CORE
Provided by U

KUOPION YLIOPISTON JULKAISUJA H. INFORMAATIOTEKNOLOGIA JA KAUPPATIETEET 11
KUOPIO UNIVERSITY PUBLICATIONS H. BUSINESS AND INFORMATION TECHNOLOGY 11

MARKO JÄNTTI

Difficulties in Managing Software Problems and Defects

Doctoral dissertation

To be presented by permission of the Faculty of Business and Information Technology of
the University of Kuopio for public examination in Auditorium,
Microteknia building, University of Kuopio,
on Friday 1st February 2008, at 12 noon

Department of Computer Science
University of Kuopio

KUOPION YLIOPISTO

KUOPIO 2008

Distributor:
Kuopio University Library
P.O. Box 1627
FI-70211 KUOPIO
FINLAND
Tel. +358 17 163 430
Fax +358 17 163 410
www.uku.fi/kirjasto/julkaisutoiminta/julkmyyn.html

Series Editors:
Professor Markku Nihtilä, D.Sc.
Department of Mathematics and Statistics

Assistant Professor Mika Pasanen, Ph.D.
Department of Business and Management

Author's address:
Department of Computer Science
University of Kuopio
P.O. Box. 1627
FI-70211 KUOPIO
FINLAND
Tel. +358 17 162 375
Fax +358 17 162 595
E-mail: Marko.Jantti@uku.fi

Supervisors:
Professor Anne Eerola, Ph.D.
Department of Computer Science
University of Kuopio

Professor Jyrki Kontio, D.Tech.
Department of Computer Science and Engineering
Helsinki University of Technology

Reviewers:
Associate Professor Mira Kajko-Mattsson, Ph.D.
Department of Computer and System Sciences
Stockholm University and Royal Institute of Technology
Sweden

Professor Markku Oivo, D.Tech.
Department of Information Processing Science
University of Oulu

Opponent:
Professor Ilkka Haikala, Ph.D.
Institute of Software Systems
Tampere University of Technology

ISBN 978-951-781-990-9
ISBN 978-951-27-0109-4 (PDF)
ISSN 1459-7586

Kopijyvä
Kuopio 2008
Finland

Jäntti, Marko. Difficulties in managing software problems and defects. Kuopio University
Publications H. Business and Information Technology 11. 2008. 61 p.
ISBN 978-951-781-990-9
ISBN 978-951-27-0109-4 (PDF)
ISSN 1459-7586

ABSTRACT

Many IT organizations are struggling with the increasing number of software problems and defects. The number of software problems and defects has increased due to complex IT systems, new technologies, and tight project schedules. Software quality problems can rapidly increase the costs of software maintenance and development. Unfortunately, support teams of IT organizations have limited resources for resolving software problems and defects. Often, they do not have well-defined process models for problem management. Additionally, traditional defect management models are not adequate to service-oriented software business in which problem resolution requires communication between several service providers.

The process of managing problems and defects includes a large number of difficulties and challenges but has been given little consideration in software engineering research. The research work of this thesis is organized around four goals. The first goal is to study software quality assurance methods that can be used to detect defects. The second goal is to identify the difficulties that organizations have in managing software defects and which improvements are needed to existing defect management models. The third goal is to study the concepts of service-oriented problem management. The fourth goal is to study challenges and difficulties of service-oriented problem management methods.

In this thesis, we have examined software quality assurance methods with a strong focus on UML-based testing and studied how early test case planning helps to detect defects and problems. We have identified difficulties that IT customers and IT providers have regarding defect management. We have introduced a service-oriented problem management model that describes the concepts of problem management, the relationships between concepts, and connections between problem management and other service support processes. The model has been tested and evaluated in practice with several case organizations. As a part of the model, a checklist for evaluating problem management was created and a knowledge base for proactive problem management was established.

The main contributions of this thesis are 1) evaluation of a UML-based test model as a defect detection technique, 2) a documented list of challenges and difficulties regarding defect management, 3) a systematic approach on service-oriented problem management with recommendations and guidelines, and 4) a list of difficulties concerning service-oriented problem management. These contributions can be used by problem managers and quality managers to improve problem management processes.

Universal Decimal Classification: 004.052.4, 004.415.5, 004.415.53, 006.015.5
Inspect Thesaurus: computer software; software quality; software process improvement; quality assurance, quality management; software management; errors; error detection; Unified Modeling Language; program testing

Acknowledgments

In the fall 2003, Professor Anne Eerola encouraged me to start writing a doctoral thesis after I had finished my master's thesis on "Test case design based on UML models". I have always been interested in defects and problems. Therefore, the process of managing defects and problems was an obvious choice of research topic. Now, it is time to give credit to the people who supported me during the research process.

This thesis is a result of research carried out in the Department of Computer Science at the University of Kuopio. First of all, I want to thank my supervisor, Professor Anne Eerola, for excellent guidance and support when I was writing this thesis. I would also like to thank Associate Professor Mira Kajko-Mattsson and Professor Markku Oivo for their reviews and valuable comments regarding the thesis.

I would like to thank all my research colleagues and the students that I have worked with: Tanja Toroi, Aki Miettinen, Harri Karhunen, and Niko Pylkkänen who helped me during the research process. I sincerely thank the organizations that participated in the PlugIT project and Service Oriented Software Engineering project. Many thanks to the co-writers and research partners from Kuopio University Hospital, TietoEnator Forest & Energy, Savon Voima, Navicore, and Softera. Special thanks to Kari Kinnunen, Kyösti Vähäkainu, Pirkko Pessi, Juha Vaaranmäki, and Jukka Kaukola for their contribution.

My sincere thanks go also to the Department of Computer Science at the University of Kuopio for providing an excellent working environment. I would like to thank Risto Honkanen, Paula Leinonen and Janne Nieminen who gave me support in using Latex. Additionally, I would like to thank Kenneth Pennington from the HUT Language Center for the language inspection of this thesis.

This work was supported financially by TEKES, the European Regional Development Fund and the partner companies of the PlugIT project and the Service Oriented Software Engineering (SOSE) project.

Finally, I want to thank my family and my wife Elina for their patience, support and love.

Abbreviations and notations

Abbreviation	**Description**
CAB	Change Advisory Board
CI	Configuration Item
CMDB	Configuration Management Database
CMM	Capability Maturity Model
FAQ	Frequently Asked Question
ITIL	IT Infrastructure Library
KB	Knowledge Base
RFC	Request For Change
SLA	Service Level Agreement
SLM	Service Level Management
SPOC	Single Point of Contact

List of the original publications

This thesis is based on the following published articles, which are referred to in the text by their Roman numerals I – VI:

I. M. Jäntti, T. Toroi, UML-Based Testing: A Case Study. In: Koskimies K, Kuzniarz L, Lilius J, Porres I (editors). *Proceedings of NWUML'2004. 2nd Nordic Workshop on the Unified Modeling Language*, pages 33-44, Turku, Finland, August 19-20, Turku: Turku Centre for Computer Science, TUCS General Publication 35, 2004.

II. M. Jäntti, T. Toroi and A. Eerola, Difficulties in Establishing a Defect Management Process: A Case Study. In: Münch Jürgen, Vierimaa Matias (editors), *Product-Focused Software Process Improvement, 7th Conference, PROFES 2006*, pages 142-150, Amsterdam, The Netherlands, June 12-14, 2006, Germany: Springer Verlag, 2006.

III. M. Jäntti, A. Eerola, A Conceptual Model of IT Service Problem Management. *Proceedings of the IEEE International Conference on Service Systems and Service Management ICSSSM'06*, pages 798-803, Troyes, France, 25-27 October, 2006.

IV. M. Jäntti, A. Miettinen, K. Vähäkainu, A Checklist for Evaluating the Software Problem Management model, a Case Study. In: *The IASTED International Conference on Software Engineering SE 2007*, pages 7-12, Innsbruck, Austria, February 12-15, 2007.

V. M. Jäntti, K. Kinnunen, Improving the Software Problem Management Process: A Case Study. *Proceedings of European Systems & Software Process Improvement and Innovation*, pages 40-49, Joensuu, Finland, October 11-13, 2006. Germany: Springer Verlag, 2006.

VI. M. Jäntti, K. Vähäkainu, Challenges in Implementing a Knowledge Base for Software Problem Management. *Proceedings of the IASTED International Conference on Knowledge Sharing and Collaborative Engineering (KSCE 2006)*, pages 63-68, St. Thomas, US Virgin Islands. November 29-December 1, 2006.

Contents

1 Introduction **13**

2 Research Methodology **15**
- 2.1 Research framework 15
  - 2.1.1 Traditional software quality assurance 15
  - 2.1.2 Service-oriented quality assurance 16
- 2.2 Research questions 18
- 2.3 Research methods 19
- 2.4 Research process and phases 21

3 Quality assurance in software engineering **23**
- 3.1 Traditional quality assurance methods for finding problems and defects 24
- 3.2 Difficulties in managing problems and defects 26

4 Service-oriented problem management **29**
- 4.1 Background for problem management 29
- 4.2 Basic concepts of problem management 30
- 4.3 Reactive and proactive problem management 32
- 4.4 Problem management tools 36
- 4.5 Metrics and the process maturity 37
- 4.6 Connections to other service management processes 39
- 4.7 Difficulties in service-oriented problem management 40

5 Summary of papers **43**
- 5.1 Relation of research papers and projects 43
- 5.2 Summary of papers 44
  - 5.2.1 UML-Based Testing: A Case Study 44
  - 5.2.2 Difficulties in Establishing a Defect Management Process: A Case Study 45
  - 5.2.3 A Conceptual Model of IT Service Problem Management . . 46
  - 5.2.4 A Checklist for Evaluating the Software Problem Management Model 46
  - 5.2.5 Improving the Software Problem Management Process 46

5.2.6 Challenges in Implementing a Knowledge Base for Software Problem Management 47
5.3 Summary of the results 47

6 Conclusions **51**
6.1 Contributions of the thesis 51
6.2 Future work 53

Bibliography **54**

Chapter 1

Introduction

Software problems have become a part of our daily life. During a working day and free time we face numerous software problems, including application failures, security bugs, errors in user documentation, poor usability, availability and performance problems associated with IT services. Problems such as these are very common because the IT systems are used everywhere.

The National Institute of Standards and Technology has estimated that software defects and problems annually cost 59.5 billions the U.S. economy [24]. The rework in software projects (problem resolution and bug fixes) leads to higher software development and maintenance costs and and higher prices for IT services and products. In addition to the increasing number of problems and defects, the quality of the process of managing problems and defects needs to be improved.

Problems concerning quality in a problem management process can lead, for example, following symptoms: end users do not know which contact person is the correct one and report problems to the wrong service desk; customers claim that the IT provider does not react to the problems reported by customers; the number of open problems rapidly increases in relation to closed problems at the service desk; and customers have to wait a long time for solutions to problems.

An interesting question is why the above quality problems still exist in software engineering? Previous studies have presented dozens of software quality assurance methods, including reviews [91], inspections [12, 25], pair programming [80], various testing approaches (module testing, integration testing, system testing, and acceptance testing) [61], risk management techniques [57], and defect management methods [22, 13]. The answer might be that most of these methods were developed to assure quality in product-oriented software development, not in service-oriented business.

Why do we need service-oriented problem management? First, traditional defect management and testing primarily focus on managing and resolving defects related to program code and other software artifacts. These defects are often directly assigned to developers that have to resolve defects. Defect resolution decreases the time that was originally reserved for developing new modules. In service-oriented problem management, all the support requests are first sent to the service desk that is able to resolve many problems without interrupting developers' work.

Second, IT organizations world-wide have started to update their product-oriented software development models to service management models, such as IT Infrastructure Library (ITIL) [73] and COBIT [15]. Hence, old product-oriented and project-oriented quality assurance methods need to be updated to service-oriented quality assurance methods. Instead of software product quality, today's IT managers emphasize service quality. Service-oriented problem management uses the same concepts than service management standards and frameworks. Thus, it is easier to integrate a service-oriented problem management model into the organization's service management framework than a traditional defect management process.

Third, service-oriented problem management provides features that are either completely omitted or poorly defined by defect management frameworks, such as how to use service level agreements to define target problem resolution times, and how to use a knowledge base as a self-service desk. Service-oriented problem management can be used both for managing non-code-related software problems and software defects. It combines the customer-centric problem management activities of the service desk with development-centric defect management activities.

Finally, service-oriented problem management emphasizes the customer's active role in problem management and service quality improvement. An *informed customer* concept in service management means that customers should receive enough information from an IT provider during problem handling, such as a confirmation that a customer's support request has been accepted by the service desk, and notifications of problem status changes. Additionally, customers should be able to review problem resolution before the problem is closed.

There is a clear need for service-oriented models of software problem management that could help IT organizations both to handle already reported problems in IT services as well as prevent problems before they occur.

The summary consists of five chapters. Chapter 2 describes the research framework, research questions, research methods, and the research process. In Chapter 3, we introduce the most common quality assurance methods of software engineering and discuss the challenges and bottlenecks related to the process of managing problems and defects. Chapter 4 presents the service-oriented problem management approach with the description of problem management activities, roles, metrics, and connections to other support processes. Chapter 5 summarizes the original papers and provides a summary of research results. Finally, Chapter 6 draws conclusions from this thesis.

Chapter 2

Research Methodology

In the following sections, we first describe the research framework with two viewpoints: traditional software quality assurance and service-oriented quality assurance. Second, the research questions are defined. Third, we describe the research methods used in this thesis. Finally, the research process and its phases are described.

2.1 Research framework

Currently, there are two main approaches that IT organizations use to assure quality and to manage defects and problems: 1) traditional software quality assurance (including defect management) and 2) service-oriented quality assurance (including problem management). These two approaches are introduced in the following sections.

2.1.1 Traditional software quality assurance

The traditional software quality assurance methods include testing, reviews, inspections, defect management models and techniques, such as defect prevention [64], the defect management process [81] and root cause analysis method [59, 13]. In addition to these methods, Horch [32] emphasizes standards, vendor management, education of software people, safety and security issues, and risk management. Software quality assurance methods can be organized into a *software quality system (SQS)*. There are two goals for the software quality system [32]: to build quality into the software from the beginning and to keep the quality in the software throughout the software life cycle.

This thesis focuses on the process of managing defects. We will not deal with safety and security issues and risk management. Defect management is a research field that includes many terms (defects, errors, bugs, faults, failures, and problems) that are difficult to distinguish from each other. The definitions that we use are based on international standards. Occasionally, software developers (programmers and designers etc.) make *errors* or *mistakes* that are actions that result in software containing a fault [21, 34].

A *fault* (or a bug) is “an accidental condition that causes a functional unit to fail to perform its required function” [34]. A fault may cause a failure. Failures occur during the execution of a software program. According to IEEE standard 729-1983 [34] a *failure* is “an event in which a system or system component does not perform a required function within specified limits”. Failures are caused by faults.

Moreover, a very common term in software engineering is a defect. A framework for counting problems and defects defines a *software defect* as “any flaw or imperfection in a software work product or software process” [21]. Another definition is given by the IEEE Standard Dictionary of Measures to Produce Reliable Software [35]. It defines defects as product anomalies such as omissions and imperfections found during early life-cycle phases and software faults.

We shall define a *defect management process* as a well-defined and documented process for preventing, finding, and resolving defects. Defect management is a traditional way to see the process of handling defects. Defect management focuses on preventing defects and resolving existing defects. Typical adopters of this approach are product development units that focus on programming and testing software and correcting bugs.

Defect management models usually include numerous references to other traditional quality assurance techniques. For example, the defect management process of the Quality Assurance institute [81] emphasizes the importance of risk management in identifying critical problem areas in software products and the software development process. Risk management typically involves identifying critical risks, estimating expected impact and minimizing the estimated impact of risks [57].

2.1.2 Service-oriented quality assurance

Many IT departments and IT organizations have started to use IT service management frameworks in order to manage both the services they provide their customers and the services that they purchase from third-party service providers. Support and maintenance services for software products and application services are good examples of IT services. Problem management plays an important role within the support and maintenance process in software engineering. A recent industrial IT service management survey [63] showed that IT organizations consider support processes (configuration management and problem management) as key development targets in the near future.

Additionally, a systematic approach for managing defects and problems helps to decrease support and maintenance costs and the amount of rework in software development. Thus, it increases the effectiveness and the efficiency of the support processes. Problem management aims to minimize the impact of software problems and defects on the business and to identify the root cause of these problems [73]. A *software problem* can be defined as “any problem that a customer or a user encounters while using a software product or an IT service” [21]. Hence, the definition of a problem encompasses software problems, hardware problems, and other problems, such as operational ones.

Problem management has received little attention in the academic literature. Rather, most research has focused on defect management. The following studies on problem tracking, software maintenance and support, and service management have dealt with problem management. Cunningham [18] has examined the problem tracking and resolution process at the university of Kansas and emphasized the importance of a problem tracking tool: “A problem logbook is also more efficient, clearer, more consistent in format than the paper notebook could ever be”.

Similarly, Kanter and Jones [53] have presented the features of the problem-tracking system in the California State University network. They state that “The problem resolution process has given our group a more service-oriented focus and enhanced reputation with our campus members”. Gruhn and Urbainczyk [28] have studied the development of a business process-based problem-tracking system that can handle both internal problem reports created by the quality assurance team and external problem reports created by customers. Sandusky and Gasser have studied the process of managing software problems in distributed software environments [84].

In software maintenance studies, problem management has been classified as corrective software maintenance [46, 50]. Kajko-Mattsson et al. [48] have developed the corrective maintenance maturity model (CM3) that includes problem management. Their framework was developed in cooperation with ABB. Kajko-Mattsson notes that main functions of problem management are collecting information of problems, identifying defects related to problems and removing defects [46]. Additionally, April et al. [3] have presented software maintenance capability maturity model SM CMM. Finally, service management studies dealing with service levels and service level agreements [89, 29] are also valuable for problem management research. However, none of these studies have thoroughly examined the difficulties that might arise in the problem management process.

The most comprehensive description of the problem management process can be found in the IT service management framework IT Infrastructure Library (ITIL) [73]. For each service management process, ITIL has defined goals, benefits, basic concepts, process activities, metrics, and roles. ITIL framework categorizes problem management activities into two dimensions: reactive and proactive problem management. Reactive problem management is focused on resolving reported incidents and problems whereas the goal of proactive problem management is to prevent incidents before they occur.

An *incident* can be defined as "any event which is not part of the standard operation of a service and which causes, or may cause, an interruption to, or a reduction in the quality of that service" [73]. A *problem* is defined in ITIL as “the unknown underlying cause of one or more incidents” [73]. Reactive problem management is further divided into problem control and error control activities where the problem control is responsible for identifying the root cause of a problem and defining a temporary solution (work-around) for the problem.

Problem control aims to convert problems into known errors. A *known error* is “an incident or problem for which a root cause is known and for which a temporary work-around or a permanent alternative has been identified” [73]. The error control activity is responsible for processing known errors. Additionally, there are other service management frameworks (CoBIT [15]) and company-specific IT service management models (Microsoft Operation Framework [66]) available for problem management.

2.2 Research questions

This thesis answers four main research questions:

1. Which methods can be used to detect defects?
2. What types of difficulties do IT organizations have regarding defect management?
3. What is service oriented problem management and which concepts are related to it?
4. What types of difficulties do IT organizations have in introducing service-oriented problem management?

Why is it important to answer those research questions? The first question is important because current software projects spend about 40-50 percent of their effort on avoidable rework [9]. By rework, Boehm means, for example, fixing defects that could have been detected earlier in software development and fixed less expensively. All stages of testing are important for detecting defects. Because substantial research has been made in the area of system testing, we do not deal with it in this thesis. However, relatively little research effort [47] has been spent on how developer-testing should be conducted and how UML diagrams support developer-testing and user-side testing.

Many studies on software testing recommend that testing should be started in the early phase of software life-cycle [61, 65, 83]. It is much cheaper to fix problems in the design phase than in the maintenance phase. Similarly, the service transition process of the ITIL emphasizes that it is important to start service validation, including testing, early in the service lifecycle [77]. McGregor and Korson [65] state that existing work products, such as UML diagrams, can be used as the basis of deriving test cases.

The second research question is important because the difficulties regarding defect management have been given very little attention in the software engineering literature. We argue that organizations have several difficulties and challenges regarding defect management. Information on difficulties in the current defect management processes helps software development teams to improve the process of managing defects.

Unfortunately, a defect management model covers only part of the activities associated with customer support. Defect management is a development-centric process that does not define how user problems are managed or how problems are caused by defects. There is a need for a well-defined, documented problem management model that combines the processes of problem management and defect management together and is suitable for service-oriented software businesss.

Concerning the third research question, more research efforts are needed to examine service-oriented problem management, as it is a relatively new approach compared to defect management. IT organizations need well-defined process models for managing problems and defects. A service-oriented approach introduces new concepts that need to be mapped into traditional quality assurance concepts thus highlighting the need for a conceptual model describing service-oriented problem management.

The fourth research question addresses the need of IT organizations for guidelines on how to introduce service management processes, including problem management, effectively and efficiently. It is important to publish research results on the challenges arising from the introduction of service management processes because these results help other IT organizations to identify and avoid similar problems in process implementation projects. Furthermore, these results help organizations to establish the basis for continuous process improvement concerning problem management.

2.3 Research methods

There are several research approaches and methods that can be used in software engineering research. Järvinen and Järvinen [45] categorize research approaches that study reality into conceptual-analytical and empirical approaches. *Conceptual-analytical* studies focus on analyzing existing constructs, identifying theories, models and frameworks of previous studies, and performing logical reasoning. *Empirical approaches* can be further divided into 1) those that examines the current state and past (including a theory-testing approach and a theory-creating approach) and 2) those that take a constructive approach.

The *theory-testing approach* includes field studies, surveys, and laboratory experiments, while a *theory-creating approach* includes normal case studies, and grounded theory studies. This thesis uses both conceptual-analytical approaches and empirical approaches. According to Järvinen [44] case studies belong to either theory-creating or theory testing approaches. Yin [93] defines a case study as “an empirical inquiry that investigates a contemporary phenomen within its real-life context, especially when the boundaries between the phenomenon and the context are not clearly evident”. Using a case study as a research method has been criticized for a number of reasons. It has been claimed that a case study lacks the academic *rigor* (the control of research), research results based on case studies cannot be generalized, and that a case study requires a lot of resources and experienced case study researchers. We can answer to these claims by using Yin’s arguments concerning the case study method [93]. To increase the amount of rigor in our case studies, we studied the recommendations for a good case study research provided by Eisenhardt [19] and Yin [93], cited in Järvinen and Järvinen [45]. Yin [93] states that instead of statistical generalization, the case study method provides the possibility to make analytical generalizations and thus extend theory. Regarding the last claim, a case study requires considerable resources and good case study researchers. However, this challenge also affects other research methods.

In Paper I, the purpose was to test a theory-based UML test model in practice and to collect experiences regarding its strenghts and weaknesses. Thus, the paper has characteristics of both theory-creating and theory-testing approaches. We explored how test cases based on behavioral UML diagrams can be used to detect software problems and defects. UML-based testing well supports traditional software quality assurance methods, such as reviews and inspections. Paper II was more a theory-creating paper that presented a list of defect management challenges. Paper III introduced a conceptual model for IT service problem management. The model was produced by analyzing previous models and frameworks. It used a conceptual-analytical approach with a constructive research method. Paper IV presented a checklist for problem management and experiences on applying the checklist. It used a combination of constructive research method and a theory-creating case study method. In Paper V we used a theory-creating case study method to improve the problem management process of a case organization. Paper VI used both a case study research method and a constructive research method (with a knowledge base as a construction). We selected case organizations from the pool of our research partner organizations. The organizations that were interested in software quality assurance were selected. Additionally, our goal was to choose customer supplier pairs for our research in order to create an fruitful environment for process improvement. Thus, a convenience sampling was the main case selection method.

Eisenhardt [19] points out that although data analysis plays a very important role in creating theory of case studies, it is the most difficult part of the research process. Eisenhardt introduces two different analysis techniques: a within-case analysis and a cross-case analysis. The basic idea of the *within-case analysis* is to examine cases carefully as stand-alone entities before making any generalizations. The *cross-case analysis* in turn aims to search cross-case patterns [19]. In Paper II, we used a cross-case analysis, while the other case studies focused on a within-case analysis.

A case study method was selected because it provides a deeper understanding of the factors related to process improvement challenges than that offered by large surveys. Additionally, those organizations participating in the research project recommended using other data collection methods rather than long question forms. Our case studies in papers IV-VI had features of an action research method [4] because the researchers were involved as active participants, together with the employees of case organizations, in improving the process and identifying bottlenecks in the working methods. However, action research requires several research cycles which was not possible in our case due to limited time resources.

Figure 2.1 describes the research areas from which the literature was selected.

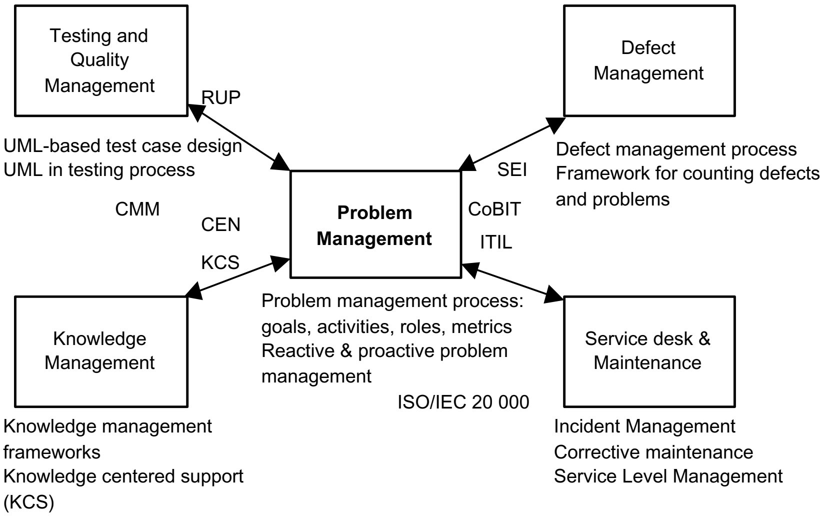

Testing and Quality Management
RUP
UML-based test case design
UML in testing process
CMM
CEN
KCS
Defect Management
Defect management process
Framework for counting defects and problems
SEI
CoBIT
ITIL
Problem Management
Problem management process: goals, activities, roles, metrics
Reactive & proactive problem management
ISO/IEC 20 000
Knowledge Management
Knowledge management frameworks
Knowledge centered support (KCS)
Service desk & Maintenance
Incident Management
Corrective maintenance
Service Level Management

Figure 2.1: Literature used in the research process.

The context in which the studies were conducted was the service support interface between IT companies and IT customers. In Paper I, the research was focused on testing techniques [61, 79, 58] and model-based testing [30, 7]. In paper II, defect management studies played an important role. Two frameworks were found to be particularly valuable for this thesis: the defect management process of the Quality Assurance Institute [81] and the framework for counting problems and defects of the Software Engineering Institute. In Paper III, the research focus shifted to problem management [73] and other service management processes [72, 70, 71]. Finally, the problem management became the central focus of the research and the literature search was expanded to include knowledge management frameworks [14, 11], knowledge base studies [27, 85] and service desk and maintenance studies [84, 46, 48].

2.4 Research process and phases

Figure 2.2 describes the four phases of the research process and how research papers are related to the research process.

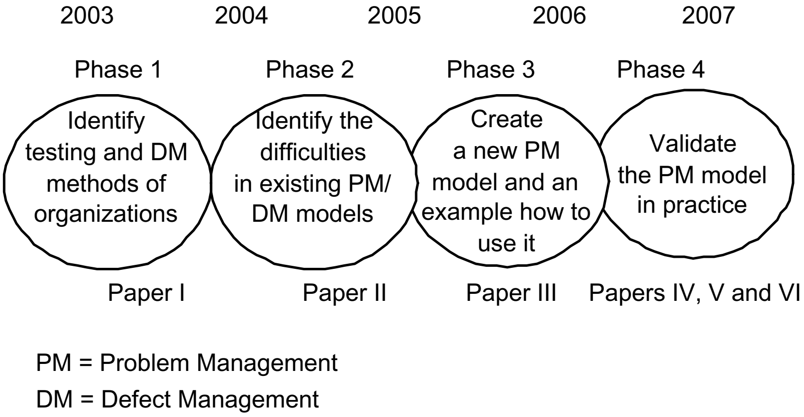

2003
2004
2005
2006
2007
Phase 1
Phase 2
Phase 3
Phase 4
Identify testing and DM methods of organizations
Identify the difficulties in existing PM/ DM models
Create a new PM model and an example how to use it
Validate the PM model in practice
Paper I
Paper II
Paper III
Papers IV, V and VI
PM = Problem Management
DM = Defect Management

Figure 2.2: Phases of the research process.

In the first phase, we focused on early test case planning as a defect detection method. It is much cheaper to remove potential problems from the products in the design phase rather than after delivery of the software to the customers. The results of the phase 1 were reported in Paper I. The study described in Paper I included one case organization: an IS department of a hospital. Data collection methods included interviews with customer-side testers and observations conducted during the testing experiment (a researcher had access to the test version of a health care application). The first phase provided the following results: evidence that that the customer support interface contained several targets for improvement (including problem and defect reporting procedures) between IS customers and IT providers, and documented experiences on how UML-based testing can be utilized in practice to detect software defects.

In the second phase, we identified the difficulties encountered in establishing a defect management process. The reporting of this phase includes Paper II describing a case study with four case organizations: an IT service company, a project-oriented software company, an energy company, and the IS department of a hospital. Data collection methods included personal interviews that were based on a question form. Data analysis was performed by tabulating data on the cases, comparing the results between cases, and looking for similarities and differences in results. The list of identified challenges comprised the main result of the second phase. Additionally, we recognized the need for a more customer-centric approach for managing defects and problems and thus started to examine the problem management process of the ITIL framework.

In the third phase, we developed a conceptual model that describes the concepts of problem management, the relationships between concepts, and their connections to other service support processes within service management. The conceptual model established the basis for problem management research. The model also helped us to describe ITIL concepts to IT organizations and IT customers within the SOSE project. In this phase, we used a constructive research method to establish the model and a design pattern (Coplien Form) method [17] to document the pattern. The results of the third phase (a conceptual model for IT service problem management and an example on how to use it) were reported in Paper III. At the end of the third phase, we started to study whether IT service management processes could be supported by knowledge management frameworks.

The fourth phase focused on applying the service-oriented problem management model in practice. The results of this phase were reported in Papers IV, V and VI. Because the conceptual model of problem management was a theoretical model, guidelines were needed to apply it in practice. Hence, we constructed a checklist of issues considered important in establishing an ITIL-based problem management model (Paper IV). Using the model and checklists together, IT organizations are able to create an effective problem management process. In Paper V, we explored the strenghts and challenges of problem management in a case organization. In the fourth phase, a knowledge base for proactive problem management was developed and implemented. The knowledge base includes information concerning known problems and their resolutions. In Paper VI, we presented a list of the challenges that were identified during a knowledge base implementation project.

Chapter 3

Quality assurance in software engineering

Software Engineering can be understood as a systematic approach to the analysis, design, implementation and maintenance of software. The goal of software engineering is to build a new software product or improve existing products [41]. Thus, a *software development process* focuses on transforming customer's requirements into software artifacts (a software product and its documentation) and transforming changes in those requirements into new product versions. Each software development organization has its own unique software development process that is usually based on a software life-cycle model.

A software development process is an evolving concept. Various software life-cycle models have become available for software development organizations. These life-cycle models differ from each other in their use of different methodologies and types of phases, or a number of phases. Royce has defined the three generations of software development [82]: conventional (1960s-1970s), transition (1980s-1990s) and modern software development (2000 and later). The waterfall model is a classic example of a conventional model consisting of six phases: requirements, analysis, program design, coding, testing, and maintenance. Transition models include object-oriented models and the spiral model defined by Boehm [10]. A Rational Unified Process is a widely used, modern software development model that consists of four iterative phases: inception, elaboration, construction, and transition [41]. Additionally, modern models include agile development models, such as Agile Unified Model [1]. Despite the transition from old process models to more modern process models, the same problems remain in software development: software projects are not completed within budget, on schedule and nor do the output of the project meet business requirements [6].

Lientz and Swanson [60] have categorized maintenance activities into four classes: *adaptive* (implementing changes in software environment), *perfective* (dealing with users' new requirements), *corrective* (fixing errors and defects), and *preventive* maintenance. Bennet and Rajlich [5] use the term *software evolution* for those maintenance activities performed after the initial development. They note that software evolution aims to adapt the application to ever-changing user requirements and operating environment. It also aims to correct the application faults and respond to both developer and user learning.

According to Wallmueller [91], software quality is comprised of two aspects: 1) the quality of the software product and 2) the quality of the development process. Software quality includes characteristics such as maintainability, user-friendliness, reliability, efficiency, portability, and modularity. However, in addition to process features and product features, today's IT people emphasize service features, such as IT service availability and IT service performance.

Various standards, theoretical frameworks and models regarding customer support, problem management, and defect management can be used to improve the process of managing problems and defects. Previously mentioned software development lifecycle models (Rational Unified Process) include information on quality assurance methods such as testing, risk management and defect prevention. Quality standards, such as the IEEE Standard Classification for Software Anomalies [33], IEEE Standard Dictionary of Measures to Produce Reliable Software [35], and ISO 20 000 service management standard [36] provide standard definitions and auditable requirements for processes.

Maturity models are designed for measuring the maturity level of software development processes or service management processes. Perhaps the most well-known maturity model in software engineering is Capability Maturity Model CMM [42]. There is also a specific CMM model for IT service management processes [68]. IT service management frameworks can be used to define how to perform support processes (ITIL [73], COBIT [15], and Microsoft Operations Framework [66]). Finally, there is a wide selection of other quality assurance models such as Defect Management Process of Quality Assurance Institute (QAI) [81], and the Framework for Counting Problems and defects by Software Engineering Institute (SEI) [21]. For our purposes, the most useful models are those process models that clearly define the activities of problem management and defect management (ITIL problem management, COBIT, the QAI model and the SEI model).

3.1 Traditional quality assurance methods for finding problems and defects

The IEEE Standard Classification for Software Anomalies [33] states that anomalies (problems and defects) may be found during the review, test, analysis, compilation, or use of software products or documentation. Similarly, the Framework for Counting Problems and Defects by the Software Engineering Institute (SEI) emphasizes the importance of software product synthesis, inspections, formal reviews, testing and customer service in finding problems and defects [21]. These defect finding methods will be briefly described below. First, we discuss the types of difficulties and challenges associated with these methods followed by suggestions for improvements to existing defect management models.

Software product synthesis: SEI's framework defines the software product synthesis as "the activity of planning, creating and documenting the requirements, design, code, user publications, and other software artifacts that constitute a software product" [21]. This definition should also include IT services in addition to products, since many IT customers and IT providers see software products as services.

Software inspection is a formal evaluation technique in which requirements, design and source code are examined to detect defects [12]. The important role of inspections has already been noted in the 1970s. The following steps can be identified in the inspection process: Entry, Planning, Kickoff meeting, Individual checking, Logging meeting, Edit, Follow up, Exit, and Release [25]. Inspections are an efficient way to find defects from documentation but require an experienced inspection leader that is able to produce appropriate checklists and metrics for the inspection process, and organization-wide rules and guides.

Formal reviews include code walkthroughs and defect causal analysis (DCA) meetings. The defect causal analysis method is based on the data received from a software problem report. The DCA approach has three major principles [13]:

1. *Reduce defects to improve quality:* software quality can be improved if the organization focuses on preventing and detecting defects in early phase of software development.

2. *Apply local expertise:* people who really know the cause of the failure and how to prevent problems in the future should participate in causal analysis meetings.

3. *Focus on systematic errors:* DCA people should select a sample of systematic problems from a problem database to be reviewed because support resources are limited and it is impossible to bring all problems and defects into DCA meetings.

A DCA meeting consists of the following steps: select problem sample, classify selected problems, identify systematic errors, determine principal cause, develop action proposals, and document meeting results [13].

Software testing is a process of executing a program on a set of test cases and comparing the actual results with expected results. The testing process requires the use of a test model that describes what should be tested and how the testing should be executed [58]. Previous studies have emphasized the need of shifting testing to early phases of the software development process, such as requirements, analysis [61], and design [7].

UML-based test models are used to create test cases based on UML (Unified Modeling Language) models of the system. In previous studies, the UML-based test model has been used for such tasks as component integration testing [30] and generating test cases from UML statecharts [52]. Salem and Balasubramanian recommend that the following steps be used to generate UML-based test cases [83]: study the software requirements, develop use cases with scenarios (including system's responses to inputs), generate test cases based on UML use cases, execute generated test cases, and evaluate and analyze the results. In Paper I of this thesis, we studied whether the UML-based test model provides useful information for software testing in practice. According to our experiences use case diagrams together with use case scenarios provide descriptions of exceptional and alternative flows that are often sources of defects. Activity diagrams show different action flows that a tester should test (activity coverage), thus making software testing more systematic with a test model. Furthermore, state diagrams can be used to measure the transition coverage (the tester must go through each state transition).

Defect management techniques including defect analysis create a bridge between product-oriented software quality control and process-oriented software quality assurance [32]. The defect management process of the Quality Assurance Institute includes activities such as defect prevention, defect discovery, defect resolution and process improvement. It especially emphasizes the role of defect prevention [81]. Defect prevention activity involves the analysis of defects that were encountered in the past and defining and implementing actions to prevent the occurrence of those defects in future projects. Trends are analyzed to track the types of defects that have been encountered as well as to identify defects that are likely to recur. Defect discovery describes the techniques used to find defects. A defect is considered to have been discovered once it has been formally brought to the attention of the developers, and the developers acknowledge that the defect is valid.

Defect resolution consists of prioritizing and scheduling the fix, fixing the defect and reporting the resolution [81]. When the defect has been fixed, the vendor has to send a defect resolution report to customers, end-users and system developers to inform that the defect no longer exist, which parts of the system were fixed, and how the fix would be made available, such as a website for downloads.

3.2 Difficulties in managing problems and defects

Most of the research that has been conducted in software quality assurance has focused primarily on introducing new quality assurance models and techniques for finding defects and testing them in practice. However, difficulties and challenges associated with these quality assurance methods have not been studied comprehensively. Nevertheless, the following challenges can be extracted from previous studies. Inspections and reviews are considered to be useful, though expensive quality assurance methods since they require considerable resources [90]. Miller et al. point out that the inspection process requires computer-based monitoring facilities [67]. An additional challenge is how to estimate the number of remaining faults after an inspection [88]. In terms of risk management the problem often entails the quantification and ranking of risks [57].

Testing-related challenges include, for example, how testing can be started at early phases of the software project [7], test tools cannot test the whole application or testing is not given enough resources, time, or priority until initial development is completed [78]. Ahonen et al. [2] have reported that the problems of testing are more related to the organizational model rather than technical problems with testing. For example, it is impossible to ensure that all teams use good practices and it is difficult to get skilled people from the other teams when needed. Kajko-Mattsson and Björnsson [47] have reported that organizations do not have well documented developers' testing processes. As a solution, they provide a framework that describes most of the activities of developers' unit testing and unit integration testing. There are also challenges in specifications-based testing, such as in using a UML-based testing method. First, testers and developers may not have the necessary domain knowledge [87] to create a comprehensive test model. Second, testers need to be trained to use and understand various UML-diagrams. One UML diagram does not provide enough information for testing. UML diagrams have to be well-planned and documented in order to be useful for testing purposes.

Surprisingly little research has focused on the difficulties arising during the process of managing problems and defects. The results of the QAI research report [81] indicate that the key problems regarding defect management are the lack of well-defined defect management processes, the lack of anyone who would be responsible for tracking and reporting on defects, the lack of a common vocabulary to describe defects, and a lack of an accepted set of defect reports or metrics. According to Boehm and Basili [9], avoiding defects is difficult due to the software's complexity and accelerated project schedules. The Framework for Counting Problems and Defects by the Software Engineering Institute [21] addresses the variety of defect finding activities and related problem reports leading to difficulties in communicating clearly. Additional challenges have been a lack of list of known errors in delivery documentation [31] and too simple defect classification categories [56].

Theoretical models for managing problems and defects also need improvements. Although the QAI model [81] provides a comprehensive description of defect management activities and references to other quality improvement techniques, such as testing, early test case design, inspections, and risk management, it poorly defines the connection between service desk and customer support activities. However, it is noted that users might report software problems that would never become defects.

The Framework for Counting Problems and Defects by the Software Engineering Institute [21] provides information on how to find problems and defects and information on the metrics that can be used for problem management and defect management processes. The strenght of the SEI model lies in its discussion of the differences between problems and defects, a comprehensive list of problem attributes and examples of how to use the metrics. The major weakness of this model, however, is the absence of process diagrams for problem management.

Because there is a clear research gap in the defect management research, the second research goal of this thesis is to determine the types of difficulties associated with managing software problems and defects. In Paper II of this thesis, we presented the results of a case study that examined the difficulties in implementing defect management in four case organizations.

Chapter 4

Service-oriented problem management

In this chapter, we describe the background of the service-oriented problem management approach and its the basic concepts, activities, roles, metrics and tools. This chapter provides an answer to the third research question: what is service-oriented problem management and which concepts are related to it?

4.1 Background for problem management

IT Service management is divided into two sections: service delivery processes and service support processes. *Service delivery* consists of service level management, capacity management, financial management for IT services, availability management and IT service continuity management. The processes of *service support* include incident management, problem management, change management, configuration management, and release management. In this thesis, the main focus is on problem management. Perhaps the most comprehensive description of the problem management process can be found in the IT service management framework ITIL. The ITIL problem management section describes process goals, scope (inputs, outputs), benefits, basic activities, and the metrics for problem management. There are also other service management frameworks that include problem management, such as COBIT (Control Objectives for Information and related Technology). COBIT provides management guidelines for service management processes such as Deliver and Support processes (DS) including problem management (DS10) [15]. COBIT includes a detailed description of control objectives for the problem management process. It defines process inputs (change authorisation, incident reports, IT configuration details, error logs), outputs (request for change, problem records, process performance reports and known problems, known errors and workarounds) functions, goals and metrics for problem management. However, COBIT process descriptions are not as broad as in the ITIL framework.

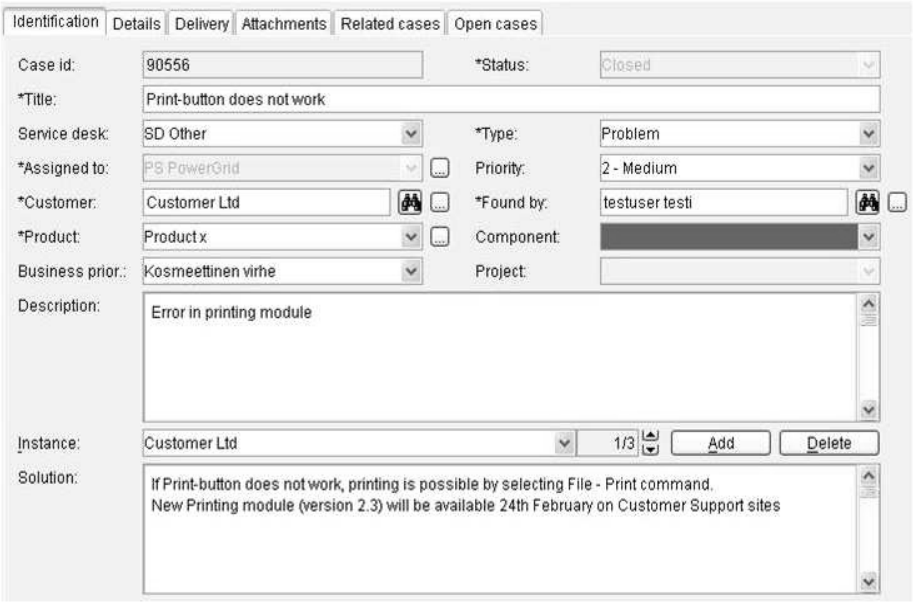
An important question is why should IT organizations update their traditional defect management models to a service oriented problem management approach. The first answer is that traditional defect management is too development-oriented. Defect management models do not give guidance on how service-related problems (for example, service availability problems) are monitored or handled. Second, defect management models focus only on defects. They do not tell how problems reported by customers are related to defects. Third, the service oriented approach emphasizes the importance of three support levels. Therefore, customers and users cannot get into a direct contact with programmers or developers in a problem situation and interrupt their work, which is a major problem in many IT organizations. When service desk and problem management can resolve most of the simple problems, developers have more time for software development. Finally, IT organizations world-wide are adopting service management processes that can be audited through ISO/IEC 20 0000 international standard [36]. IT Service Management Forum, a not-for-profit organisation dedicated to IT service management, has operation in over 40 countries [39]. Thousands of IT organizations use ITIL-based processes with the number increasing rapidly. Many of these organizations have started to require that their partners and subcontractors also use service management processes, including problem management.

4.2 Basic concepts of problem management

Service-oriented customer support is based on three customer support levels: the service desk (incident management), problem management, and third-line support (change management, product development). Support processes begin when a customer encounters a problem while using the software product, IT service or their documentation and contacts the service desk. This contact is usually done by phone, by e-mail or by a web-form. The term “incident” is used for this contact. An *incident* is "any event which is not part of the standard operation of a service and which causes an interruption to, or a reduction in the quality of the service" [73]. Incidents can be classified into two groups according to their type: failures and service requests. *Failures* are situations in which customers or users seem to have a clear problem, such as a hardware failure or a software failure. *Service requests* are requests from the user for support, delivery, information, advice or documentation, not being a failure in the IT infrastructure [75]. Typical service requests are those requests for new passwords or requests for more disk quota. In addition to incidents, customers and users send requests for change (RFCs) to the service desk. *RFCs* are “requests for a change to configuration items or to procedures within the infrastructure” [73].

The service desk is a function that performs an incident management process. It must be noted that problem management cannot work well without a well-defined service desk function that is responsible for managing incidents. Service desk workers are responsible for carrying out the following incident management activities: detecting and recording incidents, classifying incidents and giving initial support, investigating incidents and providing a diagnosis, resolving incidents and closing incidents. Figure 4.1 shows an example of the incident report that is created by the service desk engineer when a customer contacts the service desk.

If the service desk is not able to identify the root cause of an incident, the next step is to open a problem record. Root cause is “the underlying or original cause of



Case id:	90556	*Status:	Closed
*Title:	Print-button does not work		
Service desk:	SD Other	*Type:	Problem
*Assigned to:	PS PowerGrid	Priority:	2 - Medium
*Customer:	Customer Ltd	*Found by:	testuser testi
*Product:	Product x	Component:	
Business prior.:	Kosmeettinen virhe	Project:	
Description:	Error in printing module		
Instance:	Customer Ltd 1/3		
Solution:	If Print-button does not work, printing is possible by selecting File - Print command. New Printing module (version 2.3) will be available 24th February on Customer Support sites		

Figure 4.1: An incident report.

an incident or problem" and a root cause analysis is "an activity that identifies the root cause" [76]. In software engineering studies, the term 'root cause' is often related to defects [13]. It is typically used for IT Infrastructure failures [76]. ITIL recommends that problem records should be independent of incident records. This can be understood as follows:

- The service desk can collect several incidents before opening a problem record. If there is a major incident, the service desk can assign it to problem management.
- In a normal case, closing a problem also closes any related incidents.
- Problem investigation can also continue after an original incident has been closed.

Incidents and problems have a status attribute that reflects their position in their lifecycle (new, accepted, scheduled, assigned, work in progress, on hold resolved, closed) [73]. Thus, resolved and closed statuses are not synonyms. Incidents and problems are open until they reach a status "closed".

The second support level is problem management. *Problem management* is a process of managing problems and errors. Problem management aims to find the root cause of incidents and define a corrective solution or a process improvement [73]. The objectives of problem management include minimizing the impact of incidents and problems, solving problems and errors before customers send incidents, reducing the number of problems, and performing trend analyses and root cause analyses [73, 94, 66].

Paper III of this thesis introduces the conceptual model that captures the concepts (artifacts, processes and support levels) within problem management and the relationships between them. The problem management process includes a problem manager and problem support specialist roles [73]. A *problem manager* is responsible for developing the problem management process, developing and maintaining the applications of problem control and error control, reviewing both reactive and proactive problem management activities, and producing information for management.

A *problem support specialist* has both reactive and proactive responsibilities. Reactive responsibilities include identifying and investigating problems, generating RFCs, defining work-arounds, advising service desk of work-arounds, and handling major incidents by identifying the root cause. Proactive responsibilities include identifying problem trends, creating RFCs to prevent problems from happening again and preventing the replication of problems. Problem management has both reactive and proactive aspects (see Figure 4.2).

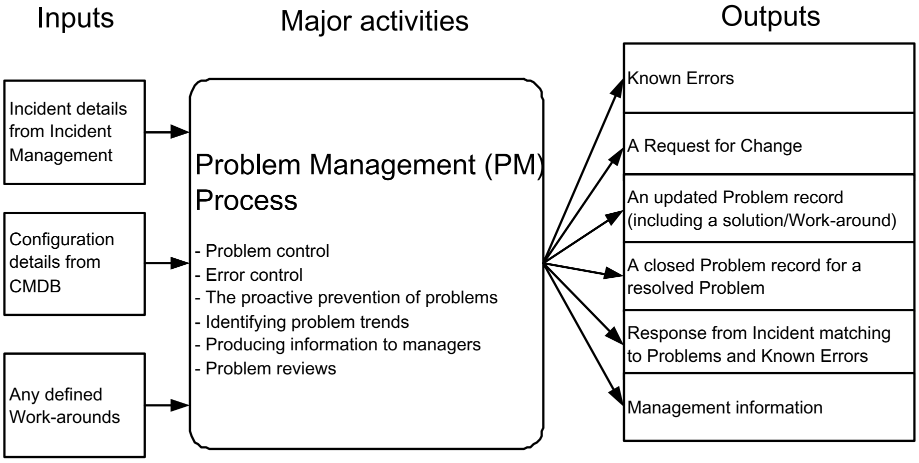

Inputs
Major activities
Outputs
Incident details from Incident Management
Configuration details from CMDB
Any defined Work-arounds
Problem Management (PM) Process
- Problem control
- Error control
- The proactive prevention of problems
- Identifying problem trends
- Producing information to managers
- Problem reviews
Known Errors
A Request for Change
An updated Problem record (including a solution/Work-around)
A closed Problem record for a resolved Problem
Response from Incident matching to Problems and Known Errors
Management information

Figure 4.2: Problem management activities.

4.3 Reactive and proactive problem management

Reactive problem management aims to resolve already reported incidents and problems whereas the goal of proactive problem management is to prevent incidents before they occur. Reactive problem management is further divided into problem control and error control activities where problem control is responsible for identifying the root cause of a problem and defining a temporary solution (work-around) for the problem. A *work-around* (for example, simple advice from the service desk) enables the user to continue using a service or a product while problem management is looking for a permanent solution to the problem. Figure 4.3 shows the phases of problem control and error control.

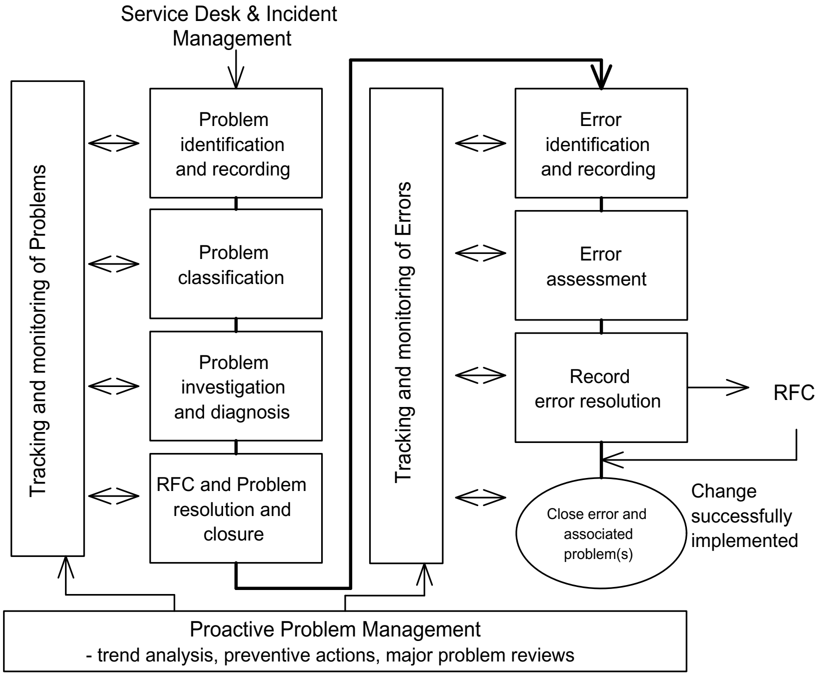
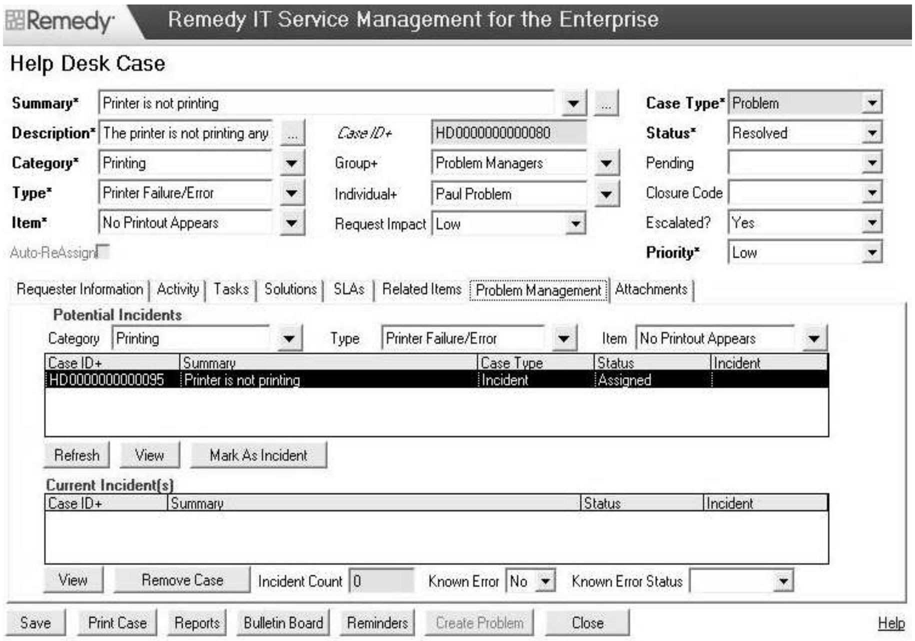

Service Desk & Incident Management
Tracking and monitoring of Problems
Problem identification and recording
Problem classification
Problem investigation and diagnosis
RFC and Problem resolution and closure
Tracking and monitoring of Errors
Error identification and recording
Error assessment
Record error resolution
RFC
Close error and associated problem(s)
Change successfully implemented
Proactive Problem Management
- trend analysis, preventive actions, major problem reviews

Figure 4.3: Phases of problem control and error control.

Problem control consists of four phases: problem identification and recording, problem classification, problem investigation and diagnosis, and problem resolution and closure [73]. In the *problem identification and recording* phase, a problem record is opened if the service desk is not able to identify the root cause of an incident or if there is a major incident or several similar incidents that need careful investigation. A problem record can include the following attributes: problem id, title, description, team code, impact, urgency, priority, a configuration item, status, category, entering date and time, the name of the person who entered the problem, reference to a service level agreement or target resolution time, resolution or work-around, and a comment field. The number and names of problem attributes vary between different IT organizations, between teams within the same organization, and between different problem management frameworks. As a result, it is impossible to compare the performance of problem management between different organizations and teams if the measurement framework is not the same.

Problem classification involves defining the category, impact, urgency, and priority for the problem (see Figure 4.4).


Remedy
Remedy IT Service Management for the Enterprise
Requester Information
Activity
Tasks
Solutions
SLAs
Related Items
Problem Management
Attachments
Refresh
View
Mark As Incident
View
Remove Case
Save
Print Case
Reports
Bulletin Board
Reminders
Create Problem
Close
Help

Help Desk Case

Summary*: Printer is not printing
Description*: The printer is not printing any
Category*: Printing
Type*: Printer Failure/Error
Item*: No Printout Appears
Auto-ReAssign ☐
Case ID+: HD0000000000080
Group+: Problem Managers
Individual+: Paul Problem
Request Impact: Low
Case Type*: Problem
Status*: Resolved
Pending:
Closure Code:
Escalated?: Yes
Priority*: Low

Potential Incidents

Category: Printing
Type: Printer Failure/Error
Item: No Printout Appears

Case ID+	Summary	Case Type	Status	Incident
HD0000000000095	Printer is not printing	Incident	Assigned	

Current Incident(s)

Case ID+	Summary	Status	Incident

Incident Count: 0
Known Error: No
Known Error Status:

Figure 4.4: A problem record [8].

Additionally, the problem status is defined (problem, known error, resolved, closed) [75]. Advanced IT service management tools provide a category tree that enables the categorization of a problem into domains and subdomains. The priority of the problem can be determined based on its impact and urgency [73].

Problem investigation and diagnosis aims to determine the root cause of the problem. This phase is perhaps the most difficult part of the problem management process [94]. In order to analyse and diagnose the problem, the problem management team might need the expertise of other teams, such as product development teams or communication with third-party service providers. However, customers and end-users appreciate *Single Point of Contact (SPOC) service* in which they can communicate with one IT service provider although several service providers might be involved in resolving the problem. The SPOC service consists of three contact levels between the IT provider and the customer [74]: a business manager level in which service level agreements are negotiated, a project manager level in which the RFCs from a customer are handled by the IT provider, an end-user level in which incidents are managed. The SPOC model usually requires substantial changes to existing support service agreements. Additionally, a SPOC service provider needs to charge problem handling costs from third-party service providers.

Problem investigation provides detailed information regarding which *configuration* *items*, components of an IT infrastructure, the problem is related to. Causes to problems can be classified into defects and non-software causes [49]. In addition to hardware and software errors, problems can be caused by human errors, documentation errors and procedural errors [75]. Some useful methods for this phase are Fault Tree Analysis (FTA) [69] and the Kepner and Tregoe analysis. *Fault Tree Analysis, FTA* is based on the fault tree in which the root node represents the most principal failure [69]. The *Kepner and Tregoe analysis* emphasizes five steps: defining the problem; describing the problem, its location and scope; determining the possible causes of the problem; testing the most probable cause; and verifying the true cause [54]. After the root cause of the problem and related configuration items have been identified, problem specialists check whether there is a workaround available for the problem. If not, a workaround is defined by the problem management team. A workaround might, for example, consist of an instruction for how to install a printer, an example of how to change file permissions, a reference to a user manual, or a security bug fix.

Problem resolution and closure: Once the root cause of the problem has been found and the workaround for the problem defined, the status of the problem is changed to a known error. According to ITIL release 3 [76] an error is “a design flaw or malfunction that causes a failure of one or more configuration items or IT services” or “a mistake made by a person or a faulty process that affects a CI or IT service” whereas IEEE defines an error as “ a human action that results in software containing a fault” [35] or “deviation from the expected results”. A *known error* is “an incident or problem for which a root cause is known and for which a temporary work-around or a permanent alternative has been identified” [73]. Thus, a known error is a problem that is known in the organization and it may be under resolution or not. If it is not under resolution, the reason might be that a problem is too expensive to resolve and there is a good workaround for it. In practice, the known-error record is almost the same as the problem record. Only the problem status is different. Finally, the problem can be closed and a Request for Change raised if necessary or the problem is sent to the error control team. In fact, the error control activity seems to be equivalent to the defect management process in software engineering. In addition to problem control activity, product development and testing units produce information on known but unresolved errors.

Error identification: If resolving the problem requires changing a configuration item or the cause of a problem is a defect in a component, the problem is escalated to the error control team that is responsible for identifying and recording known errors, classifying them and making an error assessment [73]. Whereas the problem control team is responsible for determining the root cause of the problem and identifying workarounds, error control aims to ensure that errors related to problems are resolved by application development teams through the change management process.

Error assessment: An assessment of how to resolve an error is performed by the problem management team. The known error will be resolved by sending a request for change to change management and application development. According to the ITIL, there are two sources for known errors [73]: live operations environment (customers and end users) and development environment (designers and developers). ITIL also recommends that errors in third-party products are recorded in the problem database. However, Paper V of this thesis shows that all IT service providers do not record third-party errors.

Error resolution and closure: In the error resolution phase, the resolution process for each known error is recorded in the problem database [73]. In the future, this information can be used by other problem management and incident management teams. After the change has been successfully implemented, the problem management team is able to close the known error record, as well as any related problem records and incident records. Support teams should ensure that the customer is satisfied with the error fix before the final closure of the incident [73]. The IT organization should agree with customers on how often bug fixes are delivered (see Paper II).

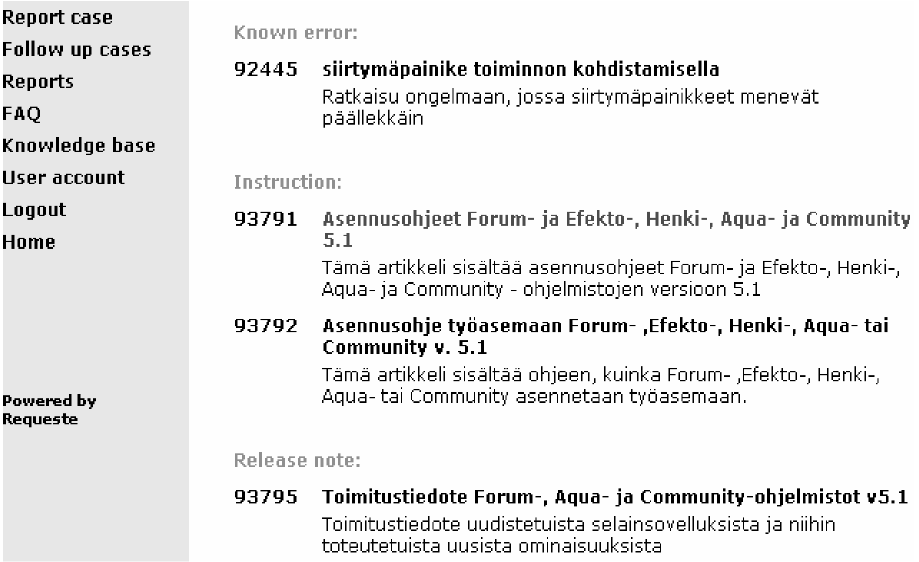
Instead of reactive problem management, IT organizations should focus on *proactive problem management*, such as preventing incidents and problems before they are found and reported by customers and users to the service desk. Proactive problem management can mean performing problem trend analyses, targeting preventive actions and carrying out major problem reviews. *Problem trend analyses* [15, 73] are usually based on incident and problem analysis reports. These analysis reports provide information on problem trends (when and where problems occur), weak components in the IT infrastructure, recurring problems, and training needs. *Preventive actions* include defining the training and education needs for both customers and developers, identifying process improvements (for example, improvements to the customer support web interface), ensuring that support teams follow the process descriptions of incident management and problem management, as well as providing feedback for software testers, developers, and designers [73]. Finally, *problem reviews* can be organized for major problems in order to define how to prevent similar problems from re-occurring.

In addition to the three above-mentioned proactive methods, we consider a knowledge base as offering a very useful tool for proactive problem management (as stated in Paper VI). A *knowledge base* can be defined as a database for knowledge management. IT organizations can use knowledge bases to collect, organize, and search the knowledge regarding IT products and services. In this thesis, the knowledge base provides its users with solutions to known problems in software products (see Figure 4.5).

A well-designed knowledge base helps the service desk to find solutions to problems quickly. It is an important step towards a self-service desk because customers and users are able to search problem solutions without the help of the service desk. Knowledge bases have been popular especially in university environments by providing, for example, technical support for university staff and students [40], and extending help desk services [27]. In paper VI, we presented results of a case study in which a knowledge base was implemented for the purposes of proactive problem management.

4.4 Problem management tools

There are numerous commercial (HP Unicenter, IBM Tivoli, Rational ClearQuest, BMC Remedy) and open source tools (e.g., RequestTracker, Bugzilla, JIRA) available for reporting problems and defects. Some of these tools are called IT service management tools that can be used in several IT service management processes in addition to problem management. Automated tools, such as web-based tools are useful for reporting incidents and problems since people are able to deliver problem reports whenever they want (also outside of office hours). Some users like to send reports



Known error:

92445 **siirtymäpainike toiminnon kohdistamisella**
Ratkaisu ongelmaan, jossa siirtymäpainikkeet menevät päällekkäin

Instruction:

93791 **Asennusohjeet Forum- ja Efekto-, Henki-, Aqua- ja Community 5.1**
Tämä artikkeli sisältää asennusohjeet Forum- ja Efekto-, Henki-, Aqua- ja Community - ohjelmistojen versioon 5.1

93792 **Asennusohje työasemaan Forum- ,Efekto-, Henki-, Aqua- tai Community v. 5.1**
Tämä artikkeli sisältää ohjeen, kuinka Forum- ,Efekto-, Henki-, Aqua- tai Community asennetaan työasemaan.

Release note:

93795 **Toimitustiedote Forum-, Aqua- ja Community-ohjelmistot v5.1**
Toimitustiedote uudistetuista selainsovelluksista ja niihin toteutetuista uusista ominaisuuksista

Figure 4.5: A web-based knowledge base for problem management.

anonymously. There are also tools that have been originally designed for knowledge management, content management and customer relationship management (Frequently Asked Question (FAQ) managers, knowledge-base applications) but are suitable for problem management purposes.

The problem management tool must fit to the organization's current needs. A cost-benefit analysis should be performed for a problem management tool as well as other software engineering tools including build-or-buy decisions. It is also important to test the problem management tool before buying it and to check whether it is possible to customize problem records, to define the lifecycle for a problem record, produce performance reports based on metrics and publish known errors to customers. Other important questions before selecting the tool include how many languages are supported by the tool, and whether the tool supports service level management, such as creating service level agreements.

4.5 Metrics and the process maturity

Defining metrics is an important part of any process improvement framework. Metrics are measurements that quantify results. Metrics provide a quantitative assessment of software quality [61]. Quality metrics can be categorized in several ways. In this thesis, we divide metrics into two groups (see Table 4.1): problem management metrics and defect management metrics.

The metrics can be also categorized into complexity metrics (the complexity of

Table 4.1: Metrics for defect management and problem management

Defect Management	Problem management (PM)
Total number of defects	Average number of open problems
Number of defects by development activity	Number of closed problems
Number of defects fixed in testing	Number of known errors
Number of reported bugs fixed	Number of problems that missed target resolution time
Number of fixes returned to developers	Number of change requests raised by PM
Manhours per major defect	Amount of time spent on investigations and diagnoses
Average age of uncorrected defects	Average time to close a problem
Mean time between failures	Total user downtime
Estimated number of remaining faults	Customer satisfaction on PM

the code), defect metrics (number of defects by type), product metrics (module size, number of lines of code), process metrics (inspection hours, defect correction time), cases per period metrics, and time-based performance metrics.

Problem management metrics are used to monitor the performance of the problem management process. Defect management metrics are also suitable for the error control phase of problem management though they are often categorized into application development metrics. Using metrics provides several benefits. In addition to performance measuring, metrics help organizations to identify weak areas and strengths in the processes, as well as to avoid dangers by identifying them in time [37]. However, an organization should carefully plan which metrics will be used, since measuring a process and producing reports can incur significant costs.

Another aspect of process measurement is measuring the maturity of service management processes. There are several maturity models that can be used to measure the maturity level of the problem management process: IT Service Capability Maturity Model [68], COBIT maturity model [15], and ITIL Service Management Self Assessment Model [38]. In the IT Service Capability Maturity Model [68] the service process maturity is measured by five maturity levels: initial, repeatable, defined, managed, and optimizing. The problem management process is mentioned on the third level and problem prevention on the fifth level. Incident management belongs to Level 2. COBIT provides a different type of maturity model in which each service management process is evaluated according to a 1-5 scale [15]. The ITIL Service Management Self Assessment Model [38] consists of a self-assessment questionnaire that tests whether there is complete conformance with ITIL. Organisations can use maturity models to set the goals for process improvement, for example, they might set a goal that the problem management process shall achieve maturity level 3 by the year 2009.

4.6 Connections to other service management processes

There should be a close interface between problem management and other service management processes, including incident management, change management, configuration management, service level management, and application development. In Paper VI of this thesis, we proposed that the knowledge management process would be beneficial for problem management.

Incident management: An IT organization can identify problems and known errors by analysing already reported and recorded incidents. Problem management receives incident details and temporary solutions (workarounds) from incident management [73, 54, 15]. Thus, the success of the problem management process depends very much on the performance of the incident management process. An IT organization should carefully analyze which datafields are really needed in incident and problem records. Recording incidents and problems is not effective if there are too many datafields. Additionally, the IT service management tool should have a function that enables relating similar incidents to one problem, as we stated in Paper V.

Change management is responsible for carrying out or controlling some of the error resolution activities: impact analysis, detailed assessment of the error, replacing the faulty component, and testing the change. The major task of change management is to process requests for change. A change advisory board (CAB) can be established to assist the change management team. CAB members typically review submitted RFCs and participate in CAB meetings where changes are authorized [73].

Configuration management aims to identify and record configuration items (CIs) and the relationships between CIs [75] in the IT infrastructure. Configuration items should be stored in a secure repository [92] referred to as a *Configuration Management Database (CMDB)* [73]. The version information of CIs must also be recorded in the CMDB. Configuration details should be available for problem management team as well as other service management processes. Paper V of this thesis proposed that in addition to problems in software configuration items, problem management should also record problems related to hardware configuration items.

Service level management (SLM): Service level management focuses on maintaining and improving IT Service quality. The SLM process manages *service level agreements* (SLAs) that are written agreements between service providers and customers [72]. SLAs define service targets and responsibilities of an IT service provider and a customer [55]. The problem management process should provide a close connection to service level management (see Paper V). Service level requirements (target resolution times for incidents and problems, for instance) should be defined for problem management. Monitoring services helps organizations to identify the weakest components in the service. An IT organization should ensure that service level requirements for handling problems and incidents in SLA match the SLA targets that are configured to the service desk tool. A service desk tool needs to monitor what has been agreed on between the IT provider and customers.

Application development and testing: Developers should be well aware of defensive programming methods [26]: exception handling, coding assertions (early warnings of upcoming failures), overengineering (building the system stronger than needed), and audit trails (a trail of past events that might help to identify the cause of the failure).

The responsibilities of application developers and testers should include participation in problem reviews and defect causal analysis meetings, since application development and testing produce considerable information that is useful for problem management purposes such as defect lists, inspection material and known error lists. Unfortunately, the connection between testing and problem management in ITIL is very poorly defined. Therefore, organizations implementing ITIL should not forget the importance of testing. It might be a good idea to maintain the traceability chain between incidents, problems, errors, changes and test cases, as was proposed in Paper V.

Knowledge Management (KM): In the European KM framework [14], *knowledge management* is defined as “the management of activities and processes for leveraging knowledge to enhance competitiveness through better use and creation of knowledge resources.” Similarly, Schultze and Leidner define knowledge management as "a process of generating, representing, storing, transferring, transforming, applying, embedding, and protecting organizational knowledge" [86]. Problem management can also benefit from the basic methods of knowledge management. Knowledge management helps IT organizations to share and reuse existing experiences of software development [20]. In problem management, organizations create and share knowledge concerning software products, IT services, technologies, customers, and development partners (see Paper VI). Hence, the concepts and methods of knowledge management and organizational learning can strongly support studies of software problem management. Knowledge management helps organizations to take a step towards a learning IT organization with knowledge-centered support. *A learning organization* is an organization skilled at creating, acquiring, and transferring knowledge, and at modifying its behavior to reflect new knowledge and insights [23]. The main principle of *knowledge-centered support* is to define the workflow for knowledge items, document problem solutions in a structured way, collect experiences on how people search for information and produce just-in-time solutions to users through a knowledge base [16].

4.7 Difficulties in service-oriented problem management

Although the service-oriented problem management resolves many shortcomings of traditional defect management models, it also results in new challenges and the following difficulties. The first challenge is related to the terminology. Service management frameworks use concepts (incidents, known errors, service level agreements) that have not been used in traditional software development models. Additionally, some concepts, such as problems and service requests, are used in different ways in service management and software engineering.

The second challenge is the lack of practical examples that would show the distinction between incidents, problems, service requests and change requests [62]. These concepts are quite unclearly explained in service management frameworks. Often, it is unclear how incidents are related to problems and known errors and which attributes are related to these concepts. It is important to remember that incidents never become problems and they are stored in separate records. Thus, ten incidents can lead to opening of one problem record. This challenge can be solved by providing support team members with enough examples, for example, which help-desk cases can be classified as service requests.

The third challenge arises from poorly defined connection between problem management and software testing and defect management. However, there are recently published studies that deal with this challenge. For example, Kajko-Mattsson has suggested a process model that defines front-end and back-end support processes [51]. Additionally, ITIL does not define how to integrate service management models and organization's existing software development models. Nevertheless, ITIL recommends its own application management process for software development.

The fourth important challenge is that service management frameworks do not define clearly enough how the knowledge base is related to service management processes, nor how service management roles are connected to the maintenance of the knowledge base. The difficulties and challenges regarding knowledge-base implementation are presented in Paper VI. These included tool-related problems (poor language support and tailoring problems) but also process-related (creating KB articles is difficult) and resource-related problems (a service desk needs more time for proactive problem management).

The fifth challenge is that service management frameworks do not tell how to handle incidents and problems that are sent to the wrong service desk. Often, an IT customer is dependent on several service providers and does not know which contact person is the correct one for a particular problem situation. Therefore, incidents and problems might pass through a long decision chain, thus lengthens the problem resolution time or even causing a situation in which customers may never find a solution to their problem. A Single Point of Contact service model might solve this challenge. The final challenge is that there is no comprehensive service support process diagram that would show the connections between different service support processes and their activities.

Most of these challenges are discussed in papers IV, V and VI. Solutions to the last two challenges are currently under work in our research group. These solutions shall include a description of a SPOC model in which a customer communicates with one service provider that assigns the case to other service providers if necessary. The first version of the service support process diagram was published in August 2007 [43]. It shows the activities of incident management, problem management and change management in a single process diagram.

Chapter 5

Summary of papers

In this chapter, the original papers are summarized and reviewed. This thesis includes six research papers all of which are related to managing problems and defects. These research papers are related to two research projects: the PlugIT and SOSE projects.

5.1 Relation of research papers and projects

PlugIT was a Finnish national research and development project on healthcare applications integration (2001-2004) financed by the National Technology Agency TEKES and private companies. PlugIT was implemented by a multidisciplinary research group from four departments at the University of Kuopio and the Savonia Polytechnic. The PlugIT research team included 20-30 researchers, developers, students and supervisors. PlugiT was carried out in four subprojects. The first research paper (UML-Based Testing: A Case Study) of this thesis was created in a TEHO subproject. The goal of TEHO was to develop and improve the methods and practices of software engineering, software quality assurance and testing. The research work regarding testing focused on component-based testing, automated testing, and early test case design.

SOSE (Service-Oriented Software Engineering) was a research project conducted at the University of Kuopio, Department of Computer Science (2004-2007). The project was financed by the National Technology Agency TEKES (European Regional Development Fund, ERDF) and five companies. The objective of the SOSE project was to develop methods of software engineering and software business with IT companies and their customers. The SOSE research team consisted of a project manager, two full-time researchers and part-time research assistants.

The SOSE project covered the following research areas: software business, business driven and service-oriented software engineering, service architectures, process integration, and application integration. The project was strongly focused on examining the service management processes of the IT Infrastructure Library (ITIL). In practice, the SOSE research team analyzed the existing support processes of IT organizations, identified challenges and bottlenecks in the customer support and problem management, and produced recommendations on how to resolve these challenges and transform processes into ITIL-based processes. The results of the SOSE project include international research publications on problem management and service architectures; masters theses on knowledge management, problem management, software project failures, and defect management; and reports on availability management and service management maturity models. Additionally, SOSE produced research material for designing and establishing a knowledge base for problem management purposes. Papers II-VI were created during the SOSE project.

Both projects dealt with software quality assurance techniques. While the PlugIT TEHO project focused more on traditional software quality assurance methods and defect detection techniques, such as testing and inspections, the SOSE project emphasized the service-oriented way of managing problems and defects. These projects played a very important role as a research platform for this thesis. The IT companies and IT customer organizations participating in these research projects provided us with a playground where we were able to test quality assurance methods in practice, and served as valuable information sources when we examined challenges in the support processes. Because we performed our case studies in cooperation with the companies that were members of our research project, our case selection method is quite close to a convenience sampling method. However, it is a very common way to select case organizations.

5.2 Summary of papers

For each article, the author of this thesis was the corresponding author with most contribution. In paper I, Section 4 (Testing components) was created in cooperation with Tanja Toroi. Regarding paper II, the co-authors reviewed the paper and provided valuable comments on the structure of the paper. In paper III, the co-author professor Anne Eerola provided good comments regarding research methods, the structure of the paper, and the content of the conceptual model. The first version of paper IV was published at the 29th Information System research seminar (IRIS), with the final version becoming published at the IASTED conference on Software Engineering. The second co-author, Aki Miettinen, helped in the literature search and in defining the problem management checklist. The third co-author represented the case organization and acted as a valuable informant for the study. The fifth paper was designed and implemented by the author of this thesis. The second co-author provided some comments concerning the process improvement ideas. Finally, the paper VI was written by the author of this thesis and the second co-author provided us with access to the knowledge base application and helped to perform the study in practice. The following sections briefly describe the research papers.

5.2.1 UML-Based Testing: A Case Study

Paper I introduces a UML-based test model that can be used for building test cases more systematically. The UML-based test model helps testers and developers to detect defects effectively. The model focuses on behavioral UML diagrams, such as activity diagrams, state diagrams, and use case diagrams. The main contribution of Paper I was to evaluate the UML test model in a case study with an IT department at the Kuopio university hospital. During the study, we noticed that use case diagrams and scenarios are very useful for testing purposes because they include descriptions of exceptional and alternative flows that often comprise the sources of defects. Similarly, activity diagrams show the different action flows that a tester must go through in testing. Additionally, transition coverage derived from a UML state diagram and a state transition table can be used for measuring test coverage. Our testing experiment for a healthcare application revealed three serious run-time errors and dozens of minor defects.

There are also challenges regarding the use of the UML-test model. UML diagrams are often produced in such way that they do not provide enough information to support software testing. A very common situation is that no specification documentation for the system or related UML diagrams are available, and end users and software testers do not have the time to draw UML diagrams in the testing phase. This paper was included in the thesis because it describes the starting point of my research. The study on UML based testing revealed how IT customers struggle with software problems and led us to explore defect management processes. Although our original plan was to explore defect classification in component-based software development in Paper II, we started to write an article about defect management process because our partner organizations were interested in process improvement.

5.2.2 Difficulties in Establishing a Defect Management Process: A Case Study

In Paper II, the main research question is what types of difficulties are encountered by organizations with regard to the defect management process. In the case study involving four case organizations, we explored the organizations' goals in defect management, defect management processes, and the problems related to the activities in defect management. The main contribution of Paper II was to help IT companies and IT customers to identify and avoid typical problems in defect management. Our findings showed that software providers had problems, for example, in defining good metrics for IT service problem management, creating a large amount of test data for testing, using load testing and performance testing tools, and in creating an organization-wide defect management process. IT customers complained, for example, that IT companies do not send defect resolution reports to customers, and software vendors deliver applications containing many bugs that should have been found during developer-side testing. While exploring various defect management frameworks for Paper II, we found an IT service management framework ITIL (IT Infrastructure Library). At this point, we decided to focus our research on ITIL for the following reasons: First, the ITIL model seemed to provide a clear advantage compared to other defect management models. The problem management process of ITIL combines both problem control (problem management) and error control (defect management) activities into one model while most models do not even recognize the gap between defect management and problem management. Second, the organizations within the SOSE research project were very interested in the ITIL-based process improvement.

5.2.3 A Conceptual Model of IT Service Problem Management

Paper III presents a conceptual model of problem management. The conceptual model was based on the problem management principles of the IT Infrastructure Library (ITIL). The main contribution of Paper III is to describe the concepts of problem management and their relationships as a conceptual model; explain the context, benefits, and limitations of the problem management model as a pattern; describe the attributes related to incidents, problems, and known errors; and validate the model with an example showing how to use it in software business. The conceptual model was validated during a problem management study with a case organization. The objective of the study was to examine how ITIL-based problem management concepts can be combined with the problem management process.

5.2.4 A Checklist for Evaluating the Software Problem Management Model

Paper IV continued our work regarding ITIL-based problem management. We recognized that instead of heavy IT service management standards IT organizations need practical guidelines and checklists to be able to implement ITIL-based processes. The main contribution of Paper IV was to provide a checklist of issues that are essential for implementing the problem management process.

The checklist consist of ten requirements: 1) establish a service desk, 2) define the lifecycle for incidents, 3) identify two dimensions of problem management: reactive and proactive, 4) establish a problem management repository and a knowledge base, 5) establish the problem control activity, 6) establish the error control activity, 7) define appropriate metrics for monitoring the problem management process, 8) monitor the problem management process against SLAs, 9) generate a request for change to the change management team, and 10) continuously improve the problem management process.

Managers who are responsible for establishing or improving the problem management process can use the checklist to evaluate whether their problem management processes meet ITIL requirements.

5.2.5 Improving the Software Problem Management Process

Paper V describes the challenges identified during the validation of our software problem management model. First, a list of the challenges and bottlenecks in the case organization’s problem management process were created. These challenges included, for example, the increasing number of incidents and open problems, poorly defined connections between problem management and testing, and difficulties in combining service management concepts with existing business concepts. Second, we defined the tasks or activities required to solve these challenges. For example, service-desk and product-support teams need clear instructions on how to handle incidents and service requests, and the problem records needs a category for the errors of third-party service provider’s products. Finally, we analyzed why it is important to solve these challenges.

The main contribution of Paper V was to help IT organizations to identify the challenges and problems associated with ITIL-based problem management. This paper inspired us to examine proactive problem management methods, such as the introduction of a knowledge base for problem management.

5.2.6 Challenges in Implementing a Knowledge Base for Software Problem Management

Paper VI focuses on proactive problem management. In this paper, we presented the results of a case study that identified the challenges in building a knowledge-base system for software problem management. In our case, the knowledge base (KB) provided its users with solutions to known problems. As the main contribution of Paper VI, we described the implementation process of the knowledge base in the context of software problem management, identified the challenges arising from the implementation process, described the datafields related to the knowledge-base articles, and validated the study with an example showing how a knowledge base could be used for energy network management software.

The DISER framework was used to document the implementation process of the knowledge base. DISER (Design and Implementation of SE repositories) [11] describes the process, techniques and tools for the life-cycle of an experience factory. Identified challenges included, for example, difficulties in creating public KB articles because service desk cases contain much customer-specific information, difficulties in establishing a structure for the KB, and poor language support of the tool used for the KB articles.

5.3 Summary of the results

The main contributions of this thesis were to evaluate the UML-based test model as a defect detection technique (Paper I), compile a documented list of challenges and difficulties regarding defect management (Paper II), develop a systematic approach for service-oriented problem management (Paper III), and identify a list of difficulties concerning service-oriented problem management and recommendations together with guidelines how service-oriented problem management should be performed in practice (Papers IV-VI). Next, a short summary of the results is given.

In the first phase, we provided an answer to the first research question: which methods can be used to detect defects. Traditional software quality assurance methods such as reviews, inspections, pair programming, testing approaches (module testing, integration testing, system testing and acceptance testing), risk management techniques, and defect management methods provide developer-side information on defects. UML-based testing is also a promising defect-detection technique with certain restrictions. Additionally, customer support (help desks and service desks) collects very useful information for defect management.

The second research question was: What types of challenges are related to the process of managing defects. The following challenges regarding defect management were identified during the research process. Based on the case studies (Papers I, II) we argue that IT providers have the following difficulties in managing defects: Project teams use non-standardized problem management and defect management methods; there is no service desk that would be responsible for collecting all service requests; it is difficult to find a good frequency for customer bug fixes; and IT providers have limited resources for fixing defects. The first two issues are consistent with the results of the Quality Assurance Institute [81]. IT organizations should establish organization-wide processes for managing defects. A well-defined, documented defect management process enables organizations to repeat the process from day to day and to improve the methods for managing defects. It was not surprising that there were limited resources for fixing defects and that this was a challenge. However, the biggest surprise was that there are still software development organizations that have not implemented a service desk. In these organizations, developers and coders are usually responsible for recording defects and problems reported by users, even the most simple problems.

In order to gain a richer view on the challenges hindering defect management, we studied the types of experiences encountered by IT customers with regard to defect management. Our results showed that the problems experienced by IT customers were different from those of IT providers. For example, IT customers do not receive confirmation on whether their support request has been accepted, nor do they receive any information concerning the progress of defect handling. Additionally, IT customers do not receive enough feedback on how problems were resolved or could be avoided in the future. Customers also considered such comments from the service desk as “This is not a bug” or “This is not our bug” to be a big problem. Moreover, users reported that they need training regarding defect reporting tools. Finally, customers complained that software vendors deliver applications that include bugs that should have been found during the developer-side testing.

The third research question was: what is service-oriented problem management and which concepts are related to it? We created a conceptual model and validated it in cooperation with an IT service provider and an IT customer. The validation started with an observation phase where we observed working practices of a service desk worker and participated in the customer support team’s meetings. We analyzed whether the ITIL-based concepts of our conceptual model were visible in the customer support processes of the IT provider. Additionally, we analyzed relationships between concepts and the rationale of concepts in the case organization. We reported our findings to customer support teams during an ITIL training day. The following list of recommendations for a service-oriented problem management model is based on both a literature review (defect management and problem management literature) and empirical findings (interviews with IT providers and IT customers, and our observations) during the research process.

1. The service-oriented problem management model should include a clear definition of the goals, inputs, outputs, activities, roles, and metrics for problem management. A customer user role needs to be divided into a main user role and an end-user role. A well-defined, documented process can be measured and improved, in contrast to ad hoc processes.

2. The model should be based on the three-level service support model in which the first level consists of the service desk that manages the incident management

process, the second level corresponds to problem management, and the third level includes, for example, change management and application development processes. Simple incidents and problems are resolved by the service desk and problem management.

3. The model should provide a description of the problem management concepts (incidents, problems, known errors and requests for change) and attributes related to these concepts. Clear rules are needed concerning how to classify incidents into failures and service requests, as well as how to relate incidents into problems, known errors, and RFCs. Service-desk workers do not have extra time to think about how to categorize or classify incidents and service requests such as service availability problems and support tool-related problems

4. The model should provide a process diagram that shows the connections between problem management and other processes such as service desk & incident management, change management, service level management and application development. Well-established connections between processes decrease the number of information gaps within the organization and improve the total performance of service management. Additionally, other processes (application development and testing, for instance) produce considerable information that is useful for problem management purposes such as defect lists, inspection material and known error lists.

5. The model should define both proactive and reactive activities. A greater focus should be placed on proactive problem management, including problem reviews, trend analyses, and preventive actions. All the defect prevention and problem prevention techniques used in the organization should be documented and the staff trained to use them. The results of problem reviews and trend analyses help in identifying weak areas in products and services.

6. The model should define how problems and errors in the products and services of third-party service providers are monitored and handled. The service quality of the service provider often depends on the quality provided by third-party service providers. Thus, it is important to describe how incidents move forward in the network of service providers.

7. The model should combine problem management with defect management into a seamless process in order to close the gap between service-desk activities and software-development activities. A common tool for problem management and defect management is a good improvement idea.

8. The model should define how a knowledge base can be implemented and used as part of the problem management process. The knowledge base can be used as a proactive tool to decrease the number of incidents.

The last research question was: What kind of difficulties are associated with the introduction of service-oriented problem management. These difficulties were identified in evaluating service-oriented problem management methods in practice. For example, the following difficulties were discussed in research papers IV, V, and VI: there are no proactive problem management methods defined in the IT organization (IV, VI); combining ITIL concepts with existing business concepts is difficult (V); the activies and roles of IT support teams are poorly documented (IV); there is no problem category for third-party errors (V); the problem database often includes duplicate records or incomplete records (V); problem management is poorly linked to other processes, such as service level management, testing and application development (IV); and finally, it is difficult to integrate the knowledge base into service management processes (VI). Moreover, knowledge base implementation projects may include resource-related, process-related and tool-related difficulties (VI).

Perhaps the most important challenge regarding service-oriented problem management is that IT organizations solely focus on reactive problem management (solving already reported problems), thus ignoring the benefits of proactive work. The second significant problem is the different terminology between service management and traditional software engineering. Service management concepts need to be mapped into the organization's existing concepts and people need to be trained to use these concepts.

Additionally, a major problem is the gap between service desk and product development. IT organizations often consider these two processes as strictly separated activies and use different tools for these processes. This gap results in a communication barrier. Software developers do not receive all information on users' problems reported to the service desk, and the service desk does not receive all information on defects found by testers and developers.

The contributions of this thesis are useful for those problem managers, service managers and quality managers who are responsible for improving software quality assurance methods or are planning to implement a problem management process in the near future. Our research focused on identifying the difficulties and challenges regarding problem management. Some of difficulties we identified can be solved by changing a problem management tool or performing some configuration work (e.g., defining a category tree for problems and defects, sending an automated incident confirmation report to customers) whereas other process-related difficulties (e.g., ensuring that each team uses standardized methods for tracking problems and defects) are more difficult to resolve. In the future, we shall focus our research on providing solutions to these challenges and on establishing a well-documented, service-oriented problem management model.

Chapter 6

Conclusions

IT organizations are constantly looking for new ways to improve the quality of IT services and to decrease the costs of managing problems, while at the same time they are struggling with an increasing number of incident and problem reports. There is a clear need for a service-oriented problem management process that combines two previously separate activities: problem management and defect management. A large number of open defects and problems can cause significant financial losses for both IT companies and IT customers.

6.1 Contributions of the thesis

This thesis has made the following contributions: In the first phase, we found answers to the first research question: which methods can be used to detect defects? Various software quality assurance methods were identified and analyzed. We introduced a UML-based test model for creating test cases and identifying defects in the early phases of the software life-cycle. Several defects and problem areas in a health care application were detected using a UML-test model in a case study.

In the second phase, we identified the difficulties arising in organizations' defect management processes. Several difficulties were found: project teams use non-standardized problem management and defect management methods, and IT providers have limited resources for fixing defects and problems. IT customers also reported interesting challenges. For example, it is difficult to gain enough feedback from IT providers on how a problem was resolved or can be avoided in the future. This phase provided an answer to the second research question: What types of difficulties are encountered by IT organizations with regard to defect management?

Third phase helped IT companies and IT customers to identify the concepts and their relationships within service-oriented problem management. The service-oriented problem management model provides IT organizations with a solution on how to manage the software problems reported by IT customers, end-users or other stakeholders, as well as how to transform problems into errors (defects). IT organizations with well-defined problem management models will have fewer difficulties in managing problems than those without well-defined process models. The third phase provided an answer to the third research question: what is service oriented problem management and which concepts are related to it?

The fourth phase answered to the last research question: what types of difficulties do IT organizations experience in introducing service-oriented problem management? Recommendations concerning which issues are important in establishing a problem management process were produced in the form of a problem management checklist. Service-oriented problem management is needed because defect management models do not provide enough guidance on how service-related problems and failures can be monitored or handled. The service desk and problem management are able to resolve a large number of the simple problems that are traditionally solved by busy software developers. Service-oriented problem management ensures that developers have more time to focus on developing software.

Many IT service providers have developed their own models based on ITIL. Because reading ITIL-based books and standards requires considerable time and effort from organizations, we have extracted information on ITIL processes to a conceptual model, providing practical examples and a checklist for problem management. These examples and checklists have increased the value of the conceptual model and provided research partners with a faster learning process regarding problem management concepts. We have received positive feedback from IT organizations and IT customers within the SOSE project regarding our service-oriented problem management model, the problem management checklist and the difficulties identified in support processes. The results of the fourth phase are especially interesting and important because challenges and difficulties affecting problem management and knowledge-base implementation projects have previously been given very little consideration.

The main contributions of this thesis are 1) an evaluation of a UML-based test model as a defect detection technique, 2) a documented list of challenges and difficulties regarding defect management, 3) a systematic approach to service-oriented problem management and 4) a list of difficulties concerning service-oriented problem management, as well as recommendations and guidelines describing how service-oriented problem management can be achieved in practice. Based on the results of this thesis, we are able to present the following concluding remarks:

1. The UML-based test model effectively supports such traditional quality assurance methods as black-box testing, reviews and inspections in the identification of defects and problems. However, UML-based testing requires that testers have good UML skills and experience with test case design.

2. IT organizations have several difficulties in managing defects. One major difficulty is a lack of well-defined defect management model. These difficulties seem to have an influence on customer satisfaction with support services.

3. Service-oriented problem management provides a systematic approach for managing problems and defects. It creates a bridge between customer-oriented problem management and development-oriented defect management activities. IT organizations should focus on proactive problem management, such as using problem reviews and knowledge bases.

4. IT organizations have tool-related, process-related and resource-related difficulties in introducing service-oriented problem management.

There are some limitations to this study. The first limitation is related to the generalizability of the results. Instead of providing generalizeable conclusions we have focused on extending the theory of problem management. The second limitation is related to the small number of case organizations. Research data for this thesis has been collected from five case organizations. The reported results regarding the difficulties in problem management are based on the work with two case organizations. Service-oriented problem management concepts were tested in cooperation between one service provider company and one IT customer company. We have already started to study the challenges of service-oriented problem management also with yet other case organizations in order to improve our problem management model [43].

6.2 Future work

In this thesis, the primary focus was to identify difficulties in managing defects and problems rather than developing a well-documented problem management model. In future studies we intend to use the results of this thesis to build a well-defined, documented problem management model that meets the requirements of ISO/IEC 20 000 standards, solves most of the difficulties identified by this thesis, includes practical examples and process diagrams that would make implementing the process easier. Future work could attempt to derive a more complete list of problem management challenges and examine which proactive problem management methods are used by IT organizations. Finally, more research efforts are needed to examine service management maturity models from the IT customer's viewpoint.

Bibliography

[1] Agile Data. Agile data website. http://www.agiledata.org/, 7 2007.

[2] Jarmo J. Ahonen, Tuukka Junttila, and Markku Sakkinen. Impacts of the organizational model on testing: Three industrial cases. *Empirical Softw. Engg.*, 9(4):275–296, 2004.

[3] Alain April, Jane Huffman Hayes, Alain Abran, and Reiner Dumke. Software maintenance maturity model (smmm): the software maintenance process model: Research articles. *J. Softw. Maint. Evol.*, 17(3):197–223, 2005.

[4] Izak Benbasat, David K. Goldstein, and Melissa Mead. The case research strategy in studies of information systems. *MIS Q.*, 11(3):369–386, 1987.

[5] Keith H. Bennett and Vaclav T. Rajlich. Software maintenance and evolution: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 73–87, New York, NY, USA, 2000. ACM Press.

[6] Richard Berntsson-Svensson and Aybueke Aurum. Successful software project and products: An empirical investigation. In *ISESE '06: Proceedings of the 2006 ACM/IEEE international symposium on International symposium on empirical software engineering*, pages 144–153, New York, NY, USA, 2006. ACM Press.

[7] Robert Binder. *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Addison-Wesley, 2000.

[8] BMC Software Inc. Bmc remedy it service management. Marketing material, 8 2007.

[9] Barry Boehm and Victor R. Basili. Software defect reduction top 10 list. *Computer*, 34(1):135–137, 2001.

[10] Barry Boehm and Frank Belz. Experiences with the spiral model as a process model generator. In *Proceedings of the 5th international software process workshop on Experience with software process models*, pages 43–45, Los Alamitos, CA, USA, 1990. IEEE Computer Society Press.

[11] Frank Bomarius and Raimund Feldmann. Get your experience factory ready for the next decade: Ten years after "how to build and run one". Profes 2006 Tutorial, Amsterdam, Netherlands, 6 2006.

[12] M. Bush. Improving software quality: the use of formal inspections at the jpl. In *ICSE '90: Proceedings of the 12th international conference on Software engineering*, pages 196–199, Los Alamitos, CA, USA, 1990. IEEE Computer Society Press.

[13] David N. Card. Learning from our mistakes with defect causal analysis. *IEEE Software*, 15(1):56–63, January/February 1998.

[14] CEN Workshop Agreement CWA 14924-1. *European Guide to Good Practice in Knowledge Management, Part 1*. European Committee for Standardization, 2004.

[15] COBIT 4.0. *Control Objectives for Information and related Technology: COBIT 4.0*. IT Governance Institute, 2005.

[16] Consortium for Service Innovation. The kcs sm operational model (knowledge-centered support). http://www.serviceinnovation.org/ourwork/kcs.php, 2007.

[17] James Coplien. *Software Patterns*. Bell Laboratories, the Hillside Group, 2000.

[18] Jeanne Lee Cunningham. The problem is problems: problem tracking, resolution and record keeping in a large university environment. In *SIGUCCS '92: Proceedings of the 20th annual ACM SIGUCCS conference on User services*, pages 77–80, New York, NY, USA, 1992. ACM Press.

[19] Kathleen Eisenhardt. Building theories from case study research. *Academy of Management Review*, 14:532–550, 1989.

[20] Peter Feher and Andras Gabor. The role of knowledge management supporters in software development companies. *Software Process Improvement and Practice*, 11(3):251–260, June 2006.

[21] William Florac. Software quality measurement a framework for counting problems and defects. Technical Report CMU/SEI-92-TR-22, 1992.

[22] Michael Fredericks. Using defect tracking and analysis to improve software quality. US Air Force Research Laboratory Report SP0700-98-D-4000, 1998.

[23] David Garvin. Building a learning organization. *Harward Business Review*, pages 78–91, July-August 1993.

[24] John Geralds. Software bugs cost billions. IT Week, June 2002.

[25] Tom Gilb and Dorothy Graham. *Software Inspection*. Addison-Wesley, 1993.

[26] Robert Glass. *Building quality software*. Englewood Cliffs: Prentice Hall, 1992.

[27] Jay Graham and Brian Hart. Knowledge integration with a 24-hour help desk. In *SIGUCCS '00: Proceedings of the 28th annual ACM SIGUCCS conference on User services*, pages 92–95, New York, NY, USA, 2000. ACM Press.

[28] Volker Gruhn and Juri Urbainczyk. Software process modeling and enactment: an experience report related to problem tracking in an industrial project. In *ICSE '98: Proceedings of the 20th international conference on Software engineering*, pages 13–21, Washington, DC, USA, 1998. IEEE Computer Society.

[29] Andreas Hanemann, Martin Sailer, and David Schmitz. Assured service quality by improved fault management. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 183–192, New York, NY, USA, 2004. ACM Press.

[30] Jean Hartmann, Claudio Imoberdorf, and Michael Meisinger. Uml-based integration testing. In *ISSTA '00: Proceedings of the 2000 ACM SIGSOFT international symposium on Software testing and analysis*, pages 60–70, New York, NY, USA, 2000. ACM Press.

[31] Eva Holmquist, Malin Jernrup, Susanne Lieberg, and Tore Qvist. Experiences of testdriven development. In *EUROSTAR 2001: Proceedings of the 9th European International Conference on Software Testing Analysis& Review*, Galway, Ireland, 2001. EuroSTAR Conferences.

[32] John Horch. *Practical guide to software quality management*. Boston, MA : Artech House, 2003.

[33] IEEE Standard 1044-1993. *IEEE Standard Classification for Software Anomalies*. IEEE, 1994.

[34] IEEE Standard 729-1983 1. *IEEE Standard Glossary of Software Engineering Terminology*. IEEE, 1983.

[35] IEEE Standard 982.1-1988. *IEEE Standard Dictionary of Measures to Produce Reliable Software*. IEEE, 1988.

[36] ISO/IEC. *ISO/IEC 20000 A Pocket Guide*. Van Haren Publishing, 2006.

[37] IT Service Management Forum. *Metrics for IT Service Management*. Van Haren Publishing, 2006.

[38] IT Service Management Forum. Itil service management self assessment. http://www.itsmf.com/bestpractice/selfassessment.asp, 5 2007.

[39] IT Service Management Forum. Itsmf website. http://www.itsmf.org/, 5 2007.

[40] Anne Jackson, Gregory Lyon, and Janet Eaton. Documentation meets a knowledge base: blurring the distinction between writing and consulting (a case study). In *SIGDOC 98: Proceedings of the 16th annual international conference on Computer documentation*, pages 5–13, New York, NY, USA, 1998. ACM Press.

[41] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1999.

[42] Pankaj Jalote. *CMM in Practice, Processes for Executing Software Projects at Infosys*. Addison-Wesley, 2000.

[43] Marko Jäntti and Niko Pylkkänen. The role of problem management in customer support. In *IRIS 30. Information Systems Research Seminar in Scandinavia.Proceedings of the 30th IRIS*, Tampere, Finland, 2007.

[44] Pertti Järvinen. *On research methods*. Opinpajan Kirja, 2004.

[45] Pertti Järvinen and Annikki Järvinen. *Tutkimustyön metodeista*. Opinpaja Oy, 1995.

[46] Mira Kajko-Mattsson. Corrective maintenance maturity model: Problem management. In *ICSM '02: Proceedings of the International Conference on Software Maintenance (ICSM'02)*, page 486, Washington, DC, USA, 2002. IEEE Computer Society.

[47] Mira Kajko-Mattsson and Therez Björnsson. Outlining developers' testing process model. In *Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2007)*. IEEE Computer Society, 2007.

[48] Mira Kajko-Mattsson, Stefan Forssander, and Ulf Olsson. Corrective maintenance maturity model (cm3): maintainer's education and training. In *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*, pages 610–619, Washington, DC, USA, 2001. IEEE Computer Society.

[49] Kajko-Mattsson, M. A conceptual model of software maintenance. In *ICSE '98: Proceedings of the 20th international conference on Software engineering*, pages 422–425, Washington, DC, USA, 1998. IEEE Computer Society.

[50] Kajko-Mattsson, M. Problem management maturity within corrective maintenance. *Journal of Software Maintenance*, 14(3):197–227, 2002.

[51] Kajko-Mattsson, M. Maturity status within front-end support organizations. In *Proceedings of 29th International Conference on Software Engineering (ICSE'07)*. IEEE, 2007.

[52] Supaporn Kansomkeat and Wanchai Rivepiboon. Automated-generating test case using uml statechart diagrams. In *SAICSIT '03: Proceedings of the 2003 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology*, pages 296–300, Republic of South Africa, 2003. South African Institute for Computer Scientists and Information Technologists.

[53] Sharon Kanter and Gary Jones. No problem problem tracking. In *SIGUCCS '93: Proceedings of the 21st annual ACM SIGUCCS conference on User services*, pages 351–356, New York, NY, USA, 1993. ACM Press.

[54] Victor Kapella. A framework for incident and problem management. International Network Services whitepaper, 2003.

[55] Harri Karhunen and Marko Jäntti. Service-oriented software engineering (sose) framework. In *Proceedings of the Second IEEE Conference on Service Systems and Service Management, ICSSSM05*, June 2005.

[56] Diane Kelly and Terry Shepard. A case study in the use of defect classification in inspections. In *CASCON '01: Proceedings of the 2001 conference of the Centre for Advanced Studies on Collaborative research*, page 7. IBM Press, 2001.

[57] Jyrki Kontio. *Software Engineering Risk Management: A Method, Improvement Framework, and Empirical Evaluation*. PhD thesis, Helsinki University of Technology, 2001.

[58] Philippe Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley, 2001.

[59] Marek Leszak, Dewayne E. Perry, and Dieter Stoll. A case study in root cause defect analysis. In *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, pages 428–437, New York, NY, USA, 2000. ACM Press.

[60] Bennett P. Lientz and E. Burton Swanson. *Software Maintenance Management*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1980.

[61] Vladimir Marik, Lubos Kral, and Radek Marik. Software testing & diagnostics: Theory & practice. In *SOFSEM '00: Proceedings of the 27th Conference on Current Trends in Theory and Practice of Informatics*, pages 88–114, London, UK, 2000. Springer-Verlag.

[62] Hank Marquis. How to classify incidents. Newsletter in http://www.itsmsolutions.com/newsletters/ DITYvol2iss50.htm, 10 2007.

[63] Materna Oy. Itsm survey. http://www.materna.de/FI/Home/, 11 2005.

[64] R. G. Mays, C. L. Jones, G. J. Holloway, and D. P. Studinski. Experiences with defect prevention. *IBM Syst. J.*, 29(1):4–32, 1990.

[65] John D. McGregor and Timothy D. Korson. Integrated object-oriented testing and development processes. *Commun. ACM*, 37(9):59–77, 1994.

[66] Microsoft. Microsoft operations framework. http://www.microsoft.com/ technet/solutionaccelerators/cits/mo/mof/default.mspx, 5 2007.

[67] James Miller, Fraser Macdonald, and John Ferguson. Assisting management decisions in the software inspection process. *Inf. Tech. and Management*, 3(1-2):67–83, 2002.

[68] Frank Niessinka, Viktor Clerca, Ton Tijdinka, and Hans van Vlietb. The it service capability maturity model version 1.0. CIBIT Consultants&Vrije Universiteit, 2005.

[69] Atsushi Noda, Tsuneo Nakanishi, and Teruaki Kitasuka. Introducing fault tree analysis into product-line software engineering for exception handling feature exploitation. In *Proceedings of the 25th IASTED International Multi-Conference Software Engineering*, pages 229–234, Innsbruck, Austria, 2007.

[70] Office of Government Commerce. *ITIL Application Management*. The Stationary Office, UK, 2002.

[71] Office of Government Commerce. *ITIL ICT Infrastructure Management*. The Stationary Office, UK, 2002.

[72] Office of Government Commerce. *ITIL Service Delivery*. The Stationary Office, UK, 2002.

[73] Office of Government Commerce. *ITIL Service Support*. The Stationary Office, UK, 2002.

[74] Office of Government Commerce. *ITIL Business Perspective: The IS View on Delivering Services to the Business*. The Stationary Office, UK, 2004.

[75] Office of Government Commerce. *Introduction to ITIL*. The Stationary Office, UK, 2005.

[76] Office of Government Commerce. *ITIL Service Operation*. The Stationary Office, UK, 2007.

[77] Office of Government Commerce. *ITIL Service Transition*. The Stationary Office, UK, 2007.

[78] Tauhida Parveen, Scott Tilley, and George Gonzalez. A case study in test management. In *ACM-SE 45: Proceedings of the 45th annual southeast regional conference*, pages 82–87, New York, NY, USA, 2007. ACM Press.

[79] Mauro Pezz and Michal Young. Testing object oriented software. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 739–740, Washington, DC, USA, 2004. IEEE Computer Society.

[80] Monvorath Phongpaibul and Barry Boehm. An empirical comparison between pair development and software inspection in thailand. In *ISESE '06: Proceedings of the 2006 ACM/IEEE international symposium on International symposium on empirical software engineering*, pages 85–94, New York, NY, USA, 2006. ACM Press.

[81] Quality Assurance Institute. A software defect management process. Research Report number 8, 1995.

[82] Walker Royce. *Software Project Management: A Unified Framework*. Addison-Wesley, 1998.

[83] Ahmed Salem and Lalitha Balasubramanian. Utilizing uml use cases for testing requirements. In *Software Engineering Research and Practice*, pages 269–275, 2004.

[84] Robert J. Sandusky and Les Gasser. Negotiation and the coordination of information and activity in distributed software problem management. In *GROUP '05: Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work*, pages 187–196, New York, NY, USA, 2005. ACM Press.

[85] Annie Saunders. Online solutions: looking to the future of knowledgebase management. In *SIGUCCS '04: Proceedings of the 32nd annual ACM SIGUCCS conference on User services*, pages 194–197, New York, NY, USA, 2004. ACM Press.

[86] Ulrike Schultze and Dorothy Leidner. Studying knowledge management in information systems research: discourses and theoretical assumptions. *MIS Quarterly*, 26(3):213–242, Sep 2002.

[87] Avik Sinha and Carol Smidts. Hottest: A model-based test design technique for enhanced testing of domain-specific applications. *ACM Trans. Softw. Eng. Methodol.*, 15(3):242–278, 2006.

[88] Thomas Thelin. Team-based fault content estimation in the software inspection process. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 263–272, Washington, DC, USA, 2004. IEEE Computer Society.

[89] Jos J. M. Trienekens, Jacques J. Bouman, and Mark Van Der Zwan. Specification of service level agreements: Problems, principles and practices. *Software Quality Control*, 12(1):43–57, 2004.

[90] Gursimran S. Walia, Jeffrey Carver, and Thomas Philip. Requirement error abstraction and classification: an empirical study. In *ISESE '06: Proceedings of the 2006 ACM/IEEE international symposium on International symposium on empirical software engineering*, pages 336–345, New York, NY, USA, 2006. ACM Press.

[91] Ernest Wallmueller. *Software quality assurance: A practical approach*. Prentice Hall International, 1994.

[92] Brian White. *Software Configuration Management Strategies and Rational Clear Case*. Addison-Wesley, 2000.

[93] Robert Yin. *Case Study Research : Design and Methods*. Beverly Hills, CA: Sage Publishing, 1994.

[94] Jian Zhen. It needs help finding root causes. *Computerworld*, 39(33):26, 2005.

Kuopio University Publications H. Business and Information technology

H 1. Pasanen, Mika. In Search of Factors Affecting SME Performance: The Case of Eastern Finland.
2003. 338 p. Acad. Diss.

H 2. Leinonen, Paula. Automation of document structure transformations.
2004. 68 p. Acad. Diss.

H 3. Kaikkonen, Virpi. Essays on the entrepreneurial process in rural micro firms.
2005. 130 p. Acad. Diss.

H 4. Honkanen, Risto. Towards Optical Communication in Parallel Computing.
2006. 80 p. Acad. Diss.

H 5. Laukkanen, Tommi. Consumer Value Drivers in Electronic Banking.
2006. 115 p. Acad. Diss.

H 6. Mykkänen, Juha. Specification of reusable integration solutions in health information systems.
2006. 88 p. Acad. Diss.

H 7. Huovinen, Jari. Tapayrittäjyys – tilannetekijät toiminnan taustalla ja yrittäjäkokemuksen merkitys yritystoiminnassa.
2007. 277 p. Acad. Diss.

H 8. Päivinen, Niina. Scale-free Clustering: A Quest for the Hidden Knowledge.
2007. 57 p. Acad. Diss.

H 9. Koponen, Timo. Evaluation of maintenance processes in open source software projects through defect and version management systems.
2007. 92 p. Acad. Diss.

H 10. Hassinen, Marko. Studies in mobile security.
2007. 58 p. Acad. Diss.