounds. We personally visited companies and carried out 41 tape-recorded interviews. The interviews were conducted by two researchers. The duration of the interviews varied between one and one and a half hours and they were all tape-recorded and transcribed. A memo containing the emphasized issues was written on each interview.

The first interview round that was completed during the quantitative analysis served also as the first interview round for the qualitative analysis. The first interview round contained both structured and semi-structured (open) questions. The objective of this interview round was to understand the basic practice of testing, identify case OUs (representative polar points) for the next round, and identify problems and improvement proposals. The interviewees were managers of development or testing or both. In some interviews, there was more than one interviewee present, for example a manager of development and a manager of testing. Such interviews usually lasted more than one hour. The questions of the first round concerned general information on the OU, processes, communication and interaction between development and testing, and the development environment of the OU.

The interviewees of the second round were managers of testing. In some interviews, managers of development were also present. The duration of the interviews varied between one and one and a half hours. The objective of the second interview round

was to achieve a deeper understanding of the software testing practice. The questions were theme-based and concerned problems in testing, the utilization of software components, the influence of the business orientation, communication and interaction, schedules, organization and know-how, testing automation, and economy.

The interviewees of the third round were testers and the interviewees of the fourth round were systems analysts. The interviews in these rounds were also theme-based and concerned the work of the interviewees, problems in testing, the utilization of software components, the influence of the business orientation, communication and interaction, schedules, organization and know-how, and testing automation. The interviews lasted about one hour.

The themes of the interview rounds remained similar, but the questions evolved from general to detailed. Before proceeding to the next interview round, all interviews were scripted and coded because new ideas emerged in the coding. These new ideas were reflected on the next interview rounds.

Managers of development and testing, testers, and systems analysts were selected as interviewees because these stakeholders face the daily problems of software testing. The data collection process of all 41 interviews generated a transcription of 946 pages.

Data analysis

The objective of the qualitative studies (*Publications V, VI and VIII*) was to understand the practice of software testing from the points of view of process improvement (*Publication V*), organization and knowledge management (*Publication VI*), and outsourcing and knowledge management (*Publication VIII*). The objective of *Publication VII* was to investigate the emergent special question: the relationship between software testing schedule over-runs and knowledge transfer.

The analysis in grounded theory consists of three types of coding: open coding, where categories of the study are extracted from the data; axial coding, where connections between the categories are identified; and selective coding, where the core category is identified and described (Strauss & Corbin 1990). In practice, these steps overlap and merge because the theory development process proceeds iteratively. The theory was derived inductively from and grounded on the data.

The objective of the open coding was to classify the data into categories and identify leads in the data. The process of grouping concepts that seem to pertain to the same phenomena is called categorizing, and it is done to reduce the number of units to work with (Strauss & Corbin 1990). The open coding of the interviews was carried out using the ATLAS.ti software (ATLAS.ti - The Knowledge Workbench 2005). The open coding process started with “seed categories” (Miles & Huberman 1994) that contained essential stakeholders, phenomena, and problems. Seed categories formed

the initial set of affecting factors. The ISO/IEC standards 12207 (2001) and 15504 (2004) were used in identifying the seed categories. Seaman (1999) notes that the initial set of codes (seed categories) comes from the goals of the study, the research problems, and predefined variables of interest. In the open coding, new categories appeared and existing categories were merged, because especially in the beginning of the coding, new information sprang up. The open coding of all 41 interviews yielded 196 codes which were classified in axial coding into categories according to the viewpoints of the study.

The objective of the axial coding was to further develop categories, their properties and dimensions, and causal conditions or any kinds of connections between the categories. The categories were further developed by defining their properties and dimensions. The dimensions represent the locations of the property or the attribute of a category along a continuum (Strauss & Corbin 1990). The phenomenon represented by a category was given a conceptual name (Strauss & Corbin 1990). Our inductive data analysis of the categories included Within-Case Analysis and Cross-Case-Analysis, as explained by Eisenhardt (1989). We used the tactic of selecting dimensions and properties, and looking for within-group similarities coupled with intergroup differences (Eisenhardt 1989). Each chain of evidence in this interpretation was established by having sufficient citations in the case transcriptions.

The objective of the selective coding was to identify the core category (Strauss & Corbin 1990), a central phenomenon, systematically relate it to other categories, and generate the theory. Strauss and Corbin (1990) write that sometimes the core category is one of the existing categories, and at other times no single category is broad enough to cover the central phenomenon. In that case, the central phenomenon must be given a name. In this study, the creation of the core category meant the identification of the affecting factors (categories) and finding the relationships between these categories.

The general rule in grounded theory is to sample until theoretical saturation is reached. This means until (1) no new or relevant data seems to emerge regarding a category; (2) the category development is dense, insofar as all of the paradigm elements are accounted for, along with variation and process; (3) the relationships between categories are well established and validated (Strauss & Corbin 1990). The theoretical saturation was reached during the fourth interview round because new categories did not appear, categories were not merged, shared, or removed, the attributes or attribute values of the categories did not change, and relationships between categories were stable, i.e. the already described phenomena recurred in the data.

3.3.4 Finishing and reporting the thesis

Pfleeger and Kitchenham (2001) list "reporting the results" as the final phase of the survey process. Results of each quantitative analysis were compared with available software testing survey results. Eisenhardt (1989) defines "enfolding literature" and "reaching closure" as the last phases of the qualitative analysis. Generated hypotheses were compared with hypotheses found in the literature.

Each phase of the thesis answered for its part the research problem, but also raised new questions. In the preliminary phase, the scope of the thesis was delimited to process improvement and knowledge management, *Publication I*.

In the quantitative analysis, respondents evaluated which are the most important factors that affect concurrent cost reduction and quality improvement. Factors affecting efficiency were further decomposed according to respondents' answers into detailed issues (*Publication II)*. Respondents evaluated the current situation and improvement needs in software testing, *Publication III*. The quantitative analysis also raised new questions. Findings included that testing had over-run its schedule (*Publication III*) and the correlation and regression analysis revealed that increased knowledge transfer in the design phase between development and testing was associated with schedule over-runs (*Publication IV*). In the quantitative phase, a qualitative analysis was proposed as the research method for the next phase of the thesis because statistical analysis could not reveal root causes.

In the qualitative phase of the thesis, the factors affecting software testing practice were evaluated from the viewpoints of testing process improvement (*Publication V*), organizations and knowledge management (*Publication VI*), and outsourcing (*Publication VIII*). The statistical analysis together with the qualitative analysis revealed the association between knowledge transfer and testing schedules (*Publication VII*). The progress of the thesis and the relationships between the studies are described in Figure 8.

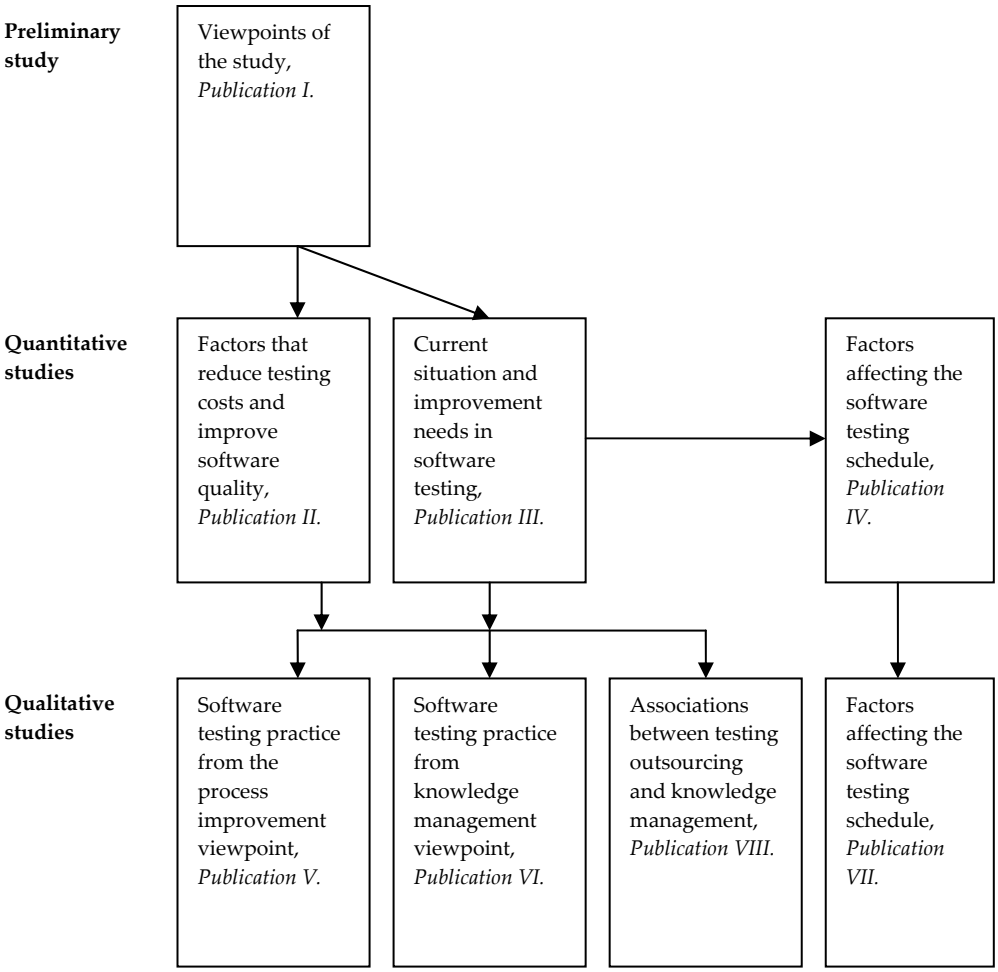


Figure 8. Products and progress of the thesis

3.4 Summary

The research phases and their essential methodical details are summarized in Table 4.

Table 4. The research phases

Phase	Preliminary phase of the thesis	Quantitative phase of the thesis	Qualitative phase of the thesis
Research problem	How to concurrently reduce testing costs and improve software quality? Viewpoints of the thesis.	How to concurrently reduce testing costs and improve software quality? Affecting factors and their relationships.	How to concurrently reduce testing costs and improve software quality? Affecting factors and their relationships.
A priori constructs		Viewpoints of the thesis, software process improvement and knowledge management.	Viewpoints of the thesis, software process improvement and knowledge management. Seed categories from the quantitative study.
Case selection/interviewees	Steering group, expert group	30 OUs	30 OUs and 5 case OUs
Instruments and protocols for data collection	Literature review, interviews	Interviews, structured questionnaire	Interviews, semi-structured questions
Data analysis	Panels, rating, ranking	Statistical analysis with SPSS software.	Qualitative analysis with ATLAS.ti software.
Reporting	*Publication I*	*Publications II-IV*	*Publications V-VIII*

4 Overview of the publications

The results of this research are presented in detail in the appendix consisting of eight publications. These publications have been published separately in scientific conferences, excluding *Publication VII* that is still in the publication process. In this chapter, each of these publications, their objectives, results, and relation to the whole, are discussed. The contents of these publications can be condensed with the following objectives of the studies:

- *Publication I*: Specification of the viewpoints of the thesis.
- *Publication II*: Identification and decomposition of the factors that affect testing cost reduction and software quality improvement.
- *Publication III*: Current situation and improvement needs in software testing.
- *Publication IV*: Statistical analysis of the factors affecting the software testing schedule (emergent special question).
- *Publication V*: Analysis of the practice of software testing from the process improvement viewpoint.
- *Publication VI*: Analysis of the practice of software testing from the knowledge management viewpoint.
- *Publication VII*: Analysis of the factors affecting the software testing schedule (emergent special question).

- *Publication VIII*: Analysis of the associations between testing outsourcing and knowledge management.

In the following, the publications are summarized.

4.1 Publication I: Finding and Ranking Research Directions for Software Testing

4.1.1 Research objectives

The objective of this Delphi derivative (Dalkey 1969; Dalkey & Helmer 1963; Schmidt 1997) study was to reveal and rank important testing research issues and generate insights into why respondents evaluate certain issues as more important than others.

4.1.2 Results

As the result, the panel (steering group) reached a consensus and selected process improvement as the most important issue in software testing research. According to the panel, process improvement increases knowledge transfer between software development and testing. Problems in knowledge transfer between software development and testing may increase both development work and testing work. According to panelists, the software engineering process is under continuous change and this change creates problems in knowledge transfer. Experts explained in the interviews that this on-going change means, for example, minimizing new programming in a company by extensive reuse of software components and by the use of COTS or third party software. An analysis of the knowledge transfer between development and testing processes can reveal important information that can be used in improving the total efficiency of both software testing and development.

4.1.3 Relation to the whole

The results of the preliminary study defined the scope of the thesis by specifying software process improvement and knowledge management as the viewpoints of the thesis. Osterweil (1997) emphasizes process improvement and writes that SPI is considered as one of the central means to make both development and testing processes more effective. On the other hand, the influence of knowledge is essential in economics and competition (Argote & Ingram 2000; Spender & Grant 1996). Edwards (2003) emphasizes knowledge and knowledge management as central in software engineering.

Other viewpoints that emerged, such as testing automation and testing tools, were delimited outside the scope of this thesis. The panel justified this exclusion by stating that testing automation is efficient in repetitive situations (e.g. regression testing) and automation makes it possible to adapt to the demand for the testing capacity. The problem is that automation does not solve the problems of earlier phases in the development process. Quite a deal of research exists in this area and there is not much potential for novel ideas. Also, testing tools for testing automation has been developed since 1980s. To clarify the cross-sectional situation between software development and testing and to get a deeper insight into the problems, an explorative survey of software testing was proposed.

4.2 Publication II: Cost Reduction and Quality Improvement in Software Testing

4.2.1 Research objectives

The objective of this industry survey (Fink & Kosecoff 1985) was to evaluate from the testing point of view the factors that affect cost reduction and quality improvement. To do this, the respondents were first asked to evaluate the cost structure of software testing. Secondly, respondents were asked to evaluate the affecting factors. Thirdly, the affecting factors were further decomposed according to the respondents' answers.

4.2.2 Results

Firstly, the respondents evaluated personnel costs as the highest cost item in testing. Testing automation costs formed another significant cost item group. Other cost items were marginal compared to these two. Osterweil (1997) reports similar results when examining software engineering as a whole. Osterweil suggests cost reduction by reducing labor costs. The reduction of the labor costs often leads to an increase in automation costs. According to our study, the reduction of personnel costs is efficient because they are the highest cost item.

Secondly, the respondents evaluated how to reduce costs and improve software quality. According to our study, the affecting factors included development and testing process improvement, the development of testing automation, the development of testing know-how, the development of testing strategies, the utilization of standards, testing outsourcing, and the utilization of software components.

Thirdly, the affecting factors were decomposed. Respondents emphasized in process improvement the early involvement of testing, integration of quality into the

development process, and process measurements. In the development of testing automation, respondents listed testing tools, the rationalization of the automation on the cost basis, and reuse. The development of testing know-how contained the development of professional skills and know-how on the operating environment. Issues emphasized in the development of testing strategies consisted of methods and avoiding unnecessary design and testing. The utilization of standards included, for example, the standardization of architectures and software solutions and standardized product platforms. Testing outsourcing contained, for example, off-shore testing in a country with a lower price level and co-operation in testing outsourcing. The utilization of software components consisted of, for example, duplicating the same tested solution to many products and modular components to produce product variants.

4.2.3 Relation to the whole

The results of the survey confirmed the findings of the preliminary study. The viewpoints selected for the thesis – process improvement and knowledge management – appeared also here as the first and the third most important affecting factors.

This exploratory survey revealed factors that affect concurrent cost reduction and quality improvement in testing and related processes. These factors were also decomposed into detailed issues according to the respondents' emphasis. The affecting factors and their decomposition formed the first part of the quantitative analysis. The quantitative analysis continued with a cross-sectional survey of the current situation and improvement needs in software testing and a statistical analysis of the affecting factors. To get a deeper insight, a qualitative analysis of the affecting factors was proposed.

4.3 Publication III: A Survey on Software Testing

4.3.1 Research objectives

The objective of this survey was to clarify the status of testing in the studied organizations and to find improvement needs in the development and testing processes. Special attention was paid to knowledge transfer between development and testing processes. The viewpoints were adopted from the preliminary study.

4.3.2 Results

The questionnaire was divided into four sections: general information on the OU, processes, knowledge transfer between development and testing processes, and the development environment. The survey revealed that the total effort focused on testing (28.9%) was less than expected. The total effort focused on testing (28.9%) was smaller than the 50% that is often mentioned in the literature (Kit 1995). The comparatively low percentage may indicate that that the resources needed for software testing are underestimated.

The assessment of testing indicated that testing tasks have over-run the time reserved and that there have been difficulties in requirements testing. Schedule over-runs may be associated with, for example, underestimated testing resources, knowledge transfer between development and testing processes, insufficient testing tools, and a lack of testing know-how. Also, testing schedules were continuously adjusted.

Respondents emphasized the need to improve knowledge transfer between testing and processes of acquisition, configuration control, and reuse. The results suggested that knowledge transfer should be improved between testing and earlier life cycle processes (e.g. preliminary study, system analysis, and design). Osterweil et al. (1996) and Harrold (2000) emphasize similar issues, i.e. the early involvement of testing. Harter and Slaughter (2000) write that software quality is designed into products rather than incorporated through testing. Respondents evaluated that insufficient testing tools and a lack of testing know-how hinder testing development.

4.3.3 Relation to the whole

The survey clarified the status of testing in the studied organizations, but also raised a special question: why testing tasks have over-run the schedule reserved. For the next phase of the project, a statistical modeling of the schedule over-runs was proposed. The objective of the statistical model was to reveal relationships between schedule over-runs and, for example, knowledge transfer between development and testing processes, insufficient testing tools, a lack of testing know-how and other explanatory variables.

4.4 Publication IV: Factors Affecting Software Testing Time Schedule

4.4.1 Research objectives

The objective of this study was to identify and model constructs that explain the over-runs of testing schedules.

4.4.2 Results

The analysis of correlation revealed that there was a significant negative correlation between success in keeping the testing schedule and knowledge transfer in the earlier life cycle phases (the preliminary phase, the analysis phase, and the design phase). This rather unexpected correlation between the schedule and knowledge transfer could be analogous to what Baskerville et al. (2001) have observed about adjusted processes. Baskerville et al. (2001) state that processes are adjusted because companies regularly trade off among feature slip, time to market, and quality. Also problems in the availability of human resources and the lack of testing know-how correlated negatively. The variable product orientation correlated negatively with keeping the testing schedule, but the correlation was not statistically significant.

The model developed with the stepwise regression algorithm was highly significant (0.000). The independent variables explained 54.8% of the variance of the dependent variable. R Square statistics allowed conclusions on the success in keeping the testing schedule, but there were also variables missing. The identification of these missing variables could be a topic for further research.

The regression model revealed a negative association (-0.409) between the lack of testing know-how and keeping the testing schedule. An increase of one unit in the lack of testing know-how was associated with a decrease of 0.409 in keeping the testing schedule. The negative effect of the variable, the lack of testing know-how, on keeping the testing schedule is apparent. More complicated is the negative association (-0.435) of the construct knowledge transfer between development and testing in the design phase. The correlation analysis also showed significant negative correlations between keeping the testing schedule and knowledge transfer in the preliminary phase, the analysis phase, and the design phase. The results gave a hint that either knowledge transfer is increased in the earlier life cycle phases, expanding and making testing deeper, which leads to schedule over-runs, or the testing schedule is adjusted under project pressure, which means that the resources needed for software testing are underestimated.

4.4.3 Relation to the whole

This exploratory study showed that knowledge transfer is associated with the testing schedule, but in quite an unexpected way. For the next phase, a qualitative analysis of knowledge transfer and other affecting factors was proposed. Qualitative analysis enables the identification of affecting factors and their relationships. Statistical data cannot reveal these kinds of root causes. Instead, the situation of each OU needs to be analyzed separately to explain the variation of reasons for the negative correlation.

4.5 Publication V: Improving Software Testing by Observing Practice

4.5.1 Research objectives

The objective of this qualitative study was to understand the complex practice of software testing, and based on this knowledge, to develop process improvement hypotheses that could be used in concurrent testing cost reduction and software quality improvement. The practice of software testing was observed from the viewpoint of process improvement and described by affecting factors and their relationships.

4.5.2 Results

The process improvement hypotheses included adjusting testing according to the business orientation of the OU, enhanced testability of software components, efficient knowledge transfer between development and testing, early involvement of testing, and use of risk-based testing. The business orientation of an OU proved to be the central affecting factor around which the other factors were integrated. The hypotheses are summarized in Table 5.

Table 5. Testing process improvement hypotheses

1. Testing ought to be adjusted according to the business orientation of the OU. 1A: Product oriented organizations should adopt a formal planned testing process. 1B: Service oriented organizations should adopt a flexible testing process that allows appropriate coordination with their clients.
2. Enhanced testability of software components. 2A: Consider testability as an important factor in the selection of components. 2B: Review the testing process of your suppliers.
3. Efficient communication and interaction between development and testing seems to reduce costs and improve software quality. Product oriented OUs could develop straightforward processes because knowledge transfer is predictable. Service oriented OUs could develop innovative solutions because knowledge transfer varies according to the customer.
4. The early involvement of testing seems to reduce costs and improve quality. The planning of testing is more straightforward in product oriented than in service oriented OUs.
5. The risk-based testing strategy helps in avoiding ad hoc decisions on testing, because the decision of what to test is based on a predefined testing strategy.

According to this study, the business orientation and testing processes were tightly interconnected. The business orientation therefore determines the kinds of improvement efforts that are viable.

Enhanced testability of software components included better documentation and interfaces of the components. Utilization of software components, COTS software, and third-party software seems to be increasing. Efficient knowledge transfer between development and testing seems to improve both testing and development. Knowledge transfer seems to depend on the business orientation of the OU. The early involvement of testing seems to shorten the testing time by enabling better planning of testing, improve the design, and increase the influence of testers. The risk-based testing strategy helped in defining the contents of testing in case of a shortage of resources.

4.5.3 Relation to the whole

The results of this qualitative study described the software testing practice and answered the research question from the process improvement point of view. OUs can apply hypotheses according to their business orientation in concurrent cost reduction and software quality improvement. To answer the research problem from the knowledge management viewpoint, we decided to continue our qualitative analysis from the above-mentioned viewpoint, *Publications VI and VIII.*

4.6 Publication VI: Observing Software Testing Practice from the Viewpoint of Organizations and Knowledge Management

4.6.1 Research objectives

The objective of this qualitative study was the same as in *Publication V*, but the practice of software testing was observed from the viewpoint of organizations and knowledge management. The practice of software testing was described by affecting factors and their relationships.

4.6.2 Results

The hypotheses included that the business orientation of an OU affects the testing organization, knowledge management strategy, and outsourcing of testing. Further, identifying and avoiding barriers and using enablers improve knowledge transfer between development and testing. The hypotheses are summarized in Table 6.

Table 6. Improvement hypotheses for knowledge management and organizations

1. Business orientation affects the testing organization. In product oriented OUs, the organization model ought to support repetitive testing tasks that enable development of a deeper expertise. In service oriented OUs, the organization model ought to support adaptation to the processes of the customers that demand broader expertise.
2. Business orientation affects the knowledge management strategy. OUs ought to adjust the knowledge management strategy according to the business orientation.
3. Business orientation and the knowledge management strategy affect outsourcing. Codification of knowledge enables testing outsourcing.
4. Identifying and avoiding barriers and using enablers improve knowledge transfer.

The results suggested that the business orientation affects the testing organization and knowledge management strategy, but we cannot suggest that organizations should select a fixed knowledge management strategy according to their business orientation. Hansen et al. (1999) view that the company should focus on only one of the knowledge management strategies. Independent testing agencies made an exception. Even though the independent testing agencies were service oriented, their knowledge management strategy was more inclined towards codification. This does not seem to directly fit the description that companies providing services should focus only on the personalization strategy. According to our study, the business orientation of an OU is not stable because many product oriented OUs strive to develop services, and service oriented OUs strive to develop products.

Product oriented OUs used more outsourced testing resources than service oriented OUs. Because product oriented OUs codify more knowledge, the development and testing processes are more repetitive and outsourcing is easier. OUs strived for efficient knowledge transfer by identifying and avoiding barriers such as the distance between design and testing locations, the lack of absorptive capacity of the persons, the lack of planned and formal meetings, and customer control of the knowledge transfer and using enablers such as contact persons to improve knowledge transfer.

4.6.3 Relation to the whole

The results of this qualitative study described the software testing practice and answered the research problem from the organization and knowledge management point of view. The derived hypotheses can be applied in concurrent cost reduction and software quality improvement.

4.7 Publication VII: Triangulating Testing Schedule Over-runs from Knowledge Transfer Viewpoint

4.7.1 Research objectives

The objective of this study was to explain how software testing schedule over-runs are associated with the knowledge transfer between development and testing processes. The quantitative study (*Publication III*: A Survey on Software Testing) raised a special question: which factors affect software testing schedule over-runs? The question was statistically analyzed in *Publication IV:* Factors Affecting Software Testing Time Schedule. The statistical analysis showed that knowledge transfer between development and testing is associated with the testing schedule, but in a quite unexpected way: increased knowledge transfer between development and testing was associated with testing schedule over-runs.

4.7.2 Results

OUs strived to reach optimal conditions where testing schedules are not over-run, the scope of testing is comprehensive, and the knowledge transfer is efficient, but the trade-off between the schedule and the scope of testing seemed to occur more frequently in practice, leading to two typical scenarios. These two typical scenarios that explain the negative association between knowledge transfer and schedule over-runs are listed in Table 7.

Table 7. Two typical scenarios explaining the negative association between knowledge transfer and testing schedules

Business orientation	Scope of testing	Schedule of testing	Knowledge transfer between development and testing	Explanation
Product-oriented	Fixed	Adjusts	May be high and efficient	Adjusting schedules and efficient KT → negative association
Service-oriented	Adjusts	Fixed	Efficiency may suffer from several barriers	Fixed schedules and inefficient KT → negative association

According to the scenarios, a product-oriented OU prefers the scope of testing instead of the schedule, its knowledge transfer has fewer barriers than in service-oriented OUs, and therefore, knowledge may be transferred efficiently. This leads to adjusting schedules and efficient knowledge transfer and thus to a negative association.

In contrast, a service-oriented OU prefers the schedule instead of the scope, it has more barriers than product-oriented OUs, and this results in fixed schedules and inefficient knowledge transfer. Therefore, again, the association is negative.

OUs can avoid barriers and use enablers to improve knowledge transfer between development and testing processes. Szulanski (1996) lists as barriers to internal knowledge transfer factors such as the recipient's lack of absorptive capacity, causal ambiguity, and an arduous (laborious and distant) relationship between the source and the recipient. We observed several factors in case OUs that weakened the efficiency of knowledge transfer. These factors included distance, knowledge transfer control by customers, and difficulties of the persons involved to understand each other. The lack of absorptive capacity was two-fold in our observations. We observed that in some cases testers were unable to understand the technical vocabulary of the developers, and the use of a foreign language as the documentation language also caused difficulties.

It was widely understood in the case OUs that the codification of knowledge should increase efficiency. According to Nonaka (1994), if the knowledge has tacit components, the knowledge transfer may require numerous individual exchanges. On the other hand, the codification strategy may not be suitable for all tasks, especially if the codified knowledge cannot be reused (Conradi & Dybå 2001; Hansen et al. 1999).

4.7.3 Relation to the whole

The two typical scenarios explained the root cause for the negative association between knowledge transfer and testing schedule over-runs and answered the emergent special question. This result confirms the findings of Baskerville et al. (2001) that processes adjust under project pressure, but the central result was that the business orientation of an OU affects the trade-off between the testing schedule and the scope of testing, leading to two typical scenarios.

4.8 Publication VIII: Outsourcing and Knowledge Management in Software Testing

4.8.1 Research objectives

The objective of this qualitative study was to explore outsourcing in software testing and shape hypotheses that describe the relationship between outsourcing and knowledge management.

4.8.2 Results

The hypotheses included that the business orientation and the knowledge management strategy of an OU affect testing outsourcing, the knowledge management strategy of an OU depends on the OU's business orientation, the efficiency of outsourcing depends on the domain knowledge of an independent testing agency, and outsourcing verification tasks is difficult. The hypotheses are summarized in Table 8.

Table 8. Hypotheses describing the relationship between outsourcing and knowledge management

1. The business orientation of an OU and the knowledge management strategy affect testing outsourcing. The product oriented OUs have more possibilities of using outsourced testing than the OUs developing customized systems. Codification and explicit knowledge enable outsourcing.
2. OUs ought to adjust their knowledge management strategy according to the business orientation.
3. Outsourcing seems to be more effective when independent testing agencies have enough domain knowledge.
4. Outsourcing verification tasks is more difficult than outsourcing validation tasks.

4.8.3 Relation to the whole

The results of this study described the relationship between testing outsourcing and knowledge management. Outsourcing and knowledge management are very much interconnected. Osterloh and Frey (2000) note that outsourcing necessitates making knowledge explicit. Also, when knowledge is codified it becomes transferable, independently of people in whom tacit knowledge is embedded (Foray 2004). When making decisions about testing outsourcing, also issues related to knowledge management should be taken into account. Cowan et al. (2000) state that codification will provide high benefits in 1) stable systems characterized by specific requirements of knowledge transfer and communication, such as delocalization and externalization, i.e. outsourcing., 2) stable systems where advances and novelties mainly proceed from recombination, reuse and cumulativeness, i.e. standardized systems, and 3) systems that require extensive memory and retrieval capabilities (e.g. firms and organizations that have long product or process development cycles or high rates of personnel turnover). Beath and Walker (1998) state that outsourcing is most effective when the supplier's existing knowledge is adequate to the tasks implied by the project.

Verification tasks require more knowledge transfer between the customer and the independent testing agency because artifacts needed in verification include, for example, source code and design documents. This may pose a potential risk to the customer. Teece (1986) notes that while codified knowledge is easier to transmit and receive, it is more exposed to industrial espionage and the like.

4.9 About the joint publications

For *Publication I,* the author collected and analyzed the data, wrote the major part of the publication, and presented it.

For *Publications II-IV,* the author designed and implemented the survey instrument, collected and analyzed the data, wrote the major part of the publications, and presented them.

For *Publications V and VI,* the author designed the research approach, participated in collecting and analyzing the data, wrote the major part of the publications, and presented them.

For *Publication VII,* the author designed the research approach, participated in collecting and analyzing the data, and wrote the major part of the publication.

For *Publication VIII, t*he author designed the research approach, participated in collecting and analyzing the data, and in writing the publication.

5 Implications of the results

In this chapter, the implications of the research results are extracted from the publications and presented as a summary. The research problem and the original objective of the thesis remained unchanged during the studies, but the research problem was divided into sub-questions and answered in respective publications. The objective of this thesis was not to produce a method, tool or other artifact of software testing but to identify and understand the factors that affect concurrent testing cost reduction and software quality improvement, and through that understanding, generate improvement hypotheses from the selected process improvement and knowledge management viewpoints. This was done by analyzing affecting factors and their relationships with quantitative and qualitative methods.

5.1 Implications for practice

The viewpoints of the thesis – process improvement and knowledge management – were selected in the preliminary phase of the thesis *(Publication I).* Experts in the preliminary study evaluated process improvement with knowledge transfer as the most important research issue, the second in ranking was testing automation and testing tools, and standardization was ranked third. Based on these results, we concluded that the selected viewpoints represent an important area of software testing and specify the scope of the thesis. The selected viewpoints of this thesis cover many important issues of software testing practice, but also important issues are delimited outside the scope of this thesis. Other viewpoints, such as testing automation and testing tools, standardization, etc., will be discussed in future research topics.

The factors that affect testing cost reduction and software quality improvement were identified and decomposed in *Publication II.* Firstly, the respondents evaluated personnel costs as the highest and testing automation costs as the second highest cost item. Other cost items were marginal compared to these two. According to the results, finding the right balance between labor and automation costs can improve the cost structure of software testing.

Secondly, the respondents evaluated how to concurrently reduce costs and improve software quality in development and testing. According to our study, affecting factors included development and testing process improvement, the development of testing automation, the development of testing know-how, the development of testing strategies, the utilization of software components, the utilization of standards, and the outsourcing of testing. Interviewees in the quantitative study, *Publication II*, evaluated "process improvement" as the most important factor and "testing know-how" as the third most important factor in concurrent cost reduction and software quality improvement. These findings confirmed the selection of the scope of the thesis made during preliminary study. Three viewpoints seem to rise above others in achieving cost reductions and software quality improvement: process improvement, knowledge management, and testing automation.

In *Publication III*, the current situation and improvement needs in software testing were analyzed. The survey described the cross-sectional situation in software testing among 30 OUs and suggested improved knowledge transfer between testing and earlier life cycle processes of software development. In evaluating knowledge transfer between testing and the process groups (ISO/IEC 2004), the respondents emphasized the need to improve knowledge transfer especially between testing and processes of acquisition, configuration control, and reuse[1]. The results offer basic statistics of 30 OUs, processes, communication and interaction, and the development environment.

Other findings in *Publication III* included that testing tasks have over-run their schedule. Schedule over-runs were evaluated to be associated with, for example, underestimated testing resources, knowledge transfer between development and testing, insufficient testing tools, and a lack of testing know-how. Also, testing schedules were continuously adjusted. These findings raised a special question, how the knowledge transfer between development and testing and testing schedule over-runs are associated. This special question that emerged was discussed in *Publications IV* and *VII*. The correlation and regression analyses in *Publication IV* revealed that increased knowledge transfer in the design phase between development and testing

[1] In the current version of the 15504-5 standard (ISO/IEC 2006) the configuration control and quality assurance process groups have been merged as the support process group.

was associated with testing schedule over-runs. The result gave a hint that either knowledge transfer in the earlier life cycle phases is increased, expanding and making testing deeper, which leads to time schedule over-runs, or the testing schedule is adjusted under project pressure, which means that the resources needed for software testing are underestimated. Finally, in *Publication VII* the negative association was explained with two typical scenarios that explain the root causes for this negative association. The central result was that the business orientation of an OU affects the trade-off between the testing schedule and the scope of testing, leading to two typical scenarios: Product oriented OUs emphasize the scope of testing, and their knowledge transfer may be more efficient because they have fewer barriers to knowledge transfer than service-oriented OUs. In contrast, service-oriented OUs prefer the schedule instead of the scope, and their knowledge transfer may be inefficient because they have more barriers than product-oriented OUs. The results explain how the business orientation of an OU affects the trade-off between the testing schedule and the scope of testing and barriers to knowledge transfer. The results can be used in balancing the trade-off between the testing schedule and the scope of testing according to the business orientation of an OU and in avoiding barriers to knowledge transfer. This can be further applied in improving knowledge transfer and in avoiding schedule over-runs.

In *Publication V* the practice of software testing was analyzed from the process improvement viewpoint. The analysis yielded testing process improvement hypotheses. The central result was that testing processes ought to be adjusted according to the business orientation of the OU. Product oriented organizations should adopt a formal planned testing process. Service oriented organizations should adopt a flexible testing process that allows appropriate coordination with their clients. The result can be applied in developing testing processes. Further, enhanced testability of software components, efficient communication and interaction between development and testing, and the early involvement of testing seem to reduce testing costs and improve software quality. Also, a risk-based testing strategy helps in avoiding ad hoc decisions on testing.

In *Publication VI,* the practice of software testing was analyzed from the knowledge management viewpoint. The analysis yielded improvement hypotheses. The central result was that the business orientation affects the testing organization and knowledge management strategy. The results suggest that the testing organization and knowledge management strategy ought to be developed according to the business orientation of the OU. An important finding was also that the business orientation of an OU is not stable because product oriented OUs develop services and service oriented OUs develop products. Further, the business orientation and the knowledge management strategy affect testing outsourcing. OUs also strive for efficient

knowledge transfer. Identifying and avoiding barriers and using enablers improve knowledge transfer.

In *Publication VIII,* the association between testing outsourcing and knowledge management was analyzed. The analysis yielded improvement hypotheses. According to this study, the business orientation of an OU affected testing outsourcing. The product oriented OUs seem to have more possibilities of using outsourced testing than the OUs developing customized systems. In OUs developing customized systems, early involvement with software development seems to be more important. Outsourcing seems to be more effective when independent testing agencies have enough domain knowledge. Independent testing agencies do not necessarily have an adequate amount of domain knowledge, which is tacit and context-dependent by nature, therefore making it difficult to transfer and codify.

Outsourcing verification tasks seems to be more difficult than outsourcing validation tasks. Independent testing agencies usually focus on system testing, and are not concerned with the software's internal processing (Dustin et al. 1999). System testing consists mainly of validation tasks. Verification is also more closely involved in many stages of software development, and requires close co-operation with several instances inside the organization. It is therefore harder to define verification as an independent and separate function, which can then be given as a task to an independent testing agency.

5.2 Implications for further research

The selected research methods describe my opinion on which methodology is the best fit for approaching the subject of the research. This mixed method research showed that the combination of quantitative and qualitative methods can reveal root causes that may be difficult to uncover when the methods are used in isolation.

The scope of the thesis was delimited by selecting the viewpoints of the thesis. The viewpoints delimited outside the scope of this thesis, such as testing automation and testing tools and standardization, ought to be analyzed in further research because analysis from these viewpoints can reveal important affecting factors, their relationships, and improvement hypotheses.

We selected both in the quantitative and in the qualitative analysis constructs of a high abstraction level. Now that the analysis of the high abstraction level constructs is available, the further analysis could concentrate on the lower abstraction level constructs. The decomposition of affecting factors (*Publication II*) could serve as the starting point for the analysis. A detailed analysis could be carried out from, for

example, the testing automation and standardization viewpoints as a new analysis and from the viewpoints of this thesis as a continuation of this work.

According to the results of this thesis, the effect of the business orientation of an OU was central in software testing. Runeson et al. (2006) studied defect detection methods. Their analysis of existing empirical studies showed no clear-cut answer to the question of which defect detection method to choose. This and the results of this thesis suggest that there exist higher level factors, such as business orientation, that affect, for example, processes, automation, and knowledge management, and further, defect detection.

6 Conclusions

6.1 Derived conclusions

This thesis describes a research project consisting of a preliminary study, quantitative studies, and qualitative studies. Each of the phases offered results and empirical observations on the practice of software testing.

This thesis makes four contributions. Firstly, in the preliminary phase the scope of the thesis was specified as the viewpoints process improvement and knowledge management. Other viewpoints of software testing are proposed as future research topics. Secondly, it shows by quantitative studies factors that affect software testing practice and that knowledge transfer is associated with testing schedules (emergent special question). Thirdly, it explains with a mixed methods analysis the negative association between testing schedules and knowledge transfer. Fourthly, it explores by qualitative analyses factors that affect testing costs and software quality from the selected viewpoints and produces improvement hypotheses.

According to this thesis, the business orientation of an OU is the central affecting factor in software testing practice. The results and observations created scenarios and improvement hypotheses for both practitioners and researchers:

1. Two typical scenarios that explain the negative association between the testing schedule and knowledge transfer (emergent special question):

- A negative association between the testing schedule and knowledge transfer existed if the OU selected a comprehensive scope of testing instead of keeping

the testing schedule and if the knowledge transfer between development and testing processes was efficient.

- A negative association between the testing schedule and knowledge transfer existed if an OU selected a reduced scope of testing to keep the testing schedule and there were barriers to knowledge transfer, most probably because of service-orientation and its consequences, between development and testing processes.

2. Improvement hypotheses from the process improvement viewpoint:

- Testing ought to be adjusted according to the business orientation of the OU. Product oriented organizations should adopt a formal planned testing process. Service oriented organizations should adopt a flexible testing process that allows appropriate coordination with their clients.

- Enhanced testability of software components. Consider testability as an important factor in the selection of components. Review the testing process of your suppliers.

- Efficient communication and interaction between development and testing seems to reduce costs and improve quality. Product oriented OUs could develop straightforward processes because knowledge transfer is predictable. Service oriented OUs could develop innovative solutions because knowledge transfer varies according to the customer.

- The early involvement of testing seems to reduce costs and improve quality. The planning of testing is more straightforward in product oriented than in service oriented OUs.

- The risk-based testing strategy helps in avoiding ad hoc decisions on testing, because the decision on what to test is based on a predefined testing strategy.

3. Improvement hypotheses from the knowledge management viewpoint:

- Business orientation affects the testing organization. In product oriented OUs, the organization model ought to support repetitive testing tasks that enable development of a deeper expertise. In service oriented OUs, the organization model ought to support adaptation to the processes of the customers that demand broader expertise.

- Business orientation affects the knowledge management strategy. OUs ought to adjust their knowledge management strategy according to the business orientation.

- Business orientation and the knowledge management strategy affect outsourcing. Codification of knowledge enables testing outsourcing.
- Identifying and avoiding barriers and using enablers improve knowledge transfer.

4. Associations between testing outsourcing and knowledge management:

    - Outsourcing seems to be more effective when independent testing agencies have enough domain knowledge.
    - Outsourcing verification tasks is more difficult than outsourcing validation tasks.

To summarize, the business orientation of an OU has a strong role in the practice of software testing. An analysis of affecting factors from different viewpoints can offer more improvement hypotheses. Empirical research can provide more knowledge and evidence necessary for the development of more advanced tools and methods for software testing.

6.2 Limitations of this thesis

All projects have their shortcomings and limitations, and several can be identified in this one, as well. The research problem delimited this thesis, because we were interested only in factors that affect concurrent testing cost reduction and software quality improvement. Based on the formulation of the research problem, our target was to describe the practice of software testing. The scope of the thesis was delimited in the preliminary phase by adopting process improvement and knowledge management as the viewpoints of the thesis and by abandoning other viewpoints. To get a more comprehensive understanding of software testing, an analysis from the abandoned viewpoints is recommendable. When considering the validity of the thesis, we must look separately at the quantitative and qualitative parts, but it is also possible to point out the benefits of methodological triangulation and how it increases the validity and trustworthiness of the entire thesis.

A possible limitation of the preliminary study is that the results can be applied only to similar environments. The informants of this study represented organizations that produce technically highly advanced products and applications in the telecommunication and automation domains. The criticality of their products is above average and the products are used in real time environments. It is possible that the rankings in other kinds of applications may have a different order and selection of issues.

A limitation of the quantitative part of the thesis is the tight specification of the population and the sample. The results can only be directly generalized when discussing comparable OUs[2]. In spite of this limitation, we believe that the results have a wider significance because the selected OUs were from companies that were at an advanced technological level and their produced applications demanded high technological sophistication. In addition, the respondents of this study had extensive experience in the field (on an average more than 13 years), and therefore, we think that the reliability of their answers is high.

The limitation of the qualitative study was the number of case OUs. It is obvious that increasing the number of cases in qualitative analyses could reveal more details, and it is possible that some polar point cases could even formulate a new explanatory factor. However, our target was not to create a comprehensive list of the factors that affect the practice of software testing, but to cover the most important factors from the point of view of our case OUs.

The purpose of the qualitative part of the thesis was to understand the testing practice in five case OUs. This kind of effort requires interpretation and exploration. Robson (2002) lists three threats to validity in this kind of research: reactivity (the interference of the researcher's presence), researcher bias, and respondent bias and seven strategies that reduce these threats. We have used Robson's strategies in the following way.

The research has lasted three years and consisted of several phases and data collection rounds. All four types of triangulation presented by Denzin (1978) have been used: data, observer, methodological, and theory triangulation. The research has consisted of regular meetings with research participants from several research institutions where the preliminary results have been presented and discussed. The interpretation of the data has been confirmed by presenting the results to company participants in the research project. An example of a negative case in our study was case D, which was a purely product oriented OU. This, however, did not disconfirm our theoretical understanding. Instead, it complemented it and provided more ingredients. All interviews have been recorded and transcribed. The notes and memos of the study have been preserved, and preliminary data coding and analysis results are available through the analysis tool used, ATLAS.ti.

The strongest method for ensuring the overall validity of the thesis has been the triangulation. To reduce the bias caused by researchers, we used observer triangulation. The bias caused by the method was minimized using methodological triangulation, and the bias caused by data using data triangulation. In addition, the

[2] In fact, this is a limitation of any survey, regardless of the sample size (see, for example, Lee & Baskerville (2003)).

Publications I-VIII of this thesis have approached the phenomenon from different viewpoints, and therefore, they enforce theory triangulation.

Methodological triangulation means that multiple research methods are used and their results are compared to each other. In this thesis, methodological triangulation consisted of the combination of statistical methods and qualitative analysis with the grounded theory method. In addition, the preliminary study was completed using the Delphi method.

In observer triangulation, researchers with different backgrounds and experiences study the same research topic and participate in the data collection. In this thesis, the quantitative analysis was carried out by one researcher and the qualitative analysis by four researchers, whose interpretations completed each other, and therefore, made the study more trustworthy.

Data triangulation means the use of multiple data collection methods that provide stronger substantiation of constructs and hypotheses (Eisenhardt 1989). The primary data collection method in this thesis was interviews. The interviews in the first round, based on the survey method, were performed by one researcher, and the interviews in the following rounds by two researchers. In addition to the interview data, we used field notes.

6.3 Future research topics

Many research implications were mentioned in section 5.2, and the results of this thesis might be extended and deepened into many directions. In the following, three of them are described.

First, the research approach used in exploring the association between knowledge transfer and testing schedule over-runs could be continued at a more detailed level and also used in explaining other complicated relationships in testing, such as testing schedules versus testing automation. Secondly, software testing and also other areas of software engineering could be explored repeating iteratively respective quantitative (as described in *Publications II-IV*) and qualitative (as described in *Publications V-VIII*) phases. Concurrently the abstraction level of the constructs used could be changed into a more detailed form. Thirdly, the results of this thesis have crated a basis for a testing assessment model. Analyzing software testing practice from new viewpoints, such as testing automation, standardization, etc., produces new affecting factors and further hypotheses. At the same time, the abstraction level can be changed into a more detailed form using the decomposition of affecting factors (*Publication II)*. The assessment of software testing with the model gives an OU important information for

developing testing processes, knowledge management, and testing automation while simultaneously optimizing the software testing costs and the software quality.

The assessment could contain four phases: First, an OU is selected. Secondly, the OU is positioned according to its business orientation and the criticality of its end products. Thirdly, the OU is assessed according to its business orientation from the viewpoints of process improvement, knowledge management, and testing automation at a detailed level. Finally, as the result of the assessment, improvement proposals are generated.

References

Abrahamsson, P. (2001), 'Commitment development in software process improvement: critical misconceptions', paper presented to the International Conference on Software Engineering (ICSE 2001). Proceedings of the 23rd International Conference on Software Engineering, Toronto, Canada, pp. 71-80.

Abrahamsson, P., Warsta, J., Siponen, M. and Ronkainen, J. (2003), 'New Directions on Agile Methods: A Comparative Analysis', paper presented to 25th International Conference on Software Engineering.

ACM, AIS and IEEE-CS (2004), *Computing Curricula 2004*.

Ambler, S. (2002), *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process*, John Wiley & Sons, New York.

Amland, S. (2000), 'Risk Based Testing and Metrics: Risk analysis fundamentals and metrics for software testing including a financial application case study', *The Journal of Systems and Software*, vol. 53, pp. 287-295.

Argote, L. and Ingram, P. (2000), 'Knowledge Transfer: A Basis for Competitive Advantage in Firms', *Organizational Behavior and Human Decision Processes*, vol. 82, no. 1, pp. 150-169.

ATLAS.ti - The Knowledge Workbench (2005), Scientific Software Development.

Aurum, A., Jeffery, R. and Wohlin, C. (1998), *Managing Software Engineering Knowledge*, Springer Verlag, New York.

Aurum, A., Petersson, H. and Wohlin, C. (2002), 'State-of-the-Art: Software Inspections after 25 Years', *Software Testing, Verification, and Reliability*, vol. 12, no. 3, pp. 133-154.

Baber, R. L. (1982), *Software reflected*, North-Holland publishing company.

Bach, J. (2003), *Explotary Testing Explained*, viewed 05.08.2006 2003, <http://www.satisfice.com/articles/et-article.pdf>.

Baruch, Y. (1999), 'Response Rate in Academic Studies - A Comparative Analysis', *Human Relations*, vol. 52, no. 4, pp. 421-438.

Baskerville, R., Levine, L., Pries-Heje, J., Ramesh, B. and Slaughter, S. (2001), 'How Internet Software Companies Negotiate Quality', *Computer*, vol. 34, no. 5, pp. 51-57.

Beath, C. M. and Walker, G. (1998), 'Outsourcing of Application Software: A Knowledge Management Perspective', paper presented to Thirty-First Annual Hawaii International Conference on System Sciences.

Beck, K. (2000), *Extreme Programming Explained: Embrace Change*.

Becker, M. C. and Knudsen, M. P. (2003), 'Barriers and managerial challenges to knowledge transfer processes', paper presented to DRUID Summer Conference on Creating, Sharing and Transferring Knowledge, Copenhagen.

Beizer, B. (1990), *Software testing techniques*, Van Nostrand Reinhold, New York.

Berner, S., Weber, R. and Keller, R. K. (2005), 'Observations and lessons learned from automated testing', paper presented to The 27th International Conference on Software Engineering, St. Louis, MO, USA, pp. 571-579.

Binder, R. V. (2001), *Testing Object-Oriented Systems*, Addison Wesley, Boston.

Boehm, B. (1991), 'Software Risk Management: Principles and Practices', *IEEE Software*, vol. 8, no. 1, pp. 32-41.

Boehm, B. (2006), 'A view of 20th and 21st century software engineering', paper presented to International Conference on Software Engineering, Shanghai, China, pp. 12-29.

Boehm, B. and Turner, R. (2003), 'Using Risk to Balance Agile and Plan-Driven Methods', *Computer*, vol. June, pp. 57-66.

Brown, A. W. (2000), *Large-Scale, Component-Based Development*, Prentice Hall.

Burnstein, I., Suwanassart, T. and Carlson, R. (1996), 'Developing a testing maturity model for software test process evaluation and improvement', paper presented to International test conference, pp. 581-589.

Cai, K.-Y., Chen, T. Y., Li, Y.-C., Ning, W.-Y. and Yu, Y. T. (2005), 'Adaptive Testing of Software Components', paper presented to the Symposium on Applied Computing, Proceedings of the 2005 ACM symposium on Applied computing, Santa Fe, New Mexico, pp. 1463 - 1469.

Carter, C. R. and Dresner, M. (2001), 'Purchaser's role in environmental management: Cross-functional development of grounded theory', *Journal of Supply Chain Management*, vol. 37, no. 3, pp. 12-28.

Cockburn, A. (2000), *Writing Effective Use Cases, The Crystal Collection for Software Professionals*, Addison Wesley.

Cohen, C. F., Birkin, S. J., Garfield, M. J. and Webb, H. W. (2004), 'Managing Conflict in Software Testing', *Communications of the ACM*, vol. 47, no. 1.

Conradi, R. and Dybå, T. (2001), 'An empirical study on the utility of formal routines to transfer knowledge and experience', paper presented to the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering, Vienna, Austria, pp. 268-276.

Cowan, R., David, P. A. and Foray, D. (2000), 'The Explicit Economics of Knowledge Codification and Tacitness', *Industrial and Corporate Change*, vol. 9, no. 2, pp. 211-253.

Cronbach, L. J. (1951), 'Coefficient Alpha and the Internal Structure of Tests', *Psychometrika*, vol. 16, no. 3, pp. 279-334.

Cusumano, M. A. and Yoffie, D. B. (1999), 'Software development on Internet time', *IEEE Computer*, vol. 32, pp. 60-69.

Dai, Z. R., Grabowski, J., Neukirchen, H. and Pals, H. (2004), 'From Design to Test with UML', paper presented to 16th IFIP International Conference, TestCom 2004, Testing of Communicating Systems, Oxford, UK, pp. 33-49.

Dalkey, N. C. (1969), *The Delphi method: An experimental study of group opinion*, RAND Corporation, Santa Monica, CA.

Dalkey, N. C. and Helmer, O. (1963), 'An experimental application of the Delphi method to the use of experts', *Management Science*, vol. 9, pp. 458-467.

Denzin, N. K. (1978), *The research act: A theoretical introduction to sociological methods*, McGraw-Hill.

Dibbern, J., Goles, T., Hirschheim, R. and Jayatilaka, B. (2004), 'Information systems outsourcing: a survey and analysis of the literature', *ACM SIGMIS Database*, vol. 35, no. 4.

Dustin, E., Rashka, J. and Paul, J. (1999), *Automated software testing: introduction, management, and performance*, Addison-Wesley, Boston.

Dybå, T. (2000), 'An Instrument for Measuring the Key Factors of Success in Software Process Improvement', *Empirical Software Engineering*, vol. 5, pp. 357-390.

Edwards, J. S. (2003), 'Managing Software Engineers and Their Knowledge', in A. Aurum, R. Jeffery, C. Wohlin and M. Handzic (eds), *Managing Software Engineering Knowledge*, Springer, New York, p. 375.

Eisenhardt, K. M. (1989), 'Building Theories from Case Study Research', *Academy of Management Review*, vol. 14, no. 4, pp. 532-550.

Emam, K. E., Drouin, J.-N. and Melo, W. (1998), *SPICE The Theory and Practice of Software Process Improvement and Capability Determination*, IEEE Computer Society Press, Los Alamitos, CA.

Fink, A. (2003), *The Survey Handbook*, SAGE Publications, Inc.

Fink, A. and Kosecoff, J. (1985), *How to conduct surveys A Step-by-Step Guide*, SAGE Publications, Inc., Newbury Park, CA.
Foray, D. (2004), *The Economics of Knowledge*, The MIT Press, Cambridge, MA.
Glaser, B. and Strauss, A. L. (1967), *The Discovery of Grounded Theory: Strategies for Qualitative Research*, Aldine, Chicago.
Graham, D. (2002), 'Requirements and testing: Seven missing-link myths', *IEEE Software*, vol. 19, no. 5, pp. 15-17.
Hansen, M. T., Nohria, N. and Tierney, T. (1999), 'What's Your Strategy for Managing Knowledge?' *Harvard Business Review*, vol. 77, no. 2, pp. 106-116.
Harrold, M. J. (2000), 'Testing: A Roadmap', paper presented to the International Conference on Software Engineering, Limerick, Ireland, pp. 61-72.
Harter, D. E. and Slaughter, S. A. (2000), 'Process maturity and software quality: a field study', paper presented to the International Conference on Information Systems, Brisbane, Australia, pp. 407-411.
Heiser, J. E. (1997), 'An Overview of Software Testing', paper presented to Autotestcon '97. 1997 IEEE Autotestcon Proceedings, Anaheim, CA, USA, pp. 204-211.
Hetzel, B. and Gelperin, D. (1988), 'The growth of software testing', *Communications of the ACM*, vol. 31, no. 6, pp. 687-695.
Highsmith, J. A. (2000), *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*, Dorset House Publishing, New York.
Hirschheim, R. A. (1985), 'Information systems epistemology: an historical perspective', in R. H. E Mumford, G Fitzgerald, T Wood-Harper (ed.), *Research Methods in Information Systems*, North-Holland, Amsterdam.
Hirsjärvi, S., Remes, P. and Sajavaara, P. (1997), *Tutki ja kirjoita*, Dark Oy, Vantaa.
Hunt, A. and Thomas, D. (2000), *The Pragmatic Programmer*, Addison Wesley.
IEEE/ANSI (1983), *IEEE Standard for Software Test Documentation*, 829-1983.
IEEE/ANSI (1986), *IEEE/ANSI Standard for Software Verification and Validation Plans, Reaff. 1992*, 1012-1986.
IEEE/ANSI (1990), *IEEE Standard Glossary of Software Engineering Terminology*, 610.12-1990.
ISO/IEC (2001), *ISO/IEC 12207:1995/FDAM Information technology-Software life cycle processes*.
ISO/IEC (2002), *ISO/IEC 15504-1, Information Technology - Process Assessment - Part 1: Concepts and Vocabulary*.
ISO/IEC (2004), *ISO/IEC CD 15504-5 Information Technology-Process Assessment-Part 5: An exemplar Process Assessment Model*.
ISO/IEC (2006), *ISO/IEC 15504-5 Information technology - Process Assessment - Part 5: An exemplar Process Assessment Model*, ISO/IEC.
Jacobson, I., Booch, G. and Rumbaugh, J. (1999), *The Unified Software Development Process*, Addison Wesley, Reading.

John, M., Maurer, F. and Tessem, B. (2005), 'Human and Social Factors of Software Engineering - Workshop Summary', *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, pp. 1-6.

Jones, E. L. (2000), 'Software testing in the computer science curriculum -- a holistic approach', paper presented to Proceedings of the Australasian conference on Computing education, Melbourne, Australia, pp. 153-157.

Järvinen, P. (1999), *On Research Methods*, Tampereen Yliopistopaino Oy, Tampere.

Kaner, C., Falk, J. and Nguyen, H. Q. (1999), *Testing Computer Software*, 2 edn, Wiley, New York.

Karlström, D., Runeson, P. and Wohlin, C. (2002), 'Aggregating viewpoints for strategic software process improvement - a method and a case study', *Software IEE Proceedings*, vol. 149, no. 5, pp. 143-152.

Katara, M. (2005), 'Testauksen työvälineet nyt ja tulevaisuudessa - akateeminen näkemys', *Systeemityö*, vol. 1, pp. 15-17.

Keil, M., Cule, P. E., Lyytinen, K. and Schmidt, R. C. (1998), 'A Framework for Identifying Software Project Risks', *Communications of the ACM*, vol. 41, no. 11.

Kit, E. (1995), *Software Testing in the Real World: Improving the Process*, Addison-Wesley, Reading, MA.

Koomen, T. and Pol, M. (1999), *Test Process Improvement: a Practical Step-by-Step Guide to Structured Testing*, Addison-Wesley.

Lee, A. S. and Baskerville, R. L. (2003), 'Generalizing Generalizability in Information Systems Research', *Information Systems Research*, vol. 14, no. 3, pp. 221-243.

Litwin, M. S. (1995), *How to Measure Survey Reliability and Validity*, Sage Publications, Thousand Oaks, CA.

Mantere, T. (2003), *Automatic Software Testing by Genetic Algorithms. Dissertation.*, vol. 112, Universitas Wasaensis, Vaasa.

Miles, M. B. and Huberman, A. M. (1994), *Qualitative Data Analysis*, SAGE Publications, Thousand Oaks, CA.

Morris, P., Masera, M. and Wilikens, M. (1998), 'Requirements Engineering and Industrial Uptake', *Requirements Engineering*, vol. 3, pp. 79-83.

Myers, G. J. (1976), *The Art of Software Testing*, John Wiley & Sons, NY.

Naur, P. and Randell, B. (1969), 'Software Engineering: Report on a Conference Sponsored by the NATO Science Committee', Garmisch, Germany, 7-11 Oct.

Nerur, S., Mahapatra, R. and Mangalaraj, G. (2005), 'Challenges of Migrating to Agile Methodologies', *Communications of the ACM*, vol. 48, no. 5, pp. 72-78.

Ng, S. P., Murnane, T., Reed, K., Grant, D. and Chen, T. Y. (2004), 'A preliminary survey on software testing practices in Australia', paper presented to the 2004 Australian Software Engineering Conference. 2004: 116-25, Melbourne.

Nonaka, I. (1994), 'A Dynamic Theory of Organizational Knowledge Creation', *Organization Science*, vol. 5, no. 1, pp. 14-37.

Nonaka, I. and Takeuchi, H. (1995), *The Knowledge-Creating Company*, Oxford University Press, New York.

Northrop, L. (2006), 'Software Product Lines: Reuse That Makes Business Sense', paper presented to ASWEC 2006.
Nunnally, J. C. (1978), *Psychometric theory*, McGraw-Hill, New York.
Osterloh, M. and Frey, B. S. (2000), 'Motivation, Knowledge Transfer, and Organizational Forms', *Organization Science*, vol. 11, no. 5, pp. 538-550.
Osterweil, L., Clarke, L. A., DeMillo, R. A., Feldman, S. I., McKeeman, B., Salasin, E. F. M., Jackson, D., Wallace, D. and Harrold, M. J. (1996), 'Strategic Directions in Software Quality', *ACM Computing Surveys*, vol. 28, no. 4, pp. 738-750.
Osterweil, L. J. (1987), 'Software Processes are Software Too', paper presented to the 9th International Conference on Software Engineering, Monterey, CA, pp. 2-13.
Osterweil, L. J. (1997), 'Software processes are software too, revisited: an invited talk on the most influential paper of ICSE 9', paper presented to the International Conference on Software Engineering, Proceedings of the 19th International Conference on Software Engineering, Boston, pp. 540-548.
Osterweil, L. J. (2003), 'Understanding Process and the Quest for Deeper Questions in Software Engineering Research', paper presented to Foundations of Software Engineering. Proceedings of the 9th European Software Engineering Conference held jointly with the 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Helsinki, Finland, pp. 6-14.
Palmer, S. R. and Felsing, J. M. (2002), *A Practical Guide to Feature-Driven Development*.
Paré, G. and Elam, J. J. (1997), 'Using Case Study Research to Build Theories of IT Implementation', paper presented to the IFIP TC8 WG International Conference on Information Systems and Qualitative Research, Philadelphia, USA.
Pather, S. and Remenyi, D. (2004), 'Some of the philosophical issues underpinning research in information systems: from positivism to critical realism', paper presented to SAICSIT, Stellenbosch, Western Cape, South Africa, pp. 141-146.
Paulk, M. C., Weber, C. V., Curtis, B. and Chrissis, M. B. (1995), *The Capability Maturity Model: Guidelines for Improving the Software Process*, Addison-Wesley, Boston.
Pfleeger, S. L. and Kitchenham, B. A. (2001), 'Principles of Survey Research Part 1: Turning Lemons to Lemonade', *Software Engineering Notes*, vol. 26, no. 6, pp. 16-18.
Poston, R. M. (ed.) 1996, *Automating specification-based software testing*, IEEE Computer Society Press, Los Alamitos, CA.
Pressman, R. S. (2001), *Software engineering: a practitioner's approach*, McGraw-Hill, NY.
Pretschner, A., Prenninger, W., Wagner, S., Kuhnel, C., Baumgartner, M., Sostawa, B., Zölch, R. and Stauner, T. (2005), 'One Evaluation of Model-Based Testing and its Automation', paper presented to the International Conference on Software Engineering (ICSE 2005), St. Louis, Missouri, USA, pp. 392-401.
Punch, K. F. (1998), *Introduction to Social Research*, SAGE Publications.
Pyhäjärvi, M., Rautiainen, K. and Itkonen, J. (2003), 'Increasing Understanding of the Modern Testing Perspective in Software Product Development Projects', paper

presented to 36th Hawaii International Conference on Systems Sciences, Hawaii.
Robson, C. (2002), *Real World Research, Second Edition*, Blackwell Publishing.
Runeson, P. (2006), 'A Survey of Unit Testing Practices', *IEEE Software*, vol. 23, no. 4, pp. 22-29.
Runeson, P., Andersson, C., Thelin, T., Andrews, A. and Berling, T. (2006), 'What Do We Know about Defect Detection Methods', *IEEE Software*, vol. 23, no. 3, pp. 82-90.
Runeson, P. and Thelin, T. (2003), 'A Case Study Using Sampling to Improve Software Inspection Effectiveness', paper presented to International Symposium on Empirical Software Engineering (ISESE'03), Rome.
Salminen, V., Yassine, A. and Riitahuhta, A. (2000), 'Strategic Management of Complexity in Distributed Product Development', paper presented to NordDesign 2000, Copenhagen.
Schmidt, R. C. (1997), 'Managing Delphi surveys using nonparametric statistical techniques', *Decision Sciences*, vol. 28, no. 3, pp. 763-774.
Schwaber, K. and Beedle, M. (2002), *Agile Software Development with Scrum*, Prentice - Hall, Upper Saddle River, NJ.
Seaman, C. B. (1999), 'Qualitative Methods in Empirical Studies of Software Engineering', *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 557-572.
Smolander, K., Rossi, M. and Purao, S. (2005), 'Going beyond the Blueprint: Unraveling the Complex Reality of Software Architectures', paper presented to the 13th European Conference on Information Systems: Information Systems in a Rapidly Changing Economy, Ragensburg, Germany.
Sommerville, I. (1995), *Software Engineering*, Addison Wesley, Essex, England.
Sommerville, I., Sawyer, P. and Viller, S. (1999), 'Managing process inconsistencies using viewpoints.' *IEEE Transactions on Software Engineering*, vol. 25, no. 6, pp. 784-799.
Spender, J.-C. and Grant, R. M. (1996), 'Knowledge and the Firm: Overview', *Strategic Management Journal*, vol. 17, no. Winter Special Issue, pp. 5-9.
SPSS 12.0 for Windows (2004), SPSS Inc.
Standish (1994), *The CHAOS Report. The Standish Group*, <http://www.projectsmart.co.uk/docs/chaos_report.pdf>.
Stapleton, J. (1997), *Dynamic systems development method - The method in practice*, Addison Wesley.
Strauss, A. and Corbin, J. (1990), *Basics of Qualitative Research: Grounded Theory Procedures and Techniques*, SAGE Publications, Newbury Park, CA.
Suchman, L. (1989), *Notes on Computer Support for Cooperative Work*, University of Jyväskylä, Department of Computer Science, Jyväskylä.

Szulanski, G. (1996), 'Exploring internal stickiness: Impediments to the transfer of best practice within the firm', *Strategic Management Journal*, vol. 17, no. Winter Special Issue, pp. 27-43.

Teece, D. J. (1986), 'Profiting from technological innovation: Implications for integration, collaboration, licensing, and public policy', *Research Policy*, vol. 15, no. 6, pp. 285-305.

Tesch, R. (1990), *Qualitative Research: Analysis Types and Software Tools*, The Falmer Press.

Tinkham, A. and Kaner, C. (2003), 'Exploring Exploratory Testing', paper presented to Software Testing Analysis & Review Conference (STAR) East, Orlando, FL.

Torkar, R. and Mankefors, S. (2003), 'A survey on testing and reuse', paper presented to the IEEE International Conference on Software - Science, Technology and Engineering (SwSTE'03), Herzlia, Israel.

Wallace, L. and Keil, M. (2004), 'Software Project Risks and Their Effect on Outcomes', *Communications of the ACM*, vol. 47, no. 4, pp. 68-73.

Wheeler, S. and Duggins, S. (1998), 'Improving software quality', paper presented to the ACM Southeast Regional Conference, USA, pp. 300-309.

Whittaker, J. and Voas, J. (2002), '50 Years of Software: Key Principles for Quality', *IT Pro*, vol. November December 2002, pp. 28-35.

Whittaker, J. A. (2000), 'What is software testing? And why is it so hard?' *IEEE Software*, vol. 17, no. 1, pp. 70-79.

Voas, J. (1998), 'COTS software: the economical choice?' *IEEE Software*, vol. 15, no. 2, pp. 16-19.