Process Improvement through Software Operation Knowledge

If the SOK Fits, Wear It!

Henk van der Schuur

SIKS Dissertation Series No. 2011-43
The research reported in this dissertation has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

ISBN 978-90-393-5668-5

Cover design: COUP

Process Improvement through Software Operation Knowledge

If the SOK Fits, Wear It!

Procesverbetering door middel van softwarefunctioneringskennis

Wie de SOK past, trekke hem aan!

(met een samenvatting in het Nederlands)

Proefschrift

ter verkrijging van de graad van doctor aan de Universiteit Utrecht op gezag van de rector magnificus, prof. dr. G.J. van der Zwaan, ingevolge het besluit van het college voor promoties in het openbaar te verdedigen op 17 november 2011 des namiddags te 12.45 uur

door

Hendrik William van der Schuur

geboren op 29 maart 1986, te Amersfoort

Promotor:	Prof. dr. S. Brinkkemper
Co-promotor:	Dr. S. Jansen

This research was financially supported by Stabiplan B.V., the Netherlands and AFAS ERP Software B.V., the Netherlands.

Preface

This dissertation is all about the fascinating topic of in-the-field software operation. During my master's thesis research in 2008, I became interested in the relationship between software vendors and their customers, particularly their end-users: although most software vendors design, develop, test and deploy their software with great care, somehow, end-users always are frustrated by software nonetheless. Now it's three years later, and this dissertation has been finished. With the software operation knowledge research it is composed of, software vendors are provided with methodical and tool support for improvement of their software processes through knowledge of the in-the-field behavior of their software and end-users. Furthermore, this research may be considered as an initial, explorative attempt to position the concept of in-the-field software operation in the research domains of product software and process improvement. This dissertation can be considered as an attempt to make software-producing organizations aware of the software operation knowledge concept, as well as its value and potential with relation to process improvement.

During my PhD research, I have been accompanied and supported by many people. First, I would like to thank my co-promotor, Slinger Jansen (Roijackers), for his enthusiasm and support throughout the past years. Slinger, thanks for the numerous reviews of my work, and for being not easily convinced during the many discussions we have had. Next, I would like to express my gratitude to Sjaak Brinkkemper, my promotor. Thank you Sjaak, for the multitude of opportunities, suggestions, reviews, comments and advice you provided me with — I certainly would not have been where I am now without your guidance and support. I must also thank Gijs Willem Sloof and Bas van der Veldt, for allowing me to perform my research, and for being inspirational and encouraging bosses. It is an honor and delight for me to have you two standing beside me as paranymphs.

I would like to thank my colleagues, Inge, Jaap, Kevin, Willem, Marijn, Johan, Rik, Ronald and others. Special thanks go to Jurriaan Hage, Andy Zaidman, Ronald Dähne and Ad van der Hoeven for their reviews and other valuable contributions to this dissertation. Furthermore, I would like to thank the members of the reading committee, Paul Gruenbacher, Tom Mens, Willem-Jan van den Heuvel, Marko van Eekelen and Arie van Deursen for taking the time to read and judge this dissertation.

Of course, lots of thanks go to the friends and family that surrounded me throughout the past years. Krijn, Rob, Sanjay, Niels, and Michiel — I am thankful for the talks we had during lunch and coffee breaks, during walks and dinners, as well as for the advice and lessons I have learned from you guys. Gratitude goes to my parents, who have supported and encouraged me since elementary school: a big thank you for always being there for me all those years. Finally, there is Sacha, my love and fiancée. Dear Sacha, your love and loyalty are invaluable. I am looking forward to spending my life with you.

Henk van der Schuur
October 2011

Contents

If I have the gift of prophecy,
and know all mysteries
and all knowledge;
and if I have all faith,
so as to remove mountains,
but don't have love,
I am nothing.

1 Corinthians 13:2 (Word English Bible, 2002)

Part I

Introduction

1

Introduction

1.1 MOTIVATION

The software market is flourishing, and is growing at a spectacular rate. Between 2003 and 2010, the international product software industry has grown with 64% (est.), sustaining an average year-over-year growth of about 7.3% [OECD 2010]. Software-producing organizations strive for high levels of end-user satisfaction [Van der Schuur *et al.* 2011c], especially since customer demands regarding software quality are identified as the second factor 'of high importance' for innovation. Or as the organization for economic co-operation and development (OECD) states: '*As society in general comes to depend more on ICT and other products which are driven by software to conduct civic, economic, and social activities, user-centered functionality requirements will increasingly guide software and other ICT innovations. And users themselves — individuals, firms, and governments — will play a growing participatory role in driving this innovation*' (OECD, Innovation Strategy 2009 [Lippoldt and Stryszowski 2009]).

The motivation for this research lies in the fact that in their continuous strive to reach higher levels of end-user satisfaction and customer satisfaction, software-producing organizations do not recognize, and are thus unable to use, the broad potential of knowledge of in-the-field software operation. For example, less than one-third of these organizations makes use of crash and usage feedback reports to acquire knowledge of the in-the-field behavior of their software and end-users [Jansen *et al.* 2008, Jansen *et al.* 2010]. Consequently, we observe an emerging need for a guiding substrate that provides and structures directions for, inter alia, acquisition, analysis and presentation of such knowledge.

In this dissertation, that substrate is proposed as a body of knowledge for in-the-field software operation. We define the life cycle of knowledge gained from such software operation, as well as perspectives on this life cycle. The aim of this knowledge body is to provide software-producing organizations with methodical and tool support for improvement of their software processes, through knowledge of the in-the-field behavior of their software and end-users.

1.2 SCIENTIFIC RELEVANCE

Software production has been extensively researched in both software engineering (SE) and software product management (SPM) domains. Both domains have been described comprehensively.

First, the domain of software engineering has been detailed extensively through the Software Engineering Body of Knowledge (SWEBOK), which identifies software engineering knowledge areas such as software requirements, design, maintenance, configuration management and quality [Abran *et al.* 2004]. Authoritative works from Sommerville [Sommerville 2007], Pfleeger and Atlee [Pfleeger and Atlee 2009] and Pressman [Pressman 2010] are used by software engineers as a software engineering guide and reference. Second, similar to the software engineering domain, the domain of software product management has been described thoroughly. Product software concepts have been identified by Xu and Brinkkemper [Xu and Brinkkemper 2007], which are used as a basis for the software product management reference framework developed by Van de Weerd *et al.* [Van de Weerd *et al.* 2006, Van de Weerd 2009]. The framework identifies software product management business functions such as portfolio management, product roadmapping, requirements management and release planning. Software product management maturity models from Van de Weerd *et al.* [Van de Weerd *et al.* 2010] and Van Steenbergen *et al.* [Van Steenbergen *et al.* 2010] are used by software product managers as SPM guide and reference.

Although above references are rich resources for software engineers and software product managers, particular software production aspects that are part of both software engineering and software product management domains, are only vaguely described and underexposed till date. Examples are the in-the-field operation of software, the tracing and monitoring of such operation, as well as integration and utilization of knowledge of such operation in software processes such as software engineering and software product management.

In this dissertation, these aspects of software production are explored, and described by the software operation knowledge framework: a structure that describes parties, perspectives and life cycle processes related to knowledge

of in-the-field software operation. Using multiple research methods as part of conducting design research [Vaishnavi and Kuechler 2009], several artifacts that are part of the framework are designed, developed and empirically evaluated. For example, this dissertation research presents a novel technique for generic recording and visualization of in-the-field software operation. We show that the technique enables software vendors get a uniform insight in operation of their software in the field, and contributes to reduction of software maintenance effort. We present a template method for situational integration of operation information in software processes. We show that by instantiating the generic template method, situational instantiations of the method are constructed. Using the template method, software-producing organizations can improve their particular software processes with acquired software operation information. A third artifact that is presented in this dissertation is the software operation summary, a medium for presentation of software operation information. We show that the summary increases awareness of in-the-field software operation throughout software-producing organizations, and demonstrate that it contributes to the reach of consensus on software maintenance task prioritization. All artifacts constituting the software operation knowledge framework are described in detail in section 1.5.

1.3 POSITIONING THE RESEARCH

1.3.1 Product Software

Product software accounts for substantial economic activity all over the world [Lippoldt and Stryszowski 2009, OECD 2010]. In 2010 (2003), the total market of the product software industry was estimated to be 325 (199) billion USD, 9% (8.4) of the overall worldwide ICT spending of 3.6 trillion USD (2.3). '*Software spending has increased more rapidly (by 7.3% a year) than computer hardware (5.9%) [...] Communications services and hardware spending have increased by 6.6%*' [OECD 2010].

The economic importance of the product software phenomenon has been observed by Xu and Brinkkemper [Xu and Brinkkemper 2007], which induced them to define the concept of product software. In this dissertation, we use their definition of product software:

> ***Product software*** — A packaged configuration of software components or a software-based service, with auxiliary materials, which is released for and traded in a specific market [Xu and Brinkkemper 2007].

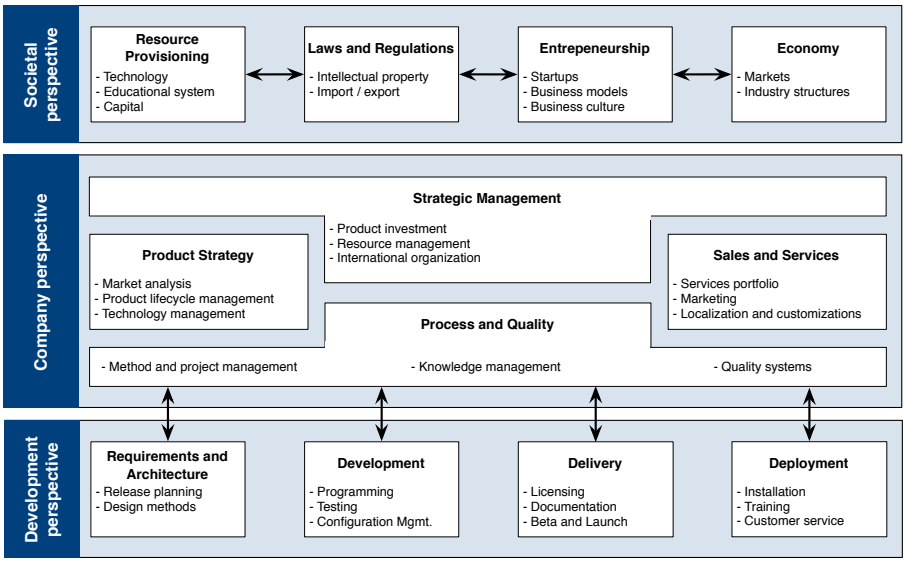


Figure 1.1 Research framework for product software [Xu and Brinkkemper 2007, Jansen 2007]

Product software is different from embedded software, since product software is sold separately from the hardware on which it will be operating. Furthermore, product software is different from tailor-made software, since it is delivered to a large number of customers and is operating in a wide variation of hardware and software environments [Xu and Brinkkemper 2007].

The research framework for product software (proposed by Xu and Brinkkemper [Xu and Brinkkemper 2007] and adapted by Jansen [Jansen 2007]; see figure 1.1) is composed of three perspectives from which the management of product software can be seen: the development perspective, the company perspective and the societal perspective. The development perspective concerns the processes that eventually produce software products that can readily be deployed at the customer site [Jansen 2007]. In relation to the software life cycle, the development perspective covers development, delivery and deployment processes. The company perspective concerns all non-development processes such as marketing, resource management, knowledge management, etc. Finally, the societal perspective covers external factors that may influence a product software vendor (e.g. laws, regulations, etc.). The focus of this dissertation is on improvement of, inter alia, all processes referred to by the product software research framework, through software operation knowledge.

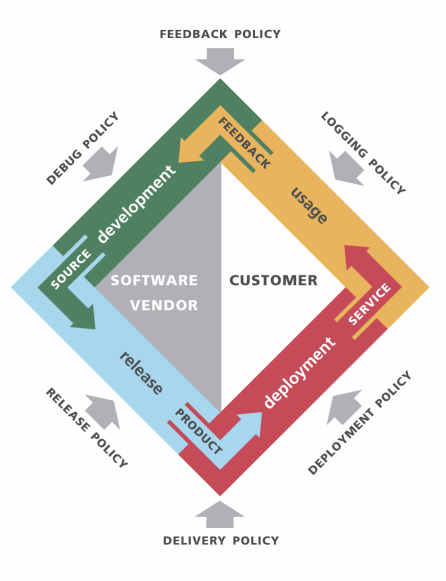


Figure 1.2 Continuous customer configuration updating model [Jansen 2007]

The release and trading activities referred to in the product software definition not only include the release of the product software into the market, but also the deployment of the software at the customer site, training users and potentially adaptation, integrations with other applications, customizations and maintenance services [Xu and Brinkkemper 2007]. Together with design and development processes, release and trading processes are part of the software life cycle [Davis *et al.* 1988, ISO/IEC 2008].

The continuous customer configuration updating model of Jansen [Jansen 2007] distinguishes two parties over which four software life cycle processes are divided: development and release processes are functioning at the software vendor site, deployment and usage processes at the customer site (see figure 1.2). This dissertation research covers both customer and software vendor parties: its focus is on improvement of processes of software vendors through knowledge that is acquired software operating at the vendor's customers. The concept of software operation knowledge, however, is best reflected by the FEEDBACK arrow from 'usage' to 'development': based on logging and feedback policies, software operation knowledge is acquired from in-the-field software operation, which can be used to support or improve software processes (e.g. software development).

1.3.2 Process Improvement

Software process improvement has been research extensively for the past decades. A multitude of approaches has been proposed to improve (aspects of) software processes [Hall *et al.* 2002], but many are considered unwieldy [Conradi and Fuggetta 2002]. For example, the Capability Maturity Model Integration (CMMI) may take twenty to twenty-eight months to be implemented [SEI 2010]. While some studies show that higher level CMMI companies are producing more reliable and predictable software [Fitzgerald and O'Kane 1999, Beecham *et al.* 2003], CMMI is frequently found too time-consuming or too complex to comprehend, implement and maintain effectively [Kuilboer and Ashrafi 2000, Staples *et al.* 2007, Smite and Gencel 2009]. Although derivatives of CMMI such as ISO/IEC 15504 (also known as SPICE, software process improvement and capability determination) were designed to be rapid and more flexible software process assessment methods [Dorling 1993], it is found that the capabilities stipulated in ISO/IEC 15504 do not necessarily improve project performance in small organizations [Rout *et al.* 2007].

Aforementioned studies show that software process improvement may fail from its own success: software-producing organizations implementing software process improvement approaches may be structurally hindered by additional administrative overhead induced by the improvement approach. This is confirmed by the recent departure from *fixed-level* maturity models (such as CMMI and ISO/IEC 15504) towards *focus area* maturity models, which are designed to provide more guidance in incrementally improvement of software processes [Van Steenbergen *et al.* 2010].

The focus of this research is on process improvement through software operation knowledge: it is investigated how software operation knowledge can contribute to improvement of software processes, instead of (or as a supplement to) process improvement through top-heavy, prescriptive frameworks and methods. This research is an effort to aid software vendors in finding the delicate balance between reaping the fruits of effective implementation of each of the software operation knowledge life cycle processes, and investing resources in implementation of these processes.

1.3.3 Software Operation Knowledge

In 1978, Stavely already argued that '*software system designers would benefit greatly from feedback about the consequences of a proposed design, if this feedback could be obtained early in the development process*' [Stavely 1978]. In the following years,

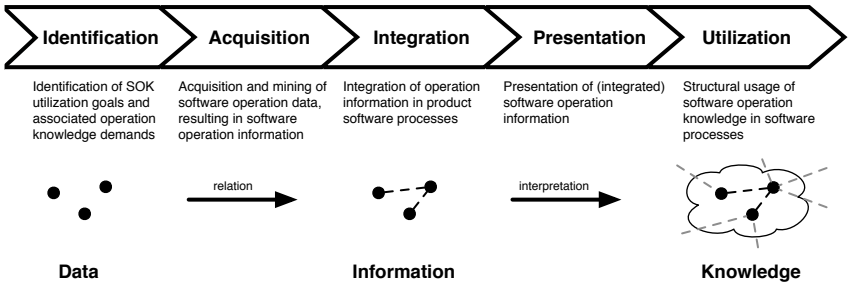


Figure 1.3 Software operation knowledge life cycle

several studies focused on involving software feedback in the software development life cycle [Kaiser *et al.* 1988, Selby *et al.* 1991, Musa 1993, Porter 1993, Wiegner and Nof 1993, Lehman 1996, Lehman and Ramil 1999], and on automation [Murphy 2004, Barry *et al.* 2007] and prediction [Nagappan *et al.* 2006, Zimmermann *et al.* 2008, Nagappan *et al.* 2010] of software feedback. Moreover, various studies were initiated to measure and improve the (perception of) behavior and operation of in-the-field software [Madhavji *et al.* 2006, Ebert and Dumke 2007], particularly the performance [Mathews 1987, Putrycz *et al.* 2005], quality [Mockus *et al.* 2005, Liu and Xu 2007] and usage [Kallepalli and Tian 2001, Hilbert and Redmiles 2000] of the software.

In this dissertation, we propose to integrate these properties of software operation in a single concept called *software operation knowledge* (abbreviated *SOK*) — in essence, it represents all knowledge one can acquire of the in-the-field operation of software. We research the life cycle of software operation knowledge, and we investigate how processes of this life cycle can be used for structural improvement of the software processes of software-producing organizations.

A hierarchy of data, information and knowledge [Kettinger and Li 2010] can be recognized in the software operation knowledge life cycle (see figure 1.3). After software operation knowledge demands have been *identified*, software operation data can be *acquired*. Relation of selected operation data results in software operation information, which then can be *integrated* in software processes and *presented* on various media. Interpretation of operation information results in software operation knowledge; structural use of such knowledge results in SOK *utilization*. A more detailed explication of the SOK life cycle is provided in chapter 2.

1.4 RESEARCH APPROACH

1.4.1 Research Questions

Supported by the previous sections, the main research question of this dissertation is stated as follows:

> ***MRQ*** *— How can software processes of product software vendors be improved through software operation knowledge?*

Although knowledge of in-the-field software operation is a broad topic that increasingly receives attention from both industry and science, it was still ill-defined at the time this research started. Authoritative works on software measurement [Ebert and Dumke 2007] and feedback [Madhavji *et al.* 2006] recognize the concept of software operation knowledge, but fail to identify the potential role of such knowledge in software process improvement.

To cover the complete software operation knowledge life cycle (see figure 1.3), six sub questions are formulated: each of the questions addresses at least one process of this life cycle.

> ***RQ 1*** *— What is the concept of software operation knowledge?*

The first step in answering the main research question is to define the concept of software operation knowledge, for instance by identifying different types of operation knowledge that are considered relevant and valuable for product software vendors (see figure 1.3). A reference framework is needed to allow adequate and consistent reasoning about software operation knowledge, as well as to effectively define the life cycle of software operation knowledge, and illustrate potential uses of such knowledge in the improvement of a vendor's products and internal software processes.

Second, to further establish and understand roles and applications of software operation knowledge between individual product software vendors, a classification of external software operation knowledge practices of vendors that successfully participate in software ecosystems, is needed. Therefore, the second sub question is

> ***RQ 2*** *— How can software operation knowledge practices within software ecosystems be classified?*

Having identified types, roles and applications of software operation knowledge, it needs to be identified *how* product software vendors can benefit from such knowledge. Hence, the following four sub questions are related to soft-

ware processes that can be improved through software operation knowledge. Before such knowledge can be used within organizations, underlying software operation data should be acquired. The third sub question is therefore stated as follows:

> ***RQ 3*** *— How can software maintenance effort be reduced through generic recording and visualization of operation of deployed software?*

Once software operation data has been acquired, software operation information can be extracted from these data (see figure 1.3). Software vendors are in need of methods to structurally integrate extracted operation information with their software processes. The fourth sub question is related to realizing process improvement through integration of extracted operation information and is defined as:

> ***RQ 4*** *— How can product software processes effectively be improved with acquired information of in-the-field software operation?*

Part of decision-making in software processes is presentation and visualization of the information on which decisions are based. To support steering and decision-making based on software operation knowledge, media or carriers are needed for presentation and visualization of operation information. Software operation knowledge media can, for instance, advance effective communication and discussion related to software operation knowledge, as well as contribute to (common) understanding of such knowledge. The fifth sub question is related to the creation of a medium on which such operation information can be summarized, and is formulated as follows.

> ***RQ 5*** *— Can prioritization of software maintenance tasks be improved through the concept of a software operation summary?*

Finally, effective utilization of software operation knowledge should lead to concrete and measurable process improvement at the software vendor site. Rapid development developments in industry demand software-producing organizations to be dynamic and agile, highly responsive to changes, while developing software at high speed, low cost and according to high quality standards. This sixth and final sub question focuses on changes in performance and usage of web services:

RQ 6 *— How can responsiveness to changes in service performance and usage be increased through utilization of knowledge of in-the-field software operation?*

Both the main research question and its sub questions are based on the hypothesis that they can be answered by providing support to product software vendors in the form of frameworks, methods or software tools. The following section describes the research approach that underlies the formation of these design artifacts.

1.4.2 Research Description

The field of research that covers application of information technology (IT) to human organizations is information systems (IS) research. Information systems are implemented within organizations for the purpose of improving the effectiveness and efficiency of that organization [Hevner *et al.* 2004]. Capabilities of the information system and characteristics of the organization, its work systems, its people, and its development and implementation methodologies together determine to which extent that purpose is achieved [Silver *et al.* 1995, Beynon-Davies 2010].

The information science discipline is characterized by two complementary but distinct paradigms: behavioral science and design science. Both paradigms are combined by the information systems research framework of Hevner *et al.* [Hevner *et al.* 2004]. Figure 1.4 shows the instantiation of the framework with the research of this dissertation into software operation knowledge.

IS Research

> Information science research typically is a combination of behavioral science and design science. Behavioral science seeks to develop and justify theories that explain or predict organizational and human phenomena surrounding the analysis, design, implementation, management, and use of information systems [Hevner *et al.* 2004]. Design science, on the other hand, seeks to create innovations that define the ideas, practices, technical capabilities, and products through which the analysis, design, implementation, management, and use of information systems can be effectively and efficiently accomplished [Denning 1997, Tsichritzis 1998]. Contributions to information science research involve both paradigms [March and Smith 1995].

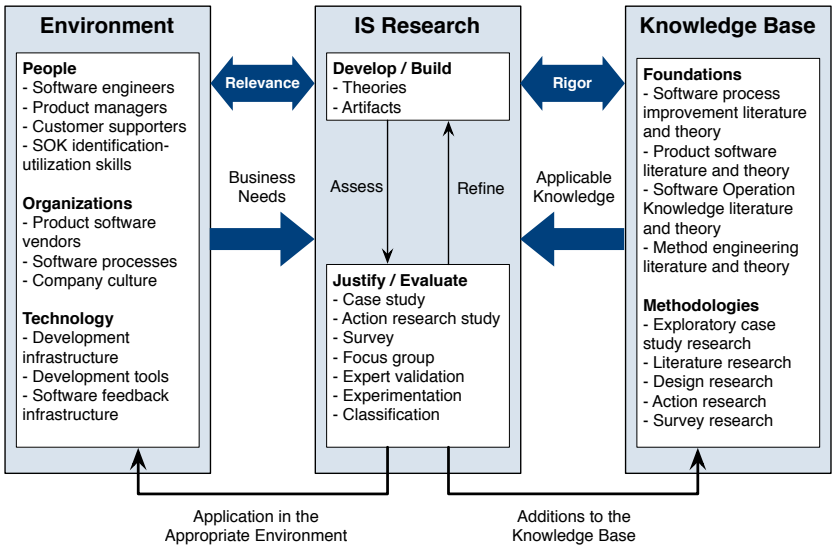


Figure 1.4 Dissertation research applied to the information systems research framework (adapted from [Hevner *et al.* 2004])

As this dissertation research is information systems research, it is a composition of the behavioral science and design science paradigms. Part of this research is the construction and evaluation of several design artifacts, such as the software operation knowledge framework and the template method for software operation knowledge integration (see table 1.1). The artifacts are built and evaluated through several justification and evaluation instruments.

Environment

The environment defines the problem space in which the phenomena of interest reside, and in which the research is conducted. The space is composed of industry people, business organizations and their existing and planned technologies and practices. Also, the environment incorporates goals, tasks, problems and opportunities that define business needs as they are perceived by people within the organization [Hevner *et al.* 2004]. Examples of people that are relevant in the problem space of this research are practically all people employed by product software vendors that can be effectively involved in, and benefit from software operation knowledge processes (e.g. software engineers, customer supporters, development managers, product managers, product software trainers, etc.). Obviously,

each employee possesses and lacks particular process skills that influence the problem space. Consequently, relevant organizations in this research problem space are organizations that develop and publish product software and have functioning product software processes (e.g. software development, software maintenance, software product management, software licensing, software training, research and innovation, etc.). Process maturity, company culture and history are factors that impact the problem space. The third relevant factor in the environment is technology. It includes process infrastructures (e.g. software development infrastructure, customer support infrastructure, etc.) as well as software tools that are used or desired to support these processes. As a composition of people, organizations and technology, the environment defines the business needs or 'problem' that can be perceived by the researcher. Research relevance is assured by framing research activities to address such needs or problems.

Knowledge base

The knowledge base provides the scientific foundations (e.g. theory and literature) and methodologies (e.g. the systems of methods used in a particular area of information systems research) from and through which information science research is performed [Hevner *et al.* 2004]. As detailed in section 1.3, prior research related to product software, software operation knowledge and software process improvement forms the foundation and context for this research in the form of theories, frameworks, models, methods, etc. Research rigor is achieved by appropriately and effectively applying existing foundations and methodologies. Contributions of behavioral science and design science in information systems research are applied to business need in a particular environment, and add to the content of the knowledge base for further research and practice [Hevner *et al.* 2004].

1.4.3 Research Methods

The research in this dissertation is carried out by using several research methods: case study research, field study research, action research and survey research. Simultaneously, as stated, this dissertation research is a composition of the behavioral science and design science paradigms. Table 1.1 lists per chapter what are applicable science paradigms, the applied research methods, resulting design artifacts and related SOK life cycle processes. Next, characteristics and strengths of each of the research methods are listed.

Action research

Action research is a research method that it is grounded in practical action, aimed at solving an immediate problem situation while carefully informing theory. Typical action researchers believe that human organizations, as a context that interacts with information technologies, can only be understood as whole entities. The fundamental contention of the action researcher is that complex social processes can be studied best by introducing changes into these processes and observing the effects of these changes [Baskerville 1999]. Action research embodies a strategy for studying change in organizations, involving the formulation of theory, intervention and action-taking, in order to introduce change into the study subject and analysis of the ensuing change behavior of the study subject. As a result of its orientation toward change, action research is relevant for the study of information systems development [Baskerville and Pries-Heje 1999]. In this dissertation research, action research has been performed to study application of the software operation knowledge integration template method (see chapter 5). To ensure rigor and relevance of this action research, we adhere to a set of interdependent action research principles and associated criteria [Davison *et al.* 2004].

Case study

Case study is the study of the particularity and complexity of a single case or multiple cases, coming to understand case activity within important circumstances [Stake 1995]. Case study research involves the close examination of people, topics, issues, or programs, for purposes of understanding [Hays 2003] and theory building and testing [Eisenhardt 1989, Dul and Hak 2008]. In this dissertation research, case studies are performed to build and evaluate design artifacts such as the software operation knowledge framework (see chapter 2), to evaluate software tools such as software operation knowledge acquisition tools (see chapter 7), and to build theories such as a classification of software operation knowledge propagation practices (see chapter 3). As stated by Darke *et al.* [Darke *et al.* 1998], successfully completing case study research within the field of information systems requires initiative, pragmatism, the ability to take advantage of unexpected opportunities, and optimism and persistence in the face of difficulties and unexpected events, particularly during data collection activities. Case study protocols [Jansen and Brinkkemper 2008, Yin 2009] were followed for collection of case study data.

Field study

A field study is used to monitor use of an artifact in multiple projects or environments [Hevner *et al.* 2004]. There is no hard and straightforward distinction between performing multiple in-depth case studies a field study [Klein and Myers 1999]. In this dissertation, the term 'field study' is used to denote the monitoring and evaluation of an artifact's in-the-field performance and behavior through mixed-method research [Sandelowski 2000, Jansen and Brinkkemper 2008]. In chapter 4, as part of a field study, two case studies and an experiment were conducted to monitor and evaluate both behavior and performance of a software operation knowledge acquisition and presentation tool.

Focus group

The focus group research method collects data through group interaction on a topic determined by the researcher. Focus groups should be distinguished from methods that collect data from naturally occurring group discussions where no one acts as an interviewer [Morgan 1996]. In this dissertation, focus groups composed from experts from industry were used to evaluate the software operation knowledge framework (see chapter 2) as well as the software operation knowledge acquisition and presentation technique and tool (see chapter 4).

Survey research

A survey is a comprehensive system for collecting information to describe, compare or explain knowledge, attitudes and behavior. The survey instrument is part of a larger survey process with clearly-defined activities such as survey planning and scheduling, survey design, participant selection, data analysis and result reporting [Pfleeger and Kitchenham 2001]. Surveys can be supervised, semi-supervised and unsupervised. In this dissertation research, we designed and carried out an unsupervised survey to investigate expectations regarding the software operation summary concept (see chapter 6).

Note that the literature research method has been omitted from the list above, since this method was used integrally throughout the research.

1.5 DISSERTATION OUTLINE

Apart from its introduction (chapter 1) and conclusion (chapter 8), this dissertation is composed of six chapters that all correspond to one of the research questions described in section 1.4.1. Furthermore, each of these six chapters covers

at least one process of the software operation knowledge life cycle, of which a detailed description is provided in chapter 2). Chapters 2 and 3 first cover the introduction of the software operation knowledge concept; chapters 4 to 7 cover the subject of process improvement through software operation knowledge.

Table 1.1 provides an overview of the outline of this dissertation by associating, inter alia, chapters, research questions, research methods and SOK life cycle processes. Next, all chapters are described in detail.

Ch.	Research question	Science paradigms	Research methods	Design artifacts	SOK life cycle processes
2	RQ 1	Behavioral, Design	Focus group, case studies	SOK definition; SOK framework	Identification
3	RQ 2	Behavioral, Design	Case studies	SOK propagation practices classification	Utilization
4	RQ 3	Design	Field study, Focus group	SOK acquisition and presentation technique; SOK acquisition and presentation tool	Acquisition, Presentation
5	RQ 4	Design	Action research	SOK integration template method	Integration
6	RQ 5	Behavioral, Design	Survey, case study	Software operation summary	Presentation
7	RQ 6	Design	Case study	SKU concept, indices and report; SOK presentation tool	Presentation, Utilization

Table 1.1 Overview of this dissertation research

Chapter 1: Introduction

The first chapter describes the motivation and relevance of this research. Furthermore, the research is positioned in context and described through research questions. Finally, the outline of this dissertation research is presented.

Chapter 2: A Reference Framework for SOK Utilization

This chapter introduces the concept of software operation knowledge and the software operation knowledge framework. The software operation knowledge concept is an effort to encompass and categorize the plethora of definitions and metrics that exist to respectively describe and measure software operation. Furthermore, by identifying parties, perspectives and processes that constitute the software operation knowledge life cycle, the software operation knowledge framework illustrates software

operation knowledge flows through software-producing organizations. Simultaneously, the framework serves as a guiding substrate in determining the scope of this research. The framework was evaluated through three case studies and a focus group questionnaire. This chapter has been published as a full research paper on the EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA) [Van der Schuur *et al.* 2010].

Chapter 3: On the Role of SOK within Software Ecosystems

In this chapter, the concept of software operation knowledge is further established by identification and classification of operational software operation knowledge practices of software-producing organizations in software ecosystems. The classification is constructed based on four case studies. It illustrates the value and importance of software operation knowledge within the context of software ecosystems. This chapter has been published as a full research paper on the International Conference on Management of Emergent Digital EcoSystems (MEDES) [Van der Schuur *et al.* 2011d].

Chapter 4: Reducing Maintenance Effort through SOK

Chapter 4 is the first chapter that covers the subject of process improvement through software operation knowledge. As an attempt to increase efficiency of software maintenance processes, this chapter introduces a technique and corresponding prototype tool [Nuntia 2011] for generic acquisition and presentation of in-the-field software operation. Both the technique and tool are evaluated through a field study (consisting of an experiment and two case studies) and a focus group. This chapter has been published as a full research paper on the European Conference on Software Maintenance and Reengineering (CSMR) [Van der Schuur *et al.* 2011b].

Chapter 5: Pragmatic Process Improvement through SOK

In this chapter, the template method for integration of software operation information with product software processes is introduced to aid and stimulate software-producing organizations in structurally improving their functioning processes through acquired operation information. The template method has been evaluated through an action research study of ten months, during which the template method was instantiated for three software processes of a European software vendor. Based on the study, four lessons learned are identified. This chapter has been pub-

lished as a full research paper on the International Conference on Product Focused Software Development and Process Improvement (PROFES) [Van der Schuur *et al.* 2011a].

Chapter 6: Leveraging SOK for Prioritization of Maintenance Tasks

This chapter introduces the software operation summary as a strive to support software processes by providing software operation knowledge, more specifically, to improve prioritization of software maintenance tasks by fostering the reach of consensus on such prioritization. Soundness and validity of the summary have been evaluated through an extensive survey among Dutch product software vendors as well as a case study at an European software vendor. Survey results confirm the need for a software operation summary. Furthermore, through the survey, crash report data types are identified on which such a summary, used for fostering reach of consensus on prioritization of software maintenance tasks. Through the case study, the value of a software operation summary that is based on crash report data, is empirically evaluated. This chapter has been published as a full research paper on the International Conference on Software Process and Product Measurement (MENSURA) [Van der Schuur *et al.* 2011c].

Chapter 7: Becoming Responsive to Service Usage and Performance Changes

In this chapter we introduce service knowledge utilization (SKU) as an approach to increase a software vendor's flexibility, and responsiveness to changes in the performance and usage of its service-based, online software. Furthermore, we introduce the SKU report, a report that is composed of three metrics-based indices expressing service performance, usability and client utilization (the tool described in this chapter can be considered as an initial version of the tool described in chapter 4). The report was evaluated through a case study at a Dutch software vendor. This chapter has been published as a full research paper on the International Workshop on Software Evolution and Evolvability (Evol) [Van der Schuur *et al.* 2008].

Chapter 8: Conclusion

In the last chapter of this dissertation, the main research question as well as all sub research questions are answered. Furthermore, research implications and limitations are discussed. Finally, we provide a reflection on this dissertation research and describe future research perspectives.

Part II

The Concept of Software Operation Knowledge

2

A Reference Framework for Utilization of Software Operation Knowledge

ABSTRACT

Knowledge of in-the-field software operation is a broad but ill-defined and fragmentarily supported subject and it is unclear how software vendors can take advantage of such knowledge. This paper introduces and defines software operation knowledge to unify existing definitions, and presents an empirically evaluated framework that is designed to aid product software vendors in gaining insight in the potential role of such knowledge in advancement of their products, practices and processes. The results of extensive case studies performed at three European software vendors show that if used correctly, software operation knowledge enables vendors to increase software quality and improve end-user experience. However, case study results also illustrate that the state of knowledge integration is still pragmatic and immature. Vendors have to adapt their workflows, processes and tools to enable structural software operation knowledge utilization.*

*This work has been published as *A Reference Framework for Utilization of Software Operation Knowledge* in the proceedings of the 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2010) [Van der Schuur *et al.* 2010]. It is co-authored by Slinger Jansen and Sjaak Brinkkemper.

2.1 INTRODUCTION

Software vendors have recently begun discovering the yields of software and end-user feedback. For example, by implementing feedback reporting in its operating systems, a large software vendor discovered that circa 50% of failures are caused by one percent of software bugs [Brelsford *et al.* 2002]. If used correctly, feedback enables software vendors to establish how successful their products and services are at achieving their goals in the field. These goals are dependent on software end-users, and constitute aspects such as performance, quality and usability. With the increase of software complexity and ever higher end-user expectations, advanced techniques are required to monitor operations of software in the field. Common examples are crash reporting applications and service performance monitoring tools.

More exotic mechanisms exist, such as 'software tomography' [Bowring *et al.* 2003] for monitoring specific aspects of an application, end-user tracing for UI improvement, as well as mechanisms for providing and delivering end-user feedback. The software community has picked up on the need for tools to support software and end-user feedback concerns. Google, for example, has created the Google Website Optimizer [GWO], which enables website builders to leverage end-user behavior by presenting end-users with different user interfaces and then measuring differences in conversion rates, site effectiveness, visitor satisfaction, etc. As another example, Mozilla has developed the Firefox Test Pilot plug-in [MLTP] to get operation feedback and usage traces from circa one percent of its end-users.

It remains unclear, however, how and to which extent software and end-user feedback can be used to improve a software vendor's practices, processes and products. Several research examples that focus on specific solutions and domains can be found. The Skoll project [Memon *et al.* 2004] focuses on user community-supported quality assurance of software operating in large configuration spaces. Furthermore, the GAMMA project [Orso *et al.* 2002] uses software tomography to gather useful information from deployed software and focuses on determining effective probe insertion locations. While these examples show that research in this area is fragmented, software and end-user feedback are generally used as main data source.

An integrated view is needed that provides product software vendors with insight in the potential role of such feedback in advancement of their products, practices and processes.

The contribution of this paper is twofold:

- A definition is introduced to unify existing definitions and uses of software feedback, as well as types of knowledge emerging from in-the-field software operation
- A framework is presented that models the life cycle of such knowledge as well as product software perspectives from which processes of this life cycle can be perceived.

Both the definition and the framework are empirically evaluated with a questionnaire and three investigative case studies at European software vendors. This paper continues with placing our work into context and with the introduction of the software operation knowledge definition (section 2.2) and framework (section 2.3). The research evaluation approach is described in section 2.4. Next, results of our empirical study are presented in sections 2.5 and 2.6. Finally, limitations of this research are discussed in section 2.7 and research conclusions are presented in section 2.8.

2.2 SOFTWARE OPERATION KNOWLEDGE (SOK)

Till date, various research has been conducted concerning the subject of knowledge of in-the-field software operation. For example, software measurement, monitoring and feedback techniques have been proposed [Ebert and Dumke 2007, Madhavji *et al.* 2006, Rompaey *et al.* 2009] and software operation data acquisition techniques and tools have been developed [Bowring *et al.* 2003, Van der Schuur *et al.* 2008]. Little research has been initiated to incorporate all processes in one framework, however. Selby *et al.* [Selby *et al.* 1991] have proposed a framework that supports multiple evaluation and feedback paradigms, but mainly focus on architectural principles for designing metric-driven analysis and feedback systems; the authors do not address integration, presentation and utilization aspects of software feedback. The work of Lehman and Ramil [Lehman and Ramil 1999] analyzes and quantifies the impact of feedback on (improving) 'the global software process' and can be seen as an argument for using software operation knowledge to advance software engineering processes. However, the research focuses on process improvement through software evolution process measurement and modeling, and does not consider the life cycle of feedback itself (as detailed in section 2.3). Tautz and Althoff [Tautz and Althoff 1997] propose case-based reasoning techniques to reuse software knowledge, but concentrate on 'improving productivity and reliability of software development' and do not consider other software engineering processes.

In short, knowledge of in-the-field software operation is an emerging and broad subject and is ill-defined till date. We provide the following definition:

> ***Software Operation Knowledge*** — Knowledge of in-the-field performance, quality and usage of software, and knowledge of in-the-field end-user software experience feedback.

SOK (κ) consists of four knowledge types: *performance* (κ_P), *quality* (κ_Q), *usage* (κ_U) and end-user *feedback* (κ_F) knowledge. Next, we detail each SOK type in terms of concepts and metrics that are encountered in research on software analysis, measurement and feedback[1].

2.2.1 Performance (κ_P)

Software performance can be specified on many types of software resources, with different measurement units. In their research on performance techniques for commercial off-the-shelf (COTS) software, Putrycz *et al.* [Putrycz *et al.* 2005] state that software performance characteristics can be described by using benchmarks (giving the delay for a component in a particular configuration, for example) or by using a causal model based on performance data. As Putrycz further states, performance data consist of three kinds of data: *device demands* (e.g., average CPU time for a component's operation), *interaction attributes* (e.g., number of required service operations demanded per component operation) and *logical resources* (e.g., threads, buffers and caches associated with a component). According to Johnson *et al.* [Johnson *et al.* 2007], elapsed time, transaction throughput and transaction response time are among most common ways to specify software performance. In their research, performance areas are (1) response time for input and output operations, (2) maximum sustainable throughput and response time, and (3) time consumed by each software layer. The performance of service-based software in particular is measured in terms of throughput (number of service requests served in a given time frame) and latency (round-trip time between sending a request and receiving the response), where higher throughput and lower latency values represent higher service performance [Mani and Nagarajan 2002]. Software performance knowledge (κ_P) consists of all performance data types identified by Putrycz, as well as the performance specifications of Johnson and both throughput and latency metrics.

[1] Note that the concepts and metrics mentioned are considered characteristic, exemplary and not complete.

2.2.2 Quality (κ_Q)

Several software quality models with diverse sets of characteristics have been proposed, and as observed by Bøegh [Bøegh 2008], many perspectives on what composes a software quality model's key quality characteristics exist. The ISO 9126 quality model [ISO/IEC 2001] is well-known and accepted in both industry and empirical research. The model classifies software quality into a structured set of characteristics and sub characteristics, divided into three quality views: internal quality, external quality and quality in use. While the internal quality view (based on the characteristics *Functionality*, *Reliability*, *Usability*, *Efficiency*, *Maintainability*, *Portability*) is concerned with static software properties that do not depend on software operation, the external quality view (which is based on the same characteristics as the internal quality view) is related to metrics applicable to the dynamic aspects of deployed software operating on computer hardware (e.g., number of exceptions, crash report details and mean time between failures [Bøegh 2008]). The quality-in-use view of the model (based on characteristics *Effectiveness*, *Productivity*, *Safety*, *Satisfaction*) is concerned with end-users performing tasks by using software in the field. Characteristics related to the quality-in-use view can only be measured when the deployed software product is used in real conditions. Examples of metrics related to this view include end-user productivity and end-user satisfaction. As shown by Gyimóthy *et al.* [Gyimóthy *et al.* 2005], software quality can be estimated by means of source code (metrics) analysis: forming the DNA of software, source code determines behavior of in-the-field software operation (e.g., algorithm complexity influences software quality).

Quality of service-based software and Web services has also been researched extensively. Yu and Lin [Yu and Lin 2005] propose a set of QoS attributes (e.g. *Cost*, *Reliability*, *Availability*), as impact factors of service selection algorithm creation. In their research on the construction of a Web service quality model, Zeng *et al.* [Zeng *et al.* 2003] present four generic service quality criteria: *Execution price*, *Reliability*, *Availability* and *Reputation*. With respect to the SOK concept, characteristics associated with both external quality and quality-in-use views, as well as the source code metrics and service quality metrics referred to are covered by the software quality knowledge type κ_Q.

2.2.3 Usage (κ_U)

Software usage describes how software is used in the field by its end-users and how software responds to end-user behavior. Analogously to the quality-in-

use view of the ISO 9126 quality model, knowledge of software usage can only be acquired during in-the-field software operation. The usage model presented by Simmons [Simmons 2006] contains three tiers: *supporting data*, *usage overview* and *usage details*, where the usage details tier contains actual usage data. Software usage is described in terms of user interface paths, method calls and object initiations.

Concerning service-based software and Web services, software usage is specified in terms of service requests, web method calls and service error types [Kallepalli and Tian 2001]. κ_U is covered by the *usage details* tier of Simmon's usage model: we consider the extent to which the software usage specifications of tiers other than the usage details tier contribute to this knowledge type, as minimal.

2.2.4 End-user Feedback (κ_F)

End-user feedback is a collection of end-user software appreciation, criticism on certain software usage aspects, and general software experience. For example, feedback from end-users frequently consists of (1) a subject that describes the aspect of the software the end-user is giving feedback on, (2) a rating that quantifies the end-user's appreciation of the aspect, and (3) feedback motivation or explanation. Average feedback rating and customer satisfaction level are metrics corresponding to end-user feedback. In short, end-user feedback knowledge (κ_F) consists of all feedback on software operation provided by end-users of the software.

2.3 SOK FRAMEWORK

Partially based on our observations of industry practices, the software operation knowledge framework (see figure 2.1) describes the SOK life cycle processes, and models the flow of software operation data, information and knowledge through software vendor tools and processes, from three product software perspectives. The framework serves as a guiding substrate in determining the scope of our SOK research, and might fulfill an equivalent role in other research initiatives on software engineering and evolution, tool development or change management. The stakeholders, processes and perspectives that constitute the framework are detailed in the following sections.

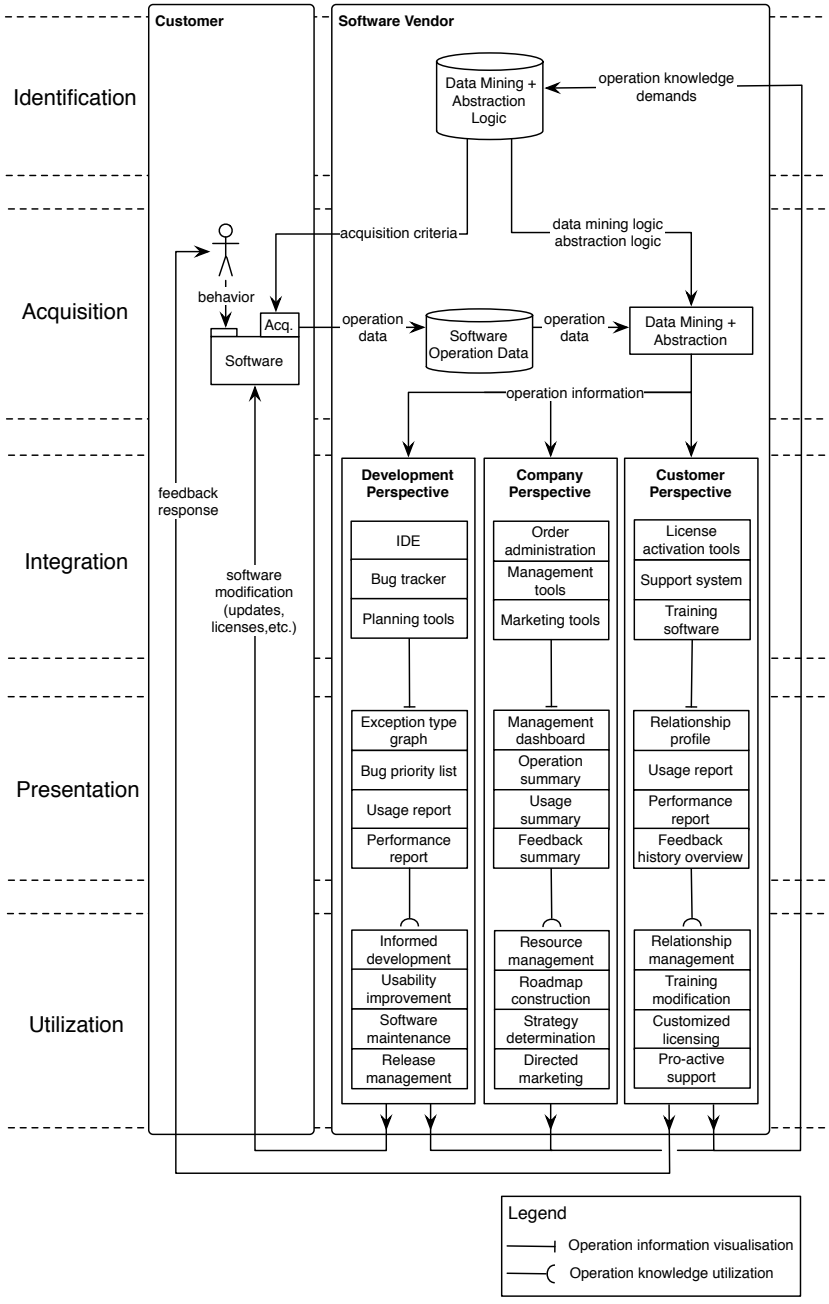


Figure 2.1 Software operation knowledge framework

2.3.1 Stakeholders

The SOK framework distinguishes two stakeholders: software vendors and customers. The 'Customer' stakeholder represents a software vendor's business-to-consumer (B2C) customers as well as business-to-business (B2B) customers. End-users, or end-users of third party enterprises are considered B2C customers, while external software vendors, partners that have licensed software of the software vendor as well as end-users of these external software vendors are considered B2B customers. End-users and their behavior form the initial source of software operation knowledge; software vendors assemble operation data from their customers and potentially respond to these data, for example through their software development, release, marketing or quality assurance processes. Other parties operating within a vendor's 'ecosystem(s)', like those defined by Jansen *et al.* [Jansen *et al.* 2009], are considered out the scope of the framework.

2.3.2 Processes

The SOK life cycle processes depicted in figure 2.1 form the SOK life cycle and illustrate the transformation of software operation data (*Identification*, *Acquisition*) via software operation information (*Acquisition*, *Integration*, *Presentation*) to software operation knowledge (*Presentation*, *Utilization*). The processes take place subsequently, cyclically and independently per SOK type. Five life cycle processes can be identified:

Identification

The first SOK process encompasses identification of SOK utilization goals and associated operation knowledge demands. Since software operation data acquisition potentially introduces a data explosion that hinders software vendors to successfully utilize SOK, directed acquisition is required. The amount of software operation data that is acquired, is controlled by acquisition criteria. Operation data are associated to one or more SOK types, and to each type $k \in \kappa$, a weight w is assigned that represents the acquisition priority of k. Furthermore, abstraction logic is defined for software operation data aggregation and encapsulation. Operation knowledge demands resulting from the utilization process are translated into acquisition criteria to direct SOK acquisition, and transformed into mining and abstraction logic to control mining of acquired data. The SOK identification process results in a set of acquisition criteria, as well as mining and abstraction logic to steer acquisition of SOK in the next process.

Acquisition

The SOK acquisition process is concerned with a number of sub processes. First, the behavior of end-users is translated to software operation data, taking into account the acquisition criteria defined in the SOK identification process. Second, software operation data are transferred from servers or workstations at which the software is deployed, to the software vendor. Next, based on mining and abstraction logic defined in the previous process, software operation data sources are identified and software operation information is extracted from all acquired operation data. Software operation information constitutes the input for the SOK integration process.

Although software operation data are often acquired manually by extending the software code base with log code or trace classes, software operation data can also be automatically deduced from deployed software [Bowring *et al.* 2003, Orso *et al.* 2002, Clause and Orso 2007]. Like logging, software operation data acquisition can be considered as a typical *cross-cutting concern* and can thus be implemented by using aspect-oriented programming (AOP) techniques [Van der Schuur *et al.* 2008]. Note that the amount of software operation data that is acquired depends on both an end-user's software usage behavior as well as the type of software operation knowledge $k \in \kappa$ that is eventually extracted from the operation data. While operation data associated with most operation knowledge types $(\kappa_P, \kappa_Q, \kappa_U)$ can be acquired automatically during software operation, the amount of acquired operation data associated with κ_F depends on and end-user's willingness to submit feedback.

Depending on security, regulation or capacity constraints, software operation data may be transferred to the software vendor in real-time or according to a schedule. Compression, AOP and tomography techniques [Bowring *et al.* 2003] can be used to configure and limit the amount of data that is transferred. Abstraction and data mining techniques are applied to the operation data stored at the software vendor, to aggregate, generalize or filter operation data, or to verify the data are representative with respect to the identified utilization goals. The mining and abstraction of operation data results in software operation *information*.

Integration

In the (optional) integration process, software operation information resulting from the acquisition process is integrated into a software vendor's existing processes and infrastructures. Existing processes and workflows

may have to be adapted, and plug-ins, conversion components or mediator services may be developed to make use of the available software operation information and to enable purposeful, context-dependent presentation and utilization of acquired SOK. For example, a plug-in may be developed to integrate software operation information of a particular code file into integrated development environments (IDE). Also, a software information conversion service could be developed to automatically register unhandled exceptions in the software vendor's bug tracker.

Presentation

The fourth SOK process is concerned with the presentation of software operation information. Data resulting from the integration process is visualized using graphs, diagrams or other presentation artefacts, possibly by means of the integration plug-ins or tools developed in that process. For example, based on exception event data, a bar chart can be created showing exception frequencies per software component. Note that each of the framework perspectives (described in section 2.3.3) may require a different visual representation of software operation information, illustrating the data at various levels of detail. Especially when presented in combination with historical software operation information or with other data (e.g., release schedules or bug tracker data), new insights and SOK are gained in the presentation process.

Utilization

The last SOK process describes processes as well as response actions that may be the result of effective SOK utilization. For example, integration and presentation of query timings, exception statistics and usage traces in an IDE respectively provides software developers with knowledge and insights about the performance, quality and usage of their software in the field, which contributes to 'informed software development'. Also, acquired software operation knowledge supports concrete decision making. For instance, software quality knowledge and end-user feedback knowledge support usability and release management decisions. In retrospect consideration of the SOK utilization process potentially results in new operation knowledge demands, which form input for the identification process. Also, SOK utilization may result in software modification, feedback response as well as propagation of SOK to particular stakeholders (e.g. partner vendors).

2.3.3 Perspectives

The route of SOK through integration, presentation and utilization processes can be observed from three product software perspectives, that find their origin in the product software research framework of Brinkkemper and Xu [Xu and Brinkkemper 2007].

First, the *Development* perspective concerns all processes that contribute to production of software products that can readily be deployed at customers. Second, the *Company* perspective concerns processes indirectly related to software development, such as marketing, sales, and quality control. Third, the *Customer* perspective represents all factors and processes that influence the existing relationship between a software vendor and its customers, such as training, support and relationship management processes.

SOK that is integrated with development tools, supports software engineering processes or results in software modifications, routes through the development perspective. Software operation knowledge that is instrumental to the indirect effects of a vendor's software engineering processes (e.g. resource management, strategy determination, roadmap development) routes through the company perspective. SOK that contributes to processes regarding a vendor's existing customers, or contributes to effective response to end-user feedback, routes through the customer perspective.

2.4 EMPIRICAL EVALUATION

The SOK framework (and therewith the SOK definition) has been empirically evaluated in two ways. First, to identify the soundness of the framework, a questionnaire with questions on the SOK definition and framework was presented to a focus group consisting of chief technology officers and managers of European software vendors, and several group discussions were held. Second, to identify the utility of the SOK framework and evaluate it in practice, extensive industrial case studies have been carried out at three software vendors that have implemented one or more SOK life cycle processes. The maturity with which such processes were implemented was used as a basis for selecting organizations.

2.4.1 Questionnaire Approach

The questionnaire consists of 21 questions divided over four sections[2]. The first section considers subject employment and experience, the size of the vendor and the vendor's main software product or service. Next, for each software operation knowledge type described in section 2.2, the focus group participants were asked whether they considered the knowledge type to be part of software operation knowledge. Third, the participants were asked if they found any processes, perspectives, flows or other elements missing from the framework that should be added. Finally, in the fourth questionnaire section, participants were inquired about activities and processes that can be improved by utilization of (a particular type of) software operation knowledge.

The questionnaire was answered by three CTOs, four product research and development managers and three lead software architects. All subjects are employed by different European software vendors, varying in size from 15 to more than 2,500 employees (626 employees on average, $\sigma = 1,065$ employees). The vendors build product or service software that has been available for between three months and 25 years (12.3 years on average, $\sigma = 8.8$ years), and of which each vendor has released five to twenty major versions. The subjects were recruited by means of an invitation sent to our professional and educational networks. All subjects were physically present in one room and answered the questionnaire questions digitally. In a one-hour presentation, the SOK concept and framework were introduced to the subjects prior to filling in the questionnaire, in order to assure common understanding among the subjects.

2.4.2 Case Study Approach

Case study techniques described by Yin [Yin 2009] have been used to gather evidence and determine the state of practice regarding identification, acquisition, integration, presentation and utilization of software operation knowledge at each of the participating case study vendors:

Document Study

Software architecture specifications, process descriptions and memos provided by the vendor were studied to get insight in its SOK life cycle processes.

[2]See appendix A for a list of all questions.

Interviews

Fifteen semi-structured interviews have been conducted with product managers, senior software engineers and software testers employed by the vendor. Interviewees were asked questions related to SOK identification, acquisition, integration, presentation and utilization processes and tools currently implemented at the vendor, and were asked to criticize and complement the stakeholders, processes and perspectives that constitute the SOK framework.

Software Study

The software of (and with) which the vendor acquires operation data was studied, in order to identify the type and complexity of the software and to analyze the vendors' data acquisition techniques.

Direct Observations

Observations were made during our presence at the vendors. For example, software development and release management meetings were attended.

Before any evidence was gathered at each of the vendors, an introductory session was held. During this session, the SOK framework and all related SOK concepts and definitions were presented to minimize discrepant understanding and to ensure that case study results could be compared adequately. Document and software study findings were cross-checked with interview questions and answers to gain correct and consistent evidence. Additional interviews were performed to clarify vague answers and to substantiate results (triangulation). To diminish our personal bias, (1) case study participants were informed about the goals of the study in advance; (2) each of the case studies was carried out with researchers present at the vendor site and (3) case study results were reviewed by corresponding case study interviewees. The case study database was reviewed by other researchers on completeness and consistency afterwards.

2.5 QUESTIONNAIRE RESULTS

Concerning the definition of the SOK concept, eight subjects indicated that they consider knowledge types κ_P, κ_Q, κ_U as well as κ_F as part of the concept. Most subjects considered the SOK definition complete in terms of its knowledge types: only subject [S6] suggested an additional $k \in \kappa$, κ_E, representing knowledge of the effectiveness of software in the field. As described in section 2.2, we consider $\kappa_E \subset \kappa_Q$, in conformity with the quality-in-use view of the ISO 9126

quality model. One subject [S3] considered none of the types $k \in \kappa$ as part of the SOK concept, and one subject [S10] only found κ_Q part of the concept.

Regarding the soundness of stakeholders, processes and perspectives that constitute the framework, subjects [S2, S6, S7, S8, S10] indicated that the SOK framework should contain external stakeholders, such as 'external vendors' that have licensed software from the software vendor or technical partners that supply software components to the software vendor. As stated in section 2.3, external stakeholders are represented by the 'Customer' stakeholder of the framework. Furthermore, questionnaire participants suggested a number of additional perspectives. [S4] suggested a 'Competition perspective' but noted that this perspective could also be part of the company perspective. [S9] proposed a 'Business perspective' that encompasses the role of SOK in context of a vendor's partners and competitors. Regarding a vendor's partners, we consider this perspective covered by the company perspective and the customer stakeholder; the role of SOK in the context of a vendor's competitors is considered out of the scope of the SOK framework.

The questionnaire participants were in harmony with respect to their opinion on the five software operation knowledge life cycle processes: no subject suggested a new process that is not already covered by a processes currently part of the framework. [S1] noted that it is not essential for operation data to pass all processes in all situations, and indicated that under certain circumstances, processes could be skipped or merged. As stated in section 2.3.2, the framework processes are descriptive rather than prescriptive and it is possible to skip or stop a process when it is clear that the goals of a certain improvement have been reached. Lastly, subject [S2] advocated to also include external systems to measure the availability and response times of those systems. As detailed in section 2.2, these metrics are part of the SOK definition. External systems are considered out of the SOK framework scope, however.

With respect to improvement of existing activities and processes by means of SOK, subjects [S1, S2, S6, S7] mentioned that their software development processes could be improved by using κ_P, for example in the process of prioritizing bug reports. Subjects [S3] and [S10] indicated that κ_P could contribute to improvement of research and development processes. Furthermore, subjects noted that software development, software maintenance and software testing processes could be improved by utilization of κ_Q and κ_U. [S5] and [S6] mentioned that software quality could be increased by using κ_U, since they consider software quality knowledge to be supportive in the process of pro-active bug fixing; [S5] added that '[without κ_U,] software developers may have a very distorted notion of the concept of software quality'. Software testing was men-

tioned by subjects [S1] and [S5] as a process that can be improved by utilization of κ_Q and κ_U. According to these subjects, SOK potentially contributes to 'the design of realistic test scenarios'. Finally, subjects [S5, S6, S10] noted that κ_F enables 'to determine which software requirements are important, and which are not, from a customer's point of view', and therefore is contributive to improvement of software product and release management processes.

2.6 CASE STUDY RESULTS

Case study results are presented per case study participant. Since all participating software vendors conduct software development activities in European countries, we consider the case study results representative for similar-sized European software vendors at the minimum; case study results might be generalizable to vendors operating in non-European countries. We regard the research as repeatable with the same results, presuming similar circumstances (similar interviewees, similar explanatory sessions, etc.). Note that for reasons of confidentiality, the names of all case study participants and their software products and services have been anonymized.

2.6.1 Wareex

Wareex develops business software for small, medium-sized and large enterprises. In addition to ERP software, Wareex develops HRM, CRM, project and workflow solutions. With 2,500 employees and establishments in 40 countries, the vendor is established on six continents. Wareex was founded in 1984 and serves customers in 125 different countries. During the case study, two product managers, three research engineers, two software architects and one customer support manager were interviewed. All interviewees considered the framework sound; no alternative processes, stakeholders or flows were suggested. The interviewees could easily project the vendor's situation in terms of SOK on the SOK framework. Wareex has implemented processes of the SOK life cycle for several products and services of its software portfolio. First, Lobeex is an off-line desktop software product that facilitates one integrated back office implementation for multiple business processes. Concerning Lobeex, Wareex has rudimentarily implemented a subset of the SOK life cycle processes presented in section 2.3.2. Wareex's principle research engineer *identified* performance knowledge (κ_P) as most relevant and valuable type of SOK: due to the size and complexity of some back office administrations, a significant part of Lobeex support calls are related to performance of its software. Wareex built a

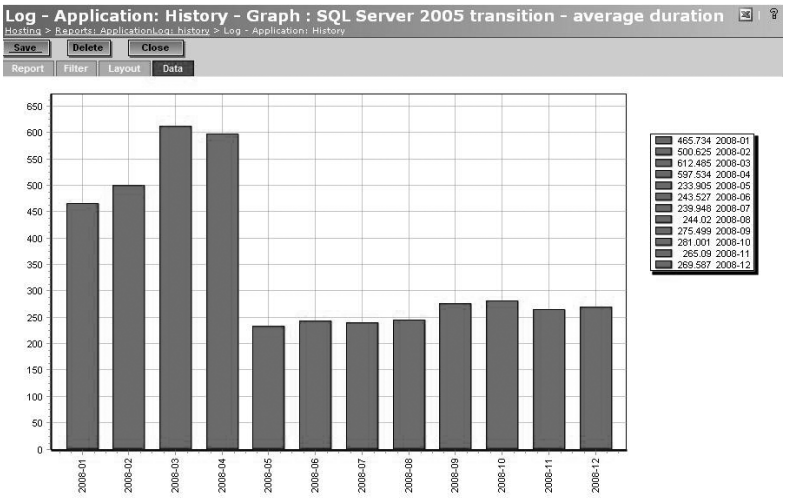


Figure 2.2 Performance graph of Wareex's software operation dashboard, showing the impact of a database version change on Lineex performance

Lobeex operation knowledge acquisition tool that is installed separately from the Lobeex software. The tool *acquires* non-sensitive customer data, such as hardware, memory and operating system details, database statistics as well as SQL query performance data. Neither the tool nor the data it acquires is *integrated* with other tools or processes. The data are *presented* in ad-hoc generated performance analysis reports, triggered by requests from the product manager or the support department. Wareex's principle research engineer as well as Wareex's Lobeex product manager indicated that the acquired software performance data are *utilized* in software maintenance and customer support processes, and is used to detect and repair query performance problems and to optimize database index schemes. Also, Wareex uses Lobeex operation knowledge to propose new hardware or database configuration prospects to its end-users.

A second product that is part of Wareex's software portfolio is Lineex. Lineex is an accounting solution provided as a secure online web application. Regarding Lineex, Wareex has developed specific SOK acquisition and presentation tools. While SOK *identification* occurs ad-hoc, triggered by occurrence of concrete problems, software operation data are *acquired* and converted into SOK automatically. The data are stored by the application's base layer (on which all application functionality is based). The acquired operation data are stored

in four logs: an application log (containing software usage and performance data such as HTTP requests, page view sequences and response times), an error log (containing software quality data such as unhandled exceptions, query timeouts and other errors), a help log (containing software feedback data like end-user help page feedback and appreciation) and a process log (containing response times and performance statistics of background processes). The data are not (yet) *integrated* with external tools, so for example bugs causing unhandled exceptions are still manually entered into Wareex's bug repository. One of the Wareex software engineers is entrusted with analysis of all logs. He decides which log entries are relevant and undertakes immediate action when required. Wareex has developed comprehensive SOK presentation software, which is used daily by developers, product managers and support assistants to monitor the load of background processes, look into recent software usage history and inspect error messages and corresponding exceptions. Lineex's SOK is *presented* via an online software operation dashboard, which provides detailed software performance, quality, usage and feedback data: figure 2.2 shows a graph that visualizes the impact of a transition from Microsoft SQL Server 2000 to 2005 on average query duration. Software developers, product managers and project leaders indicated that Wareex *utilizes* Lineex operation knowledge to improve software development, maintenance and release management processes.

One product manager noted that 'the SOK identification process determines the complexity of the subsequent processes': the effort needed to acquire, integrate, present and utilize software operation knowledge is to a large extent determined by the operation knowledge demands defined in the SOK identification process. As described in section 2.3.2, logic defined in the identification process determines the extent to which data explosion is prevented. Also, another product manager as well as the software architects indicated that a SOK integration process is not implemented at Wareex: acquired data are directly presented in logs and reports. They mentioned that at Wareex, SOK is utilized to detect, identify and fix software problems faster and therewith improve software quality. The support manager indicated that SOK would be useful for support assistants to better understand a calling customer and its situation. Regarding future SOK developments, a product manager stated that by means of data mining techniques, Wareex foresees to automatically distinguish customer profiles and usage trends from acquired SOK.

The research engineers, a software architect and a product manager mentioned that it is a goal to continuously determine and quantify the thresholds that separate 'bad' situations from 'good' situations regarding the state of soft-

ware in the field. This was found particularly relevant in the light of software scalability issues: the aforementioned interviewees found that changing circumstances (number of end-users, software updates, hardware environments) cause new issues which are hard to predict and quantify. The interviewees found it challenging to objectively acquire and prioritize SOK, and asked questions such as 'when is good software good enough?' and 'to which extent does a software vendor contribute to the "badness" of its software?'. One software architect envisaged self-repairing or self-recovering software, but added that realization of such software would be difficult since 'causes of problems and failures are not always automatically traceable or distinguishable'.

2.6.2 Sionag

Sionag is specialized in development of software for the agricultural sector. The vendor, founded in 1985, serves thousands of customers in 22 countries with 20,000 licenses in total. The vendor is established in Europe and employs 100 people, of which 20 are software engineers. Two product managers, two senior software analysts, one software engineer, one database administrator and one support assistant were interviewed. While no framework elements were rejected or new elements were suggested, interviewees noted that the framework visualizes an 'ideal situation' that is not completely representative for the situation at their organization. 'Currently, we are acquiring operation data and planning to implement new data mining techniques', one product manager stated. Next, the manager noted that he expected software operation information to be initially supportive from a development perspective, in terms of time and cost savings. For example, he expected their software maintenance and release management processes to be improved by means of acquired SOK. The manager expected acquired software operation information to be secondarily supportive from a customer perspective (in terms of customer intimacy improvement) and to be supportive from a company perspective in the long term.

Sionag has started SOK *identification* and *acquisition* for one of its software products, Eropt. Eropt is used to advise animal food compositions. When an optimal diet is composed at a farm, all nutrition data are synchronized. Eropt connects with a synchronization web service hosted by Sionag, which provides access to Sionag's main nutrition database. Synchronization is realized by means of XML SOAP messages, which Sionag utilizes to acquire SOK: apart from updated nutrition data, software operation data acquired by Eropt the last synchronization is sent to Sionag. These data consist of recent usage details, customer and agent identification data, exception data (error mes-

sage, stack traces), hardware and system details and Eropt version information. While acquired data are not explicitly *integrated* with other tools or processes, Sionag software engineers have developed a tool, Ayopt, to *present* and analyze acquired software operation data. With Ayopt, a farm's synchronization and usage history can be analyzed, for example. As recognized by the software analyst, the data synchronization process is critical to the success of Eropt, since synchronization errors (concurrency violations caused by deleted nutrition data, for example) imply loss of crucial data and re-do of two days of work. The engineer and one software analyst indicated that SOK (κ_P and κ_Q in particular) is *utilized* to reproduce software failures and quickly find bugs, therewith speeding up software maintenance processes and increasing the robustness and usability of the software. Concerning Sionag's future SOK developments, one software analyst and one product manager indicated that an online, service-based version of Eropt will be developed in order to eliminate the need to explicitly synchronize nutrition data and to be able to apply data mining techniques to acquired operation data more easily.

2.6.3 Ansta

Ansta is a European software vendor that was founded in 1990. The vendor develops an industrial drawing application, Adsta, which is targeted on the Microsoft Windows platform and is used daily by more than 4,000 customers in five countries. Since the start of its development in 1995, four major versions of the application have been released. Currently, Ansta employs 100 people and is performing development activities in the Netherlands, Belgium and Romania.

During the case study, the SOK framework was discussed with Ansta's CEO, software development manager and marketing manager. In general, the framework was considered sound. However, the marketing manager suggested to add a block 'directed marketing' to the SOK utilization process in the company perspective. He indicated that acquired SOK, whether or not integrated with external tools or presented on various media, could be used to direct the company's marketing, for example to highlight certain features of the software that are highly appreciated by a significant part of the vendor's customer base. The CEO and managers indicated that successful utilization of SOK could, in the long term, result in higher quality release plans, increased customer intimacy and improved knowledge building of software developers, trainers and supporters. They also stated that, of all SOK types, they found software usage knowledge (κ_U) and software performance knowledge κ_P to be the most contributive and valuable in the context SOK utilization at Ansta. Furthermore, it

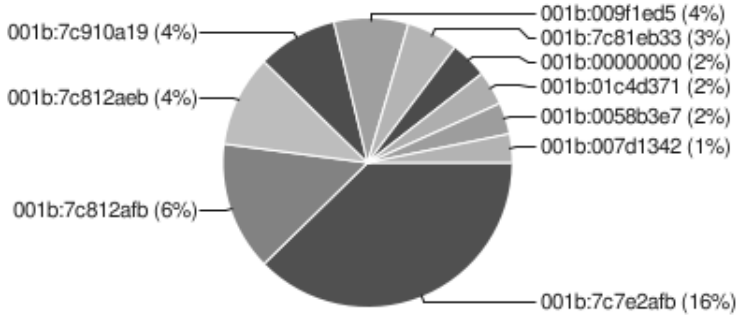


Figure 2.3 A graph from Ansta's SOK presentation tool, Denerr, showing the top 10 crash memory locations based on submitted error reports. 16% of the submissions report errors on one and the same memory location

was mentioned that in order to successfully utilize SOK, SOK should actually be analyzed and combined with other data (e.g. mailing conversion statistics, license data, etc.): the marketing manager expressed that integration of SOK with the workflows, processes and tools used by employees could significantly contribute to the effectiveness of SOK utilization.

Like Sionag, Ansta has implemented SOK *identification* and *acquisition* processes for its drawing application Adsta. Ansta has realized an infrastructure for assembly and acquirement of software performance and quality knowledge in the form of error reports. In the case of an unhandled exception, Adsta shows an error dialog that offers users the option to send an error report to Ansta, and provides functionality for end-users to determine which information is contained in the report. Potentially, an Adsta error report consists of (1) a crash log, containing exception details, as well as hardware and software environment data (processor type, amount of installed memory, operating system version, current user, etc.), (2) a crash dump consisting of the recorded state of Adsta's working memory at the time it crashed, and (3) a registry file containg Adsta's Windows registry settings.

To acquire error reports sent by end-users, Ansta has implemented a web service to which the reports are submitted. Via this web service, Adsta error reports are stored in a database. Ansta developed an intranet application called Denerr (see figure 2.3), by which all error reports are *presented* to all of the vendor's employees. Denerr provides functionality to search for and select error reports that meet certain search criteria (regarding time, contents, source, etc.). Also, error reports can be downloaded for further analysis.

Currently, Ansta is implementing tighter *integration* of acquired SOK with its processes and tools. For instance, the vendor is developing a tool that enables mapping of error report crash dumps to source code files line numbers. By integrating this tool with its Denerr application, the vendor expects to pinpoint software failures faster and more accurately. Also, Ansta plans to increase SOK *utilization* by adding report generation functionality to Denner.

2.6.4 Summary

The results of the case studies performed at Wareex, Sionag and Ansta can be summarized as follows:

(1) Demand:
product software vendors lack a long-term vision regarding SOK utilization and are in need of a guiding substrate that aids in establishing that vision;

(2) Utility:
the SOK framework is considered useful: vendors gained insight by mapping their practices, processes and tools onto the framework; and

(3) State of practice:
while vendors have identified which SOK types they consider valuable and have implemented SOK acquisition processes by means of specific software operation logging or monitoring tools, acquired SOK is not (yet) integrated or utilized with processes and tools already in place. Software vendors indicate and acknowledge that tight integration of acquired SOK contributes to mature SOK utilization.

2.7 THREATS TO VALIDITY

The validity of the research results is threatened by several factors. A primary threat to the validity of the questionnaire results is the number of subjects. Due to the small number of subjects, (differences between) questionnaire results are not statistically significant. However, taking into account the role of the subjects in their organizations as well as the variety in organizations, we consider the questionnaire results indicative and representative. Of the validity criteria for empirical research defined by Yin [Yin 2009] and others, the external validity of our case study research is threatened by the number of case studies carried out. While we believe that the case study results of the three multinational software vendors are typical for European vendors of similar size, results might be less

applicable to smaller software vendors. Further empirical research is needed to mitigate these threats. In this research, the selection of case study participants was pragmatic; we plan to perform case studies at software vendors that are more mature in terms of SOK utilization in the future.

2.8 CONCLUSIONS AND FUTURE WORK

All too often in software engineering, software and end-user feedback are overlooked as instruments to guide and advance a software vendor's activities. In this paper, software operation knowledge is presented to unify existing definitions of knowledge of in-the-field software operation, and a framework is proposed that is designed to aid software vendors in gaining insight in both the life cycle of such knowledge, as well as in product software perspectives from which processes of this life cycle can be perceived. Based on the results of our empirical evaluation approach, we conclude that the SOK definition is complete in terms of knowledge types, and the SOK framework is sound and useful. The framework aids vendors in determining next steps in terms of their path to effective SOK utilization.

Although software vendors consider SOK valuable, integration with existing activities and infrastructure is missing. Case study results show that while software vendors have implemented several processes of the SOK life cycle defined by the framework (i.e., identification, acquisition, integration, presentation and utilization), acquired SOK is rarely integrated with (tools to support) vendors' existing practices and processes. As a result, SOK is used ad hoc and software engineering processes still advance only modestly as a result of SOK utilization.

Through the questionnaire answers, we found that of each software operation knowledge type $k \in \kappa$, utilization of end-user feedback knowledge (κ_F) is expected to contribute to improvement of software engineering processes the most. Such knowledge can be used to challenge software engineering and SOK practice assumptions, and to enhance future SOK acquisition, integration and presentation tools. Future research plans include development and validation of SOK acquisition tools, such as a tool for generic recording of in-the-field software operation. Also, analysis of the (potential) role of SOK integration, presentation and utilization in software vendor organizations will be subject of future work.

3

On the Role of Software Operation Knowledge within Software Ecosystems

ABSTRACT

Knowledge of in-the-field software operation is still unrecognized as an essential pulse in the veins of software ecosystems. Although software-producing organizations are aware of the ecosystems in which they operate and their relationships with other ecosystem participants, all too often, vendors are unsuccessful in recognizing the potential value and role of such knowledge in their software ecosystems. This paper presents a classification of successful operational software ecosystem practices that may help software-producing organizations to effectively utilize and propagate knowledge of the in-the-field operation of their software, and therewith address challenges that result from ecosystem participation. Analysis of these practices confirms that infrastructures for acquisition, utilization and propagation of such knowledge, allow ecosystem participants to use the 'power of many' in increasing the quality and robustness of their software, and provide them with competitive advantage in terms of software quality, end-user satisfaction, ecosystem stability and ecosystem attractiveness.*

*This work has been published as *The Power of Propagation: On the Role of Software Operation Knowledge within Software Ecosystems* in the proceedings of the International Conference on Management of Emergent Digital EcoSystems (MEDES 2011) [Van der Schuur *et al.* 2011d]. It is co-authored by Slinger Jansen and Sjaak Brinkkemper.

3.1 INTRODUCTION

Software vendors have become networked organizations that are dependent on other software vendors for libraries, components or platforms vital to the correct operation of their software. Due to rapidly changing technology and end-user demands, software vendors have resorted to virtual integration through alliances in or between the software ecosystems in which they operate. While many software vendors thrive and advance because of their participation in software ecosystems, these vendors are simultaneously confronted with several challenges resulting from software ecosystem participation, such as software ecosystem relationship management, ecosystem orchestration strategy development, and ecosystem composition comprehension [Farbey and Finkelstein 1999, Bosch 2009, Van Angeren *et al.* 2011]. Part of the value that is created in software ecosystems is software operation knowledge (SOK): 'Knowledge of in-the-field performance, quality and usage of software, and knowledge of in-the-field end-user software experience feedback' [Van der Schuur *et al.* 2010]. Vendors use SOK, for instance, to gain insight in the in-the-field behavior of (end-users on) their software, increase software quality by mitigating issues that may surface only after deployment (e.g. compatibility issues, performance problems, etc.) and improve software processes like software maintenance, software product management and customer support [Van der Schuur *et al.* 2010].

In this paper, we state that SOK forms an essential pulse in the veins of software ecosystems, and serves as a primary enabler for software vendors to thrive and flourish in the ecosystems in which they operate. Software vendors can successfully participate in software ecosystems by addressing challenges resulting from this participation through effective utilization[1] of acquired software operation knowledge, as well as through propagation of such knowledge to other ecosystem participants. To substantiate our position, we report on the operational SOK propagation practices of four software-producing organizations, and show how these organizations have mitigated aforementioned challenges through SOK propagation. Analysis of these practices shows that through effective utilization and propagation of SOK, software quality is increased, relations between vendors are strengthened, and insight into software ecosystem composition is deepened.

This paper continues as follows. In the next section, we elaborate on the concept of SOK propagation. First, we identify two main software ecosystem par-

[1] In this paper, we define 'utilization' of software operation knowledge as the use of such knowledge to support or improve processes, practices or products.

SECO Scope Level	SOK Propagation Illustration
Software Ecosystem	Changes to the keystone platform (e.g. in-the-field API behavior) are propagated to all ecosystem participants
Actor	Knowledge of keystone software instability, caused by niche player software, is propagated to that particular niche player

Table 3.1 SOK propagation illustrated for two SECO scope levels

ticipant roles and describe SOK propagation characteristics per role. Second, we describe challenges that result from software ecosystem participation, and illustrate how these challenges can be addressed through SOK propagation. Section 3.4 presents the identified operational SOK propagation practices of four software-producing organizations (the identification approach is detailed in section 3.3); analysis of those practices is provided in section 3.5. Finally, conclusions are presented in section 3.6.

3.2 SOK PROPAGATION WITHIN SOFTWARE ECOSYSTEMS

Several different definitions of the term software ecosystem (SECO) exist [Messerschmitt and Szyperski 2003, Bosch and Bosch-Sijtsema 2010, Kittlaus and Clough 2009]. In this paper, we use the following definition of a SECO: 'A set of actors functioning as a unit and interacting with a shared market for software and services, together with the relationships among them' [Jansen *et al.* 2009]. Relationships between SECO actors are frequently underpinned by a common technological platform or market, and operate through exchange of information, resources or artifacts [Jansen *et al.* 2009, Bosch and Bosch-Sijtsema 2010]. Participation of actors in software ecosystems can be observed from various points of view. In this paper, we scope participation of software vendors in software ecosystems from two scope levels, both levels corresponding to certain 'objects of study' [Jansen *et al.* 2009]: software ecosystems as a whole, and software ecosystem actors itself (see table 3.1). Companies participating in a software ecosystem (i.e., interacting with a shared market for software and services) can have different roles. Although more roles are identified, the roles of *keystone* (i.e. orchestrator, shaper) and *niche player* (i.e. follower) are recurring in literature, e.g. [Jansen *et al.* 2009, Häcki and Lighton 2001].

Keystone

An actor providing a standards or technology platform that forms a foundation for the keystone's ecosystem. The platform defines standards and

practices, provides leverage and increases in value and attractiveness with the number of actors. Keystones create and share value (i.e. assets, incentives) within their software ecosystems [Iansiti and Levien 2004a], and aim to improve their ecosystems' quality, stability and productivity by (1) encouraging specialization (i.e. niche creation), (2) cultivating deep understanding of processes and practices and (3) developing and managing feedback loops [Iansiti and Levien 2004b, Brown *et al.* 2002, Gawer and Cusumano 2002].

Niche player

An actor that requires a standard or technology platform provided by a keystone to create business value [Jansen *et al.* 2009]. A niche player operates in a niche market and prospers through value or knowledge generated by the keystone, other actors or the software ecosystem as a whole [Häcki and Lighton 2001, Brown *et al.* 2002]. Niche players are invited to participate in software ecosystems by the ecosystems' keystones (for instance by receiving a development program qualification or software compatibility certification, or by having (unrestricted) access to keystone resources), and therewith may become keystone partners on the long term.

Based on operation knowledge demands, software vendors obtain software operation knowledge and operation information from software operation *data* (as demonstrated in [Van der Schuur *et al.* 2010, Van der Schuur *et al.* 2008, Van der Schuur *et al.* 2011b]). Figure 3.1 conceptually visualizes SOK utilization and propagation in software ecosystems. On the actor level, actors (e.g. keystones, niche players) derive software operation knowledge from (behavior of end-users on) software operating in the field and utilize such knowledge within their organizations. On the software ecosystem level, such software operation knowledge is propagated to other actors in the same ecosystem. After application of data mining techniques and data abstraction logic, both based on operation knowledge demands, operation *information* is obtained. This information serves as input for support and improvement of an actor's the products or processes. Interpretation and integration of operation information results in software operation *knowledge*, which is utilized iteratively and continuously to improve an actor's products and processes.

SOK utilization can serve various goals and can thus be viewed from various perspectives, such as a development, company and customer perspective [Van der Schuur *et al.* 2010]. As a result of their utilization of software operation knowledge, software ecosystem participants may respond by (1) con-

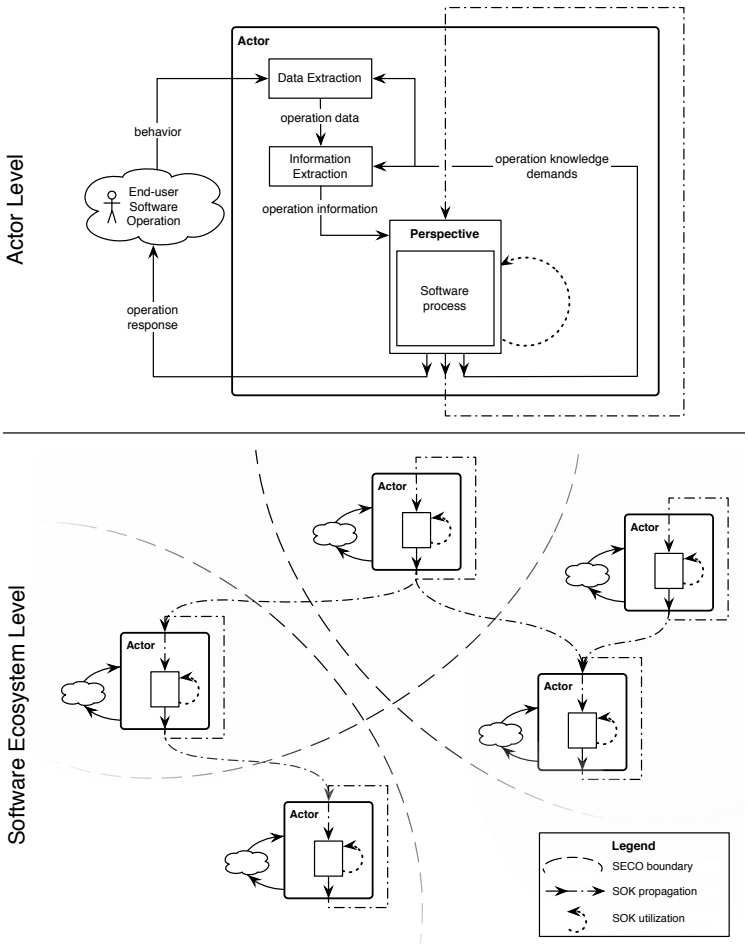


Figure 3.1 SOK utilization and propagation in software ecosystems

tacting customers and releasing software patches or updates to eliminate or thoroughly investigate software failures, (2) defining new (or altering existing) operation knowledge demands for more generic or specific operation data acquisition, and (3) propagating (utilized) software operation knowledge to other ecosystem participants, for example to notify partners of unstable or failing operation situations, or provide additional operation environment details to encourage bug elimination and software quality improvement. As detailed in the next section, an ecosystem participant may utilize SOK propagated by other ecosystem participants in addition to SOK it obtains from in-the-field operation of its 'own' software.

3.2.1 Propagation Characteristics

Propagation of software operation knowledge within software ecosystems encompasses both scope levels in table 3.1. Although software operation knowledge is propagated by one actor to one or more other actors, the process of propagation itself can be observed from both scope levels: actors as well as software ecosystems as a whole may be influenced by, or accountable for, SOK propagation. SOK propagation characteristics and goals may differ per actor role, however.

Keystone characteristics

In terms of SOK propagation, keystones set about establishing a platform for publication and propagation of acquired SOK (e.g. error reports, performance measures, etc.) to niche players in their software ecosystem [Häcki and Lighton 2001]. Keystones orchestrate which niche players have access to the platform and to which niche players knowledge of keystone software operation is propagated. Keystones are able to propagate SOK through their ecosystem at different levels of granularity, corresponding to the scope levels in table 3.1.

Keystones strive for reinforcement and growth of their software ecosystem. Software operation knowledge can be used to do so. For example, effective propagation of, and response to SOK acquired during operation of a new set of platform APIs can increase the performance and quality of these APIs rapidly, and therewith accelerate their adoption. External actors, potentially participating in rival ecosystems, may be attracted by the stability, feature-richness and 'partner intimacy' of the keystone and its platform.

With over 425,000 software applications, downloaded more than 15 billion times in total in less than 3 years, Apple's AppStore is recognized as the platform that thrives the iPhone software ecosystem. In the first months after the launch of the store, Apple prohibited public discussion about APIs, documentation and sample code. Particularly during these months, incorrect usage of (unofficial) iPhone SDK APIs led to many application crashes (for instance after installation of an operating system upgrade) and much frustration of niche players participating in Apple's iPhone ecosystem. Compatibility and stability issues caused by erroneous API use could have been prevented through SOK analysis and propagation.

Niche player characteristics

On the short term, niche players strive to create business value through the platform provided by the keystone. For instance, niche players create such value by (1) developing niche software products, (2) realizing efficiencies resulting from the sharing of assets and knowledge by other ecosystem participants, and (3) obtaining access to the keystone's customers [Häcki and Lighton 2001]. Adequate response to SOK received from keystones, as well as frequent informing of keystones on end-user feedback knowledge (end-user experiences, wishes, etc.) may be rewarded by a keystone with 'niche player partner' qualification (which provides access to a keystone's customer data, for example). While close and trustful relationships and communication with the keystone and other niche players may be advantageous, niche players are prudent and attempt to prevent overdone response (e.g. by sharing specialized knowledge or intellectual property). Niche players seek a SOK propagation response balance: too comprehensive response may reduce a niche player's competitive or knowledgeable advantage, too limited response might not result in quality and robust software and consequent overdue response may result in exclusion from the ecosystem or forfeiture of partnership, which diminishes a niche player's chances to become a keystone itself. Nonetheless, to transform into a keystone actor, niche players can merge forces or gradually adopt keystone skills.

At present, certified niche players participating in the Microsoft Windows ecosystem receive SOK propagated by Microsoft via its Windows Quality Online Services (Winqual) [WinQual, Glerum *et al.* 2009]. Niche players can link software operation failures to solutions, which allows directed distribution of software updates and corresponding support web pages to customers. Furthermore, Microsoft propagates knowledge of end-user satisfaction, and software usability and appreciation to niche players through its program for customer experience improvement [CEIP].

3.2.2 Addressing SECO Challenges through SOK Propagation

Software vendors participating in software ecosystems are faced with new challenges on each software ecosystem scope level [Jansen *et al.* 2009, Farbey and Finkelstein 1999, Bosch 2009]. Below, we describe how we envision particular challenges to be addressed through effective utilization and propagation of SOK.

SECO relationship management

Evaluation of propagated SOK results in intensification, prolongation or diminishing of relations between software ecosystem participants and their suppliers, buyers and end-users. For example, a keystone may decide to promote a niche player to a (certified) partner when the niche player responds adequately to propagated SOK and its software is robust and of high quality. New relationships are established, for instance, if SOK evaluation shows that a particular niche player's software does not meet quality level agreements and a keystone decides to replace the niche player by a new vendor, or if the keystone platform is to be extended with novel standards or techniques developed by vendors not participating in the keystone's ecosystem.

SECO composition comprehension

When SOK is continuously and forthrightly propagated through a software ecosystem, participants gain insight in the structure of and knowledge flows within their ecosystem. As an example, a keystone actor may obtain a comprehensive view of the most-used software add-ons that are operating on top of its extension architecture. Niche players get insight in which ecosystem participants are propagating the greatest amount of SOK.

Software quality management

As stated earlier, acquired SOK is used as a basis for monitoring and managing quality of software operation. Also, keystones define quality levels and analyze SOK to determine a niche player's software add-on quality level and optionally propagate SOK to increase add-on compatibility or operation quality. Niche players attempt to acquire certified partnerships by acquiring, monitoring and propagating SOK themselves, as well as by continuously responding adequate to received SOK.

Portfolio and product line planning

Ecosystem participants are supported by acquired or received SOK in deciding which new or refined functionality is included in which future versions of their software. SOK analysis may unveil which features are used often, appreciated or wanted most, or cause crashes at the majority of end-users in the field, by which, for instance, release planning activities are supported.

SECO orchestration strategy and policy development

Keystones utilize SOK for developing or perfecting their SECO orchestration strategies and policies, to increase attractiveness of their ecosystem. For example, if many software applications based on the keystone's platform are crashing, SOK analysis can be performed to identify most-occurring software failures. Next, based on SOK analysis results, a keystone can (re)define quality level agreements and compatibility certifications to increase the platform's robustness, and therewith increase the attractiveness of the ecosystem compared to rival ecosystems.

3.3 PRACTICE IDENTIFICATION APPROACH

To investigate the role of SOK in software ecosystems and to empirically evaluate how and to which extent product software vendors have addressed the challenges described in section 3.2.2, we conducted extensive semi-structured interviews with in total seven employees of four software-producing organizations that are participating in one or more software ecosystems (see table 3.2). The interviews consisted of 41 questions that were formulated after literature study, divided over five sections[2]. Interview sessions took 1.5 hour on average and were conducted on-site, or via conference calling, over a total period of 35 days. Organizations were contacted from industrial networks. Basis for contacting organizations for interviews was the extent to which an organization utilizes SOK, and propagates SOK within the software ecosystems it operates in.

Interview answers were recorded and used to determine to which extent, and, if applicable, how organizations addressed ecosystem challenges through utilization and propagation of SOK. Having clustered the organizations per challenge they addressed, we subsequently derived aspects that are common to these organizations, and could be generalizable to similar organizations that have addressed the same challenge. Derivation of these aspects (i.e. *Key derivations*) was conducted from the perspective of our position stated in this paper. Key derivations were found during inductive and deductive iterations of interview answers study. The key derivations form the basis for the classification of orchestration and propagation SOK practices in section 3.5. As the future intensions in section 3.4.6 are not yet operational, we deliberately omitted those from the classification.

[2] All questions are available at http://people.cs.uu.nl/schuurhw/sokpropagation

	OSComp	NetComp	ERPComp	CADIntComp
SECO platform	OS-, office- and development platforms	Net	ERPOffline, ERPOnline	CadInt
Platform technology	.NET	Various	.NET	.NET
SOK propagation infrastructure	Quality service, error reporting	Net hardware	OSComp Quality service	Error reporting, member portal
SECO participation incentive	Logo program	Partner alliance program	Developer key	Consortium membership
Number of employees	80,000	67,000	2,100	25[a]
Number of interviewees (role; avg. years of IT employment)	1 (principal researcher; 28)	1 (CIO Europe; 23)	3 (technology research director, product line manager, principal research engineer; 15)	2 (development director, quality assurance manager; 24.5)

[a] Excluding employees of consortium members

Table 3.2 SECO characteristics of the four software-producing organizations researched

3.4 IDENTIFIED SOK PROPAGATION PRACTICES

After having introduced the four vendors, we describe for each challenge presented in section 3.2.2 how two or more vendors address these challenges through SOK utilization and propagation. Note that for reasons of objective comparability, company names as well as names of software products and techniques have been anonymized.

OSComp

OSComp currently employs 80,000 people of which one-third is performing software engineering activities. The ecosystems around OSComp's operating system, office and software development products are only a few examples of ecosystems in which OSComp fulfills a keystone role. Also, the company is a niche player actor in Apple Mac OS X and iPhone software ecosystems. OSComp provides technologies to enable niche players to develop desktop applications, web applications and extensions. The company acquires knowledge of its in-the-field software operation by means of its Error Reporting (ER) technology, for acquisition of crash and failure data (excluding operation environment data), and the Analysis Component (AC) technology for collecting software operation data (including operation environment data) of particular customers.

NetComp

NetComp currently employs 67,000 people, of which one-third are software engineers developing communication, collaboration and operating system software at 24 locations world-wide. The company is the keystone in the software ecosystem around its Net software that is running on the vast majority of NetComp's hardware. Niche players in this ecosystem are partners of NetComp, and contribute to new releases of the Net software. With keystones as Nokia and OSComp, NetComp considers itself a niche player in the ecosystem of collaboration and conferencing software: the company develops plugins for end device software platforms to enable collaboration and conferencing on the corresponding devices. Knowledge of operation of NetComp software on idem hardware is acquired by NetComp only with accordance of its customers.

ERPComp

ERPComp is a global ERP software vendor with 2,100 employees (250 software engineers) that fulfills different roles in several ecosystems. Its main product portfolio consists of an offline desktop application, ERPComp ERPOffline, and an online, service-based web application called

ERPComp ERPOnline. With both applications based on OSComp technology, ERPComp is in the top 50–100 of OSComp's global independent software vendors team and is therefore a partner niche player in OSComp ecosystems. ERPComp participates as a keystone in its dealer and partner networks: the vendor has initiated technology platforms with SDKs or only APIs to enable dealers to develop niche add-ons for its ERPOffline product, or to allow tighter integration and collaboration between its ERPOnline product and niche players like banks or governmental tax institutes.

CADIntComp

CADIntComp is an organization that develops a Computer Aided Design (CAD) platform, and is owned and governed by its members. Members of CADIntComp integrate the main technology of this platform, CADInt, in their own software products and customize it to serve niche engineering markets. In turn, CADIntComp uses .NET technology to build CADInt. The CADIntComp consortium consists of about 50 members from 38 countries, mostly software vendors, which form the niche players in the software ecosystem of which CADIntComp is the keystone. Niche players in the CADInt ecosystem may be keystones in other software ecosystems. For example, one of CADIntComp members integrates the CADInt technology in its CAD product software for building industry. Simultaneously, the vendor has built a platform for manufacturers of engineering materials to provide and publish CAD software drawing symbols that are used in its software for drawing these materials. Two types of niche players participate in the CADInt ecosystem: commercial members (which have full source code access and may contribute to the platform) and API members (which only have access to APIs).

3.4.1 SECO Relationship Management

OSComp

Niche players can enter most of OSComp's software ecosystems by producing software for the corresponding platforms. The company attempts to intensify relationships with niche players by stimulating them to become (certified) partners through certification and compatibility programs. For example, by means of the Quality service that is part of OSComp's Logo Program, the vendor verifies compatible operation of hardware and software of certified niche players with most recent versions of its own

software (e.g., the operating system software). Also, the company propagates knowledge of software operation failures to niche players certified for program participation.

NetComp

Companies are allowed to participate in NetComp's ecosystems through the NetComp partner program, where the volume of the potential niche player as well as partnership capacity available at NetComp determine wether or not a niche player is recognized as an official partner. Knowledge of in-the-field operation of niche player software is used by NetComp to promote or acquire niche players. Also, relationships with niche players are diminished or discontinued when acquired software operation data indicates defective niche player software. Strategic alliances are formed with corporations like IBM and AT&T, players with which NetComp interchanges engineers and to which NetComp discloses virtually all of its intellectual property.

ERPComp

Niche players are invited to ERPComp's ecosystems by obtaining an SDK license (which, for instance, allows developers to add new functionality and extend data models) or registering for an API developer key (which enables developers only to import data from and export data to the ERPComp software). ERPComp stimulates its niche players to develop software for distinct niche markets, or to collaborate with other niche players by developing a joint add-on (for instance when too many players are developing equivalent add-ons for one and the same niche market).

KEY DERIVATIONS Keystones provide different incentives for stimulating niche players to participate in their software ecosystem, and therewith initiate SOK acquisition from niche player software. Selection of incentives is based on the role and behavior of niche players as well as on the keystone's capacity and resources available for supporting the niche player. Keystones attempt to initiate niche player relationships via developer keys, after which relations are strengthened through certification programs and (private) SDK access. SOK resulting from *monitoring and verification of software operation compatibility* is used as input for relationship management. Ultimately, keystones merge niche player partnerships or form strategic alliances with niche players.

3.4.2 SECO Composition Comprehension

OSComp

According to OSComp, software ecosystems compositions are continuously changing. Therefore, effective SOK acquisition is practically the only way for the vendor to gain and maintain insight in the ecosystems in which it is operating. For example, if a niche player joins the OS ecosystem and releases a new hardware driver, OSComp can be aware of it in a matter of days. Also, received SOK provides OSComp with insight in the actual distribution of (particular versions of) its software over its software ecosystems.

NetComp

NetComp is legally obliged to register what are the physical operation locations of its network products, so that it can prove that its hardware is not placed illegally at certain locations or in particular countries. In addition to knowledge of *how* Net is operating in the field (e.g. hardware uptime, number of unexpected shutdowns, request/response failure rates, etc.), knowledge of *where* Net hardware is currently operating, is part of Net operation knowledge. Hence, SOK is vital to NedComp for monitoring the geographical location of its hardware, and therewith to NetComp's comprehension of its Net ecosystem.

KEY DERIVATIONS Keystones utilize SOK to gain and maintain insight in their ecosystems' compositions. SOK is not only used to *monitor ecosystem entrance and exit of niche players and their software*, but also to keep track of the physical location of the hardware on which the software is operating. Vendors may be (legally, or otherwise) constrained to register such ecosystem composition information.

3.4.3 Software Quality Management

OSComp

OSComp's ER service has helped to improve the quality of many niche player device drivers, which caused a majority of its OS crashes in the past, and alleviated the demanding challenge of isolating obscure Heisenbugs (bugs that disappear or alter when an attempt is made to study it) [Gray 1986]. According to OSComp, it is impossible to extensively test all unique devices, let alone unique device configurations. As an attempt to nonetheless maintain and manage the quality of its software, the com-

pany uses its ER service for error report diagnosis, statistics-based debugging and automatic direction of end-users to solutions of corrected software operation failures. Since 2009, more than 700 ecosystem participants with roughly 7000 applications were using the ER technology, of which 175 participated with registering almost 1500 solutions to hardware- or software operation failures.

NetComp

Knowledge of in-the-field Net operation is used to improve the quality of NetComp's software. When operation data indicate software failures, regression analysts are asked to determine the 'real', underlying cause of the failure by mining acquired operation data, after which the bug that causes the failure is solved and the hardware involved is remotely updated with the most recent version of the software. With respect to software quality management, NetComp's goal is to be the first to identify and address failures of its software.

ERPComp

ERPComp has certified its ERPOffline product for OSComp's Logo Program, and SOK originating from in-the-field ERPOffline operation is propagated to ERPComp by means of OSComp's Quality service. Since error reports created with this service provide ERPComp insufficient data to identify the exact cause of software failure, ERPComp has developed an analysis tool that end-users can download to determine the exact problem source. When ERPOffline operation failure is caused by a niche player add-on, the niche player is contacted by ERPComp to ensure the issue is solved adequately. Concerning operation of ERPOnline, ERPComp acquires and monitors performance, quality, usage and feedback data during operation of the ERPOnline web application to quickly respond to performance issues and answer usage trend questions (e.g. regarding Internet Explorer 6 usage). The vendor is able to identify niche players responsible for malformed API requests or responses, by filtering operation data on the corresponding developer key. ERPComp propagates SOK to partner niche players, for instance to evaluate and improve operation and compatibility of couplings between its ERPOnline application and partner services.

CADIntComp

Within CADIntComp's CADInt ecosystem, SOK propagation contributes to improvement of software quality and reproduction of rare bugs. CAD-

IntComp has built a member portal which is vital to the propagation of CADInt operation knowledge through its ecosystem. Apart from patches, language translations and feature requests, members submit crash logs containing a mini memory dump, call stack as well as other software operation environment details to the portal. Crash logs are mined and analyzed, after which CADIntComp quality assurance employees are notified if critical information has been submitted.

KEY DERIVATIONS Vendors use SOK to manage quality of all kinds of software. It is clear that adequate SOK propagation is considered particularly advantageous in *determining the root cause of in-the-field software failures* that do not occur at the software vendor site, or in isolating Heisenbugs [Gray 1986]. The more vendors are able to acquire and effectively utilize knowledge of software operation, the more these vendors are successful in managing quality of their software.

3.4.4 Portfolio and Product Line Planning

OSComp

Within OSComp, SOK is used in every phase of the software life cycle. Roadmaps of new software products are based on the usage and quality of the most recently released software product as well as SOK acquired from the last product. For example, a new user interface introduced by OSComp in its office software is largely based on SOK (particularly, software usage data) that is acquired during operation of earlier versions of the software. Also, SOK is used to determine the schedule and composition of new software releases: inter alia crash data and end-user feedback are used to determine the impact of particular in-the-field software operation failures, and therewith decide on the priority of including a fix to these failures in a next release of the software.

NetComp

Operation knowledge of NetComp software operating on set-top boxes in the field is acquired to measure feature adoption and therewith determine cost-effectiveness of the boxes. In other words, if operation data analysis indicates that certain features of the set-top box software are rarely used by end-users, NetComp considers the software to offer too much (or too specific) functionality. The company then attempts to build a new version of the box operated by the same software minus the rarely-used features, for a lower price.

ERPComp

While ERPComp mainly uses acquired or received SOK in release management and product line planning processes to decide which software mutations (updates with bug fixes or new functionality) affect most customers positively, the vendor foresees SOK to be increasingly used in optimization of internal business processes. Based on SOK analysis, the vendor witnessed a significant change in its average end-user's work-life balance. ERPComp could use this (location-dependent) knowledge not only as input for portfolio planning processes, but also for staffing its support departments, for example.

CADIntComp

According to CADIntComp, propagation of SOK through their CADInt ecosystem supports realistic planning of new (versions of) CADInt releases: if crash log mining indicates weak areas in CADInt software, CADIntComp focuses on strengthening these areas when determining specifications for the upcoming release. Code repositories, build- and test results, roadmaps and design specifications as well as information resulting from crash log mining in the form of operation reports, allow members to get insight in build status, development progress and trends concerning in-the-field software operation quality and performance.

KEY DERIVATIONS SOK propagated to software ecosystem keystones is utilized by these participants to determine the composition of new versions of their software, and to schedule release of these versions. In the long term, software vendors use received SOK to *optimize the cost-effectiveness of their software products* by aligning product functionality with actual feature adoption, as well as internal business processes (e.g. support department staffing).

3.4.5 SECO Orchestration Strategy and Policy Development

OSComp

Apart from software quality, OSComp uses its Quality service to monitor the satisfaction of its partners as well as the satisfaction of its partners' end customers. This allows the company to recognize and orchestrate areas of improvement across an ecosystem. If such areas are identified, OSComp assists involved niche players in realizing improvement of software quality or partner satisfaction, and therewith increase stability and attractiveness of its ecosystems. In addition, OSComp uses its certification programs as a way to control the level of collaboration with niche

players. If a particular participant appears to distort ecosystem stability, OSComp can diminish the player's ecosystem participation, for example by limiting its platform privileges.

NetComp

One of NetComp's strategies to orchestrate its software ecosystems is to adapt its partner alliance program specifically to the needs of particular partners. Based on a partner's software operation data history, NetComp creates partner-specific incentives and policies. Also, the company involves partners of certain alliances in upscaling the operation speed and scale of its ecosystem. By prospecting potential access to its intellectual property to partners, NetComp stimulates partners to build robust software and behave well in its ecosystems.

KEY DERIVATIONS The way SOK is used by keystones to orchestrate their software ecosystem is influenced by the trust between these partners. With a relative lack of trust, keystones concentrate their incentives on niche player participation. With trustful relationships, keystones adapt their incentives and policies per partner, to *maximize each partner's contribution to ecosystem stability and attractiveness*.

3.4.6 Future Intentions

OSComp

According to OSComp, one of the biggest challenges in software development is to understand the actual intent of end-users during software operation. While software crashes that are a direct result of an end-user action obviously are considered as unwanted behavior, it is often unclear what was an end-user's actual intent when a window, waiting for response, was closed. Also, the company expects software development processes to further evolve around the question 'What is the end-user doing?'. To an increasing extent, operation failures are no longer constrained to one machine, one piece of software or one software vendor. End-users increasingly interpret operation failures as a failure of the 'total thing', i.e., the computing experience as a whole. One of OSComp's future challenges is to get a more thorough insight in the cause and frequency of software that is not fulfilling end-user expectations, based on SOK that is acquired by or propagated to partners participating in its software ecosystems.

		SECO Life Cycle Phase	
		Creation	**Continuation**
SECO Orchestration Focus	**Actor Level**	Trust building (3.4.1) • Certification programs • Developer keys	Trust strengthening (3.4.5) • Advanced certification programs • Private SDK access
	Ecosystem Level	Niche player participation (3.4.1) • Generic participation incentives • Niche teams	Partner contribution (3.4.5) • Partner-specific incentives • Partner teams
SOK Propagation Focus	**Actor Level**	Software quality management (3.4.3) • Operation failure cause identification • Heisenbug isolation	Portfolio and product line planning (3.4.4) • Release composition and scheduling • Cost-effectiveness optimization
	Ecosystem Level	Relationship management (3.4.1) • Software operation monitoring • Software compatibility verification	Ecosystem composition comprehension (3.4.2) • Niche player entrance and exit monitoring • Hardware location tracking

Table 3.3 Classification of identified orchestration and propagation practices. Numbers between parentheses refer to the respective sections presenting the corresponding interview results

NetComp

For NetComp, challenges lie in responding adequately to acquired or received software operation knowledge. The relatively uncomplicated software operating on set-top boxes is automatically updated when, based on software operation data, a bug in the software has been identified and fixed. Simultaneously, operation data acquired from NetComp's Net software running on mission-critical instances of its most advanced enterprise hardware, require highly sophisticated response to administrators of the hardware. Therefore, NetComp is investigating ways to not only use the 'power of many' in terms of SOK acquisition and utilization, but also let its employees respond adequately to received SOK. The company foresees a transition to software that is based on a platform of which is formally proven that it is stable and correct; software functionality that is not part of the platform is completely modularized. According to NetComp, such a software architecture will alleviate bug localization, module dependency determination and run-time software component updating processes.

ERPComp

In the context of software ecosystems, ERPComp envisages to provide its partner niche players with real-time add-on operation knowledge by means of a central add-on operation dashboard. According to the vendor, adequate dashboard use would result in more intimate collaboration with its partner niche players, and ultimately result in continuous improvement of software operation and software functionality.

CADIntComp

While SOK propagation helps CADIntComp in obtaining insight in the capabilities of a niche player in terms of improving CADInt technology, as well as strengthening its relationship with particular niche players, CADIntComp intends to acquire more knowledge of how CADInt is used and appreciated in the field in future. Therefore, it attempts to further intensify relationships with its niche players, since they are in closer contact with actual end-users. CADIntComp may introduce a 'social requirements engineering system', in which the priority of niche players' feature requests is based on a niche player's participation in the CADInt ecosystem, and agreement with the feature request by other niche players.

KEY DERIVATIONS On both SECO scope levels, there are several areas in which keystones plan to improve. First, keystone vendors intend to increase

effort to more precisely understand their end-users' actual behavior and intentions during software operation. Also, while SOK is contributive to understanding and responding to end-users, it is a challenge to *determine to which extent response to acquired SOK is experienced as adequate by the end-users*. Third, keystones envisage to propagate acquired SOK realtime to partners, providing them with realtime software operation graphs and statistics, and involve these partners in decision making, for instance concerning requirement prioritization.

3.5 ANALYSIS OF SOK PROPAGATION PRACTICES

After interpretation, contextualization and abstraction of both the orchestration and propagation practices, as well as the key derivations described in section 3.4, a classification along two dimensions (SECO life cycle phase and scope level) was established in accordance with the principles for interpretive field research of Klein and Meyers [Klein and Myers 1999] (see table 3.3). As reflected by the interview results, challenges resulting from software ecosystem participation are not addressed in a predetermined, specific order. Simultaneously, it appears that during the software ecosystem life cycle, focus of SECO orchestration practices (of keystones) and SOK propagation practices (of ecosystem participants in general) changes — both on actor level and ecosystem level.

When keystones begin to orchestrate their software ecosystem, for instance, their orchestration strategy and incentives are focused on building trustful relationships with other software-producing organizations (actor level) and stimulating them to participate as a niche player in their software ecosystem (ecosystem level). Keystones then propagate SOK to address the challenges of relationship management (e.g. through software operation monitoring and compatibility verification) and software quality management (e.g. through utilization of SOK for identifying the root cause of software operation failures, or Heisenbug isolation). In the longer term, keystones attempt to sustain orchestration of their software ecosystem.

On the longer term, orchestration of keystones is focused on strengthening the trust a niche player has in the ecosystem and its keystone (actor level) and on the actual contribution of partners to the ecosystem (ecosystem level). While ecosystem actors continue to be focused on software quality management, SOK is propagated for planning software portfolios and software product lines (e.g. through alignment of product functionality with actual feature adoption), and to maintain insight in the ecosystem's composition (e.g. through monitoring ecosystem entrance and exit).

Based on the identified practices, we particularly consider the challenges described in sections 3.4.1–3.4.4 (i.e. relationship management, ecosystem composition comprehension, software quality management, portfolio and product line planning) addressed through utilization and propagation of SOK. We consider the challenge described in section 3.4.5 (i.e. orchestration strategy and policy development) to a lesser extent supported by the identified practices, since we found limited concrete proof for the addressing of this challenge through actual utilization and propagation of SOK.

While challenges resulting from software ecosystem participation are addressed through effective utilization and propagation of SOK, additional challenges emerge. For example, keystones should be prepared to cope with niche player distrust resulting from SOK utilization (e.g., when SOK propagated to a keystone indicates frequent unstableness of certain niche player software). Second, vendors that acquire, utilize and propagate SOK to understand and serve their end-users, may find it challenging to determine how and when they respond in such a manner, that the end-user feels understood and served well. Third, concerning SOK acquisition, utilization and propagation processes, software-producing organizations will have to find an appropriate balance between realizing adequate customer intimacy and protecting the same customer's privacy.

3.6 CONCLUSIONS AND FUTURE WORK

While software vendors are aware of the software ecosystems in which they operate and their relationships with other ecosystem participants, vendors all too often are unsuccessful in recognizing the potential value and role of software operation knowledge (SOK) in software ecosystems.

In this paper, we state that SOK forms an essential pulse in the veins of software ecosystems, and serves as a primary enabler for software vendors to thrive and flourish in the ecosystems in which they operate. We substantiate our position by reporting on the operational SOK utilization and propagation practices of four software-producing organizations, which are successfully operating in software ecosystems and address challenges resulting from participation in these ecosystems through utilization and propagation of SOK. With in total four of five challenges addressed by these organizations, we found substantial support for our position.

Based on analysis of the identified practices, we conclude that software ecosystem keystones should continuously invest in infrastructures for SOK acquisition, utilization and propagation to (allow niche players to) effectively utilize the 'power of many' in obtaining knowledge of in-the-field software and end-

user behavior, and therewith in increasing the quality and robustness of their software. In the long term, successful SOK utilization and propagation provide keystone software vendors with competitive advantage in terms of ecosystem stability, ecosystem attractiveness, software quality and end-user satisfaction. Ecosystem participants should aim for stable and robust relationships: such relationships contribute to a stable and attractive software ecosystem. Therefore, mutual trust between software ecosystem participants is essential for effective SOK propagation within software ecosystems: distrustful ecosystem participants diminish or even discontinue propagation of their SOK to partners, which may cause these partners to focus on participation in alternative, rival ecosystems.

Future work includes quantification and qualification of SOK propagation in software ecosystems. For example, metrics for measuring the effectiveness of SOK propagation and the contribution of SOK propagation to the vitality of software ecosystems are to be developed.

Part III

Process Improvement through Software Operation Knowledge

4

Reducing Maintenance Effort through Software Operation Knowledge

ABSTRACT

Knowledge of in-the-field software operation is acquired unsophisticatedly: acquisition processes are implemented ad hoc, application-specific and are only triggered when end-users experience severe failures. Vendors that do acquire such knowledge structurally from their software applications, often are unsuccessful in visualizing it in a consistent and uniform manner. A generic approach to acquisition and presentation of software operation knowledge reduces the time vendors need to integrate acquisition logic into their applications, as well as the time needed to analyze, compare and present uniform software operation data resulting from in-the-field software operation. This paper proposes a technique for software operation knowledge acquisition and presentation through generic recording and visualization of software operation. A prototype tool implementing this technique is presented, as well as an extensive empirical evaluation of the tool using an eclectic set of instruments (an experiment, two case studies and expert focus group discussions) involving three widely-used software applications. Results show that the technique is expected to reduce software maintenance effort and increase comprehension of end-user software operation.*

*This work has been published as *Reducing Maintenance Effort through Software Operation Knowledge: An Eclectic Empirical Evaluation* in the proceedings of the 15th European Conference on Software Maintenance and Reengineering (CSMR 2011) [Van der Schuur *et al.* 2011b]. It is co-authored by Slinger Jansen and Sjaak Brinkkemper.

4.1 INTRODUCTION

One of the most challenging tasks in software maintenance is to *understand* how software operates[1] in the field. While software vendors strive to build fast, robust, and intuitive software and therefore extensively test and verify it in their own environment, a plethora of hardware and software environment facets cause software to behave differently in the field. Unknown bugs, incompatibility issues and performance problems are just three types of complications that may surface only after deployment [Glerum *et al.* 2009]. These complications can be mitigated using knowledge of in-the-field software operation during typical software engineering tasks such as bug localization and fixing, crash analysis and user experience improvement [Madhavji *et al.* 2006]. Insight in the various environments software operates in, as well as knowledge of how end-users behave, their expectations of the software and their actual intentions of using the software are only a few examples that can aid software vendors in building robust and stable software applications.

While interest in software operation knowledge or SOK (i.e., knowledge of in-the-field software operation) has broadened to include software performance, quality and usage, as well as end-user experience feedback aspects [Van der Schuur *et al.* 2010], software vendors still acquire SOK in an unsophisticated manner [Jansen *et al.* 2008]. Acquisition processes are implemented ad hoc, application-specific and exception-triggered, which causes acquired data to be unstructured and not uniform across applications. As a consequence, mining, analysis and integration of acquired data can be time-consuming and error prone, while software engineering activities benefit only little. Software vendors that do structurally acquire software operation data, frequently struggle with extracting valuable software operation knowledge from these data and visualizing extracted knowledge effectively and meaningfully.

The main question we attempt to answer in this paper is '*How can software maintenance effort be reduced through generic recording and visualization of operation of deployed software?*'. To answer this question, we propose a novel technique that (1) enables software vendors to acquire SOK independent of target software, (2) allows vendors to get insight in operation of their software in the field and (3) contributes to reduction of software maintenance effort. We present a prototype tool that implements this technique and enables vendors to analyze, visualize and compare uniform operation data, allowing easy presentation and

[1] In this paper, we define 'software operation' as the fact or condition of deployed software functioning in a specified manner.

utilization of such data. The generic SOK acquisition and presentation technique with corresponding tool are the contributions of this research.

The soundness and industrial utility of the technique are demonstrated through evaluation of the tool using an eclectic (i.e. deliberately composed) set of empirical evaluation instruments [Easterbrook *et al.* 2008, Kitchenham *et al.* 2008]: an experiment and two case studies (i.e., field study [Hevner *et al.* 2004]) as well as expert focus group discussions. The evaluation involves three widely-used software applications.

This paper continues with placing our work into context in section 4.2. Next, the software operation knowledge acquisition and presentation technique is proposed (section 4.3). Section 4.4 introduces the prototype tool; the empirical evaluation approach and results are described in section 4.5. Finally, research limitations (section 4.6), conclusions and future work (section 4.7) are presented.

4.2 RELATED WORK

Many research efforts and tools cover the subject of SOK acquisition, but refer to such knowledge with various denotations and use it to accomplish various goals. First, software operation knowledge is acquired to monitor deployed software [Bowring *et al.* 2003, Nusayr and Cook 2009, Kristjánsson and Van der Schuur 2009]. However, in general, code modifications are needed in order to integrate monitoring techniques with the target software. Furthermore, while these techniques suffice in signaling problems regarding the functioning of software, they assist only little in pinpointing problem causes and actually eliminating bugs. Using our technique, no code modifications are needed to enable operation recording. Operation recording visualizations assist in identifying and eliminating software failure causes.

Second, SOK is also acquired to debug software. Clause and Orso [Clause and Orso 2007] present a technique for recording, minimizing and replaying failing software executions. The technique is limited, however, since only interactions between an application and its environment as well as 'relevant' portions of the environment are recorded. While our technique also records certain environment details, relevant method events are recorded instead of interactions between an application and its operation environment.

Narayanasamy *et al.* [Narayanasamy *et al.* 2005] propose their BugNet architecture that continuously records information during software production runs, to support developers in characterizing bugs by enabling them to replay the program's operation before a crash. Even though BugNet provides the abil-

ity to replay an application's executions across context switches and interrupts, BugNet requires a specific environment as well as significant effort to be integrated in a vendor's software product. Also, it is left unclear to which extent the proposed techniques contribute to software maintenance or software operation comprehension. Our technique allows easy integration into existing software products, and supports both software maintenance and software operation comprehension.

Third, several techniques for presenting software operation recordings exist [Cornelissen *et al.* 2008, Jones *et al.* 2004]. Although these techniques are sophisticated, it is unclear what are the usage requirements for these techniques and to which extent these techniques contribute to comprehension of end-user software operation.

4.3 SOK ACQUISITION AND PRESENTATION

We propose a software operation knowledge acquisition and presentation technique that is designed to reduce software maintenance effort and increase comprehension of end-user software operation, through generic (i.e., independently of target software) recording and visualization of software operation.

Existing approaches have three shortcomings with respect to this goal. First, most approaches require significant integration effort (e.g. source code changes) to realize operation recording or visualization (as discussed in section 4.2). Second, most approaches only work for a specific system or software application and thus are not generically applicable. Third, resulting recordings often contrast in structure and format, depending on the software of which the operation is being recorded. As a consequence, recordings are presented erratically and are difficult to analyze, comprehend or compare. Moreover, software engineering tasks benefit only little from acquired knowledge of the behavior of software and end-users in the field. We addressed these issues by developing a technique that (1) allows generic recording of in-the-field software operation, without requiring thorough knowledge of the composition (i.e. source code) of the target software, or deployment of additional tools and (2) allows uniform storage and visualization of resulting operation recordings.

Our SOK acquisition and presentation technique consists of a weaving (4.3.1), recording (4.3.2) and a visualization (4.3.3) process and can be mapped onto the SOK framework [Van der Schuur *et al.* 2010]; a usage scenario[2] is provided in figure 4.1. Recording of software operation is preceded by a weaving process

[2]Note that the SOK framework integration process is optional [Van der Schuur *et al.* 2010] and omitted in this figure.

in which SOK acquisition logic is woven into the executable that is the target of operation recording. This logic is responsible for generically acquiring operation data and uniformly writing operation recordings to disk. Operation recordings resulting from application of our technique, called SOK prints, represent behavior of both software and end-user during software operation in the form of event sequences and sources. SOK prints can be visualized and replayed. By analyzing these recordings, knowledge of software performance, quality and usage during operation recording can be acquired.

4.3.1 Weaving

Aspect-oriented programming (AOP) is effectively deployed in the domains of software monitoring and tracing [Nusayr and Cook 2009, Van der Schuur *et al.* 2008]. However, we encounter in industry that product software vendors have to overcome time-consuming obstacles when they leverage AOP repeatedly and separately for each of their software products: vendors write product-specific SOK acquisition aspects and extend their AOP libraries to new (versions of the same) software products.

In our technique, aspect weaving is used to weave SOK acquisition logic independent of the bytecode (e.g. Java bytecode or the .NET Common Intermediate Language) of the executable of which operation has to be recorded, without requiring knowledge of, or integration into the source code of a target executable. Given the set of all methods of an executable $\mathcal{M}$, $\mathcal{S}$ is the set of weaving candidate methods, where $\mathcal{S} \subseteq \mathcal{M}$. For each method $\mu \in \mathcal{S}$, acquisition logic in the form of a set of advices $\mathcal{A}$ is woven into the executable at join points $\gamma_{entry}(\mu)$, $\gamma_{exit}(\mu)$ and $\gamma_{exception}(\mu)$. By weaving the logic at those join points, software performance, quality and usage can be recorded.

When the weaving process has finished, a SOK assembly (an executable containing the woven acquisition logic) is compiled. Also, a SOK assembly descriptor is generated. This extensible structure description contains all method descriptions (i.e., signature, visibility and return type) of all classes of the executable into which acquisition logic is woven. During the assembly descriptor generation process, a unique key is assigned to each method and method parameter. SOK print events are created during operation recording and correspond to one method and its parameters. Methods and parameters are referenced by these keys to minimize the SOK print size, and performance loss induced by the woven acquisition logic.

4.3.2 Recording

When a SOK assembly is executed, three types of events $\epsilon(\mu)$ are recorded for each method $\mu \in \mathcal{S}$: method entries, $\epsilon_{entry}(\mu)$, method exits, $\epsilon_{exit}(\mu)$ and unhandled exceptions, $\epsilon_{exception}(\mu)$. SOK acquisition occurs when, as part of software operation, an $\mu \in \mathcal{S}$ is called. A SOK print is created with the first call to any $\mu \in \mathcal{S}$. With the occurrence of each event $\epsilon(\mu)$, the SOK print is updated with additional data.

Every event $\epsilon(\mu)$ references the method μ it occurs at, as well as the parameters of μ, with the keys by which the method or parameters are defined in the assembly descriptor. Also, the time and date at which an event $\epsilon(\mu)$ occurred, as well as an event source identifier, are stored for each event. An event source represents a user (profile) that is (indirectly) accountable for events. Per event type, additional data are recorded:

Method Entries
: Per method entry event $\epsilon_{entry}(\mu)$, all string representations of the values of all method parameters are recorded, where a string representation is the string return value of a public method that can be called on an object of the same type as the parameter variable. If a method has no parameters, only call time and date are recorded.

Method Exits
: Per method exit event $\epsilon_{exit}(\mu)$, all string representations of the method return value are recorded, where a string representation is the string return value of a public method that can be called on an object of the same type as the method return value. If a method's return type is `void`, no data are recorded.

Unhandled Exceptions
: Per unhandled exception $\epsilon_{exception}(\mu)$, the exception's type, message, stack trace and data object are recorded. If an exception cause is defined in the form of an InnerException, the type, message and stack trace corresponding to the inner exception are recorded in addition.

Since end-users are the (indirect) source of software operation, an event source contained in a SOK print is described by means of properties that uniquely identify the user currently executing the woven software. Together with these credentials, a source description consists of the operation session start date and time, environment variables, hardware specifications and operating system details of the system on which the woven software is executed. When an end-user

executes a SOK assembly, closes it and executes it again, the corresponding SOK print contains two source descriptions, since two operation sessions have taken place.

4.3.3 Visualization

SOK prints can be represented graphically to support comprehension of both software and end-user behavior during software operation. For example, the causality of user actions is visualized as a state or flow diagram. Event sources are visualized based on their specific properties (e.g. username), and operation recording statistics as well as corresponding graphs are created based on the operation data that constitute one or multiple SOK prints. Also, visualizing recorded event chains can support (comprehension of) event replays. Visualization of SOK prints is not dependent on weaving or recording processes.

4.4 NUNTIA TOOL

To evaluate our SOK acquisition and presentation technique, we implemented it in a binary instrumentation tool named Nuntia. In line with the description of our technique in section 4.3, the tool enables generic weaving of SOK acquisition logic into assemblies and provides software operation recording functionality by creating SOK prints. A usage scenario of the technique using Nuntia is provided in figure 4.1.

4.4.1 Weaving

Implementation of a generic executable weaving process was a challenge. The structure and contents of compiled-code executables depend on the language the software is written in, the compiler used to create the executable as well as the platform the executable is compiled for. Since deploy-time (post-compilation) weaving requires disassembly, implementation of *language-, compiler- and platform independent* post-compilation weaving requires coping with many language-, compiler- and platform specific details that do not effect the principles of our technique. Therefore, we decided to focus on implementing the technique for one type of executables: Nuntia provides functionality to weave SOK acquisition logic into all .NET assemblies compiled with the C# compiler that is part of the .NET Framework[3] [Microsoft .NET Framework]. Although Nuntia requires .NET, our technique is also applicable to other languages that allow binary in-

[3]Note that Nuntia may operate platform independently by weaving into assemblies created with the C# compiler that is part of the Mono project [Mono Project].

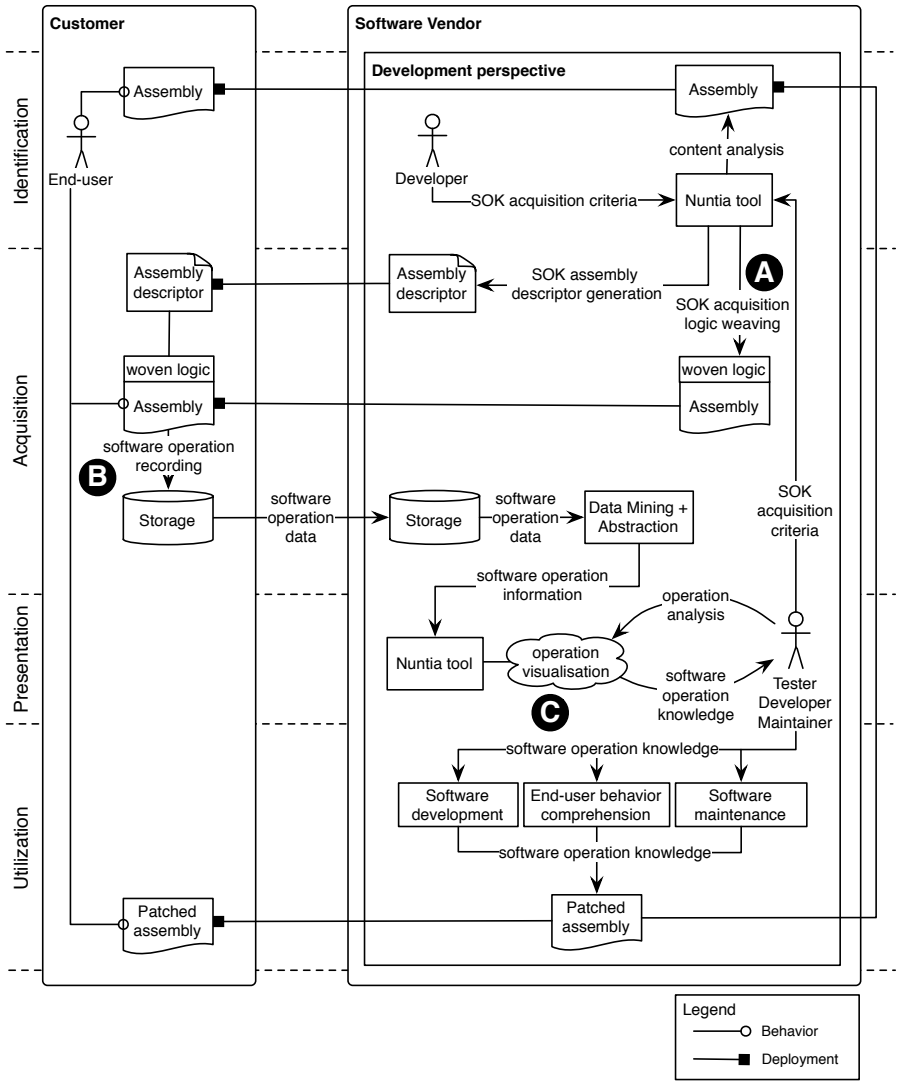


Figure 4.1 SOK acquisition and presentation technique usage scenario

strumentation and reflection (e.g. Java or Objective-C). To implement the SOK acquisition logic weaving functionality, a SOK acquisition host and a plug-in for the PostSharp framework [The PostSharp Platform] have been developed. A second challenge was to make sure that for (all possible executions of) all .NET assemblies, stacks were read and manipulated such that all types of parameters, return values and exceptions could be read correctly, without introducing unwanted side effects to the target software. We are aware of the fact that Nuntia might introduce unwanted side effects, for example in realtime, distributed or complex recursive methods. Future research and development is needed to limit these effects.

After successful weaving of the SOK acquisition logic at join points of selected methods (S), a SOK assembly descriptor is generated in the form of an XML file. SOK assembly descriptors are validated against an XML schema that defines the data structure in which assembly class, method and parameter characteristics are stored.

4.4.2 Recording

Analogous to the recording mechanism description in section 4.3, logic woven by the Nuntia tool creates an empty SOK print with the initial call of an assembly method. Similarly, with the occurrence of method entries, method exits and unhandled exceptions during operation of the assembly, the SOK print is updated with additional data. SOK prints are stored in the form of an XML file that is validated against an XML schema definition.

To minimize performance loss induced by SOK print updating, names and values are stored in SOK assembly descriptors and referenced by SOK prints instead of repetitively including those in SOK prints. However, one should recognize that both the performance loss induced by SOK print updating as well as the size of the resulting SOK prints remain directly proportional to the number of join points at which acquisition logic is woven.

4.4.3 Visualization

The Nuntia tool contains functionality to visualize and replay in-the-field software operation that is recorded in the form of SOK prints. Event sequences are replayed event-by-event.

Figure 4.2 shows a screenshotof Nuntia's SOK print visualization and replay functionality. A SOK print is visualized as a software operation sequence graph, with one edge (event) and two nodes (methods) highlighted. Nodes can

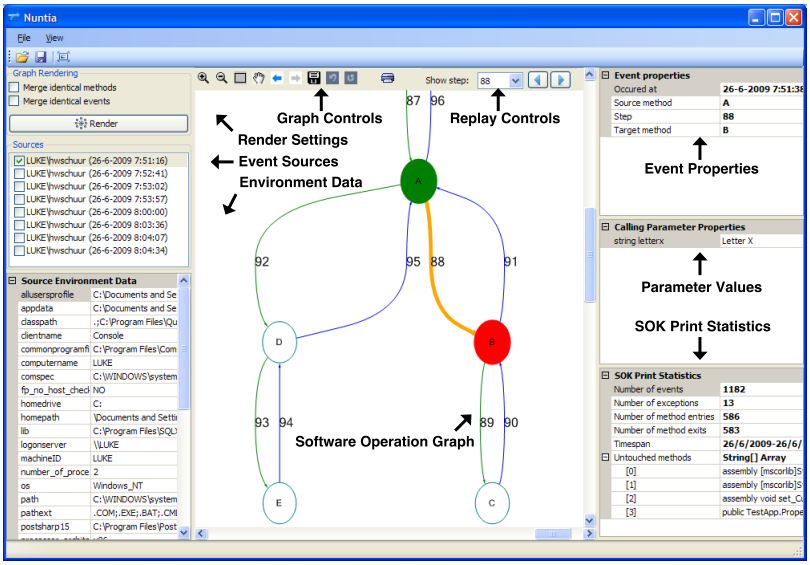


Figure 4.2 Nuntia SOK print visualization and replay

have an elliptical or rhombic shape. Elliptical nodes represent methods; rhombic nodes represent exceptions. In figure 4.2, for instance, edge 88 represents an entry of method B from method A and is highlighted. Since the selected event is a method entry event, Nuntia shows the types, names and (string representations of) parameter values that are passed to method B as part of the entry event. With the highlighted call to method B in figure 4.2, only one parameter named `letterx` with value `Letter X` is passed. For a method exit event, Nuntia shows the type, name and value of the return variable. When a method is selected, the method's class, type, name, signature and percentage of successful returns are displayed.

In addition to event and method details, Nuntia shows properties of the SOK print that is loaded. The number of events, method entries, method exits, exceptions as well as the time span during which the SOK print was recorded are displayed. Also, as shown in figure 4.2, Nuntia shows a list of event sources and per event source a list with environment data. By selecting multiple sources, event sequences of multiple sources can be rendered simultaneously. The Nuntia tool also allows simultaneous visualization of multiple software operation sessions (see figure 4.5). When doing so, instead of event numbering, Nuntia shows at the edges the number of times a particular event has occurred during the total time all SOK print recordings were recorded. SOK print visualization

is implemented using Microsoft Automatic Graph Layout [Nachmanson *et al.* 2008]. Nuntia is written in C# and requires version 3.5 SP1 of the .NET framework [Microsoft .NET Framework].

4.5 EMPIRICAL EVALUATION

To investigate the soundness and industrial utility of the SOK acquisition and presentation technique, the following research questions and propositions were evaluated. If all propositions corresponding to a particular research question are true, we consider that question to be answered positively.

RQ 1 — *Does the technique allow generic software operation recording?*

P1 Nuntia weaves functioning SOK acquisition logic into an executable, preserving the executable's original functionality without introducing unwanted or unexpected side effects.

P2 Nuntia weaves functioning SOK acquisition logic into executables of diverse applications, developed by distinct and independent software vendors.

RQ 2 — *Does the technique allow accurate recording and replay of software operation?*

P3 The events recorded in SOK prints by Nuntia are consistent with software operation during recording.

P4 Visualizations and replays of SOK prints resulting from Nuntia are consistent with software operation during recording.

RQ 3 — *Does the technique support software maintenance and operation comprehension?*

P5 Nuntia provides valuable insight in, and increases comprehension of behavior of in-the-field software and end-users

P6 Nuntia discloses new knowledge that is useful in software maintenance activities and is not available without the introduction of Nuntia.

P7 Nuntia reduces the time needed to analyze software operation failures.

After Easterbrook *et al.* [Easterbrook *et al.* 2008] and Kitchenham *et al.* [Kitchenham *et al.* 2008], we employed an eclectic set of empirical evaluation instruments to answer these research questions, and therewith evaluate our technique in both scientific and industrial contexts.

Table 4.1 shows per research question which evaluation instrument is used to answer the question. A questionnaire was used as a basis for expert focus group discussions and case study evaluation (for reasons of employee availability, case study results were evaluated with CADComp employees). Participants confirmed or rejected each statement using a Likert scale (1: strongly disagree, 5: strongly agree). Questionnaire statements and results are presented in table 4.2.

Empirical Evaluation Instrument		**RQ1**	**RQ2**	**RQ3**
Field Study	Paint.NET Experiment	✓	✓	
	Industrial Case Studies	✓	✓	✓
Expert Focus Group Discussions				✓

Table 4.1 Empirical evaluation instruments per research question

We deliberately selected both free and commercial software that is widely used in the field, to acquire SOK from. All three subjects based on the .NET framework [Microsoft .NET Framework]: Paint.NET is based on Windows Forms (C#), ERPProd on ASP.NET (Visual Basic) and CADProd on Windows Presentation Foundation (C#). For all three subjects, we tried to create a realistic instance of the scenario presented in figure 4.1. We regard the research as repeatable with the same results, presuming similar circumstances (similar tools, similar-sized software vendors, etc.). Note that for reasons of confidentiality, names of the latter subjects and their vendors have been anonymized.

Paint.NET Experiment

The correctness of Nuntia's software operation recording and visualization functionality (i.e., the extent to which Nuntia SOK prints represent the actual software operation correctly) is demonstrated by comparing SOK acquired by Nuntia during usage of Paint.NET, throughout we deliberately exposed bugs, with Paint.NET's change log. Paint.NET is a free and widely-used image and photo editing application for the Windows operating system, developed by dotPDN. Since the 1.0 release in 2004, 14 releases of the application have been published. The last stable release at the time of writing is 3.5.6, which dates from November, 2010.

Industrial Case Studies

The extent to which Nuntia reduces software maintenance effort, and supports software development as well as comprehension of end-user software operation, is evaluated by means of two case studies performed at CADComp and ERPComp.

CADComp is a European software vendor that was founded in 1990 and is currently performing development activities in the Netherlands, Belgium and Romania. Development of CADProd, a CAD drawing management application, started in 2007. Version 1.0 has been released in 2009 and is used by more than 300 customers. ERPComp is an ERP software vendor with 2,500 employees and establishments in 40 countries. ERPComp was founded in 1984 and serves customers in 125 different countries. ERPComp develops ERPProd, an accounting solution provided as a secure online internet service. ERPProd 1.0 has been released in 2005. Since then, ten major versions have been released. The last release at the time of writing is 2010-2, which is used by 16,600 customers.

SOK prints resulting from recording operation of these two industrial software applications (CADProd being deployed at the customer site) using Nuntia, were analyzed and discussed with key developers, maintainers and managers employed by the two vendors. Also, these employees were invited to participate in evaluative sessions in which their opinions about the functionality and utility of the Nuntia tool were evaluated.

Expert Focus Group Discussions

Chief technology officers, product managers and senior team leaders from industry (recruited by means of an invitation sent to our professional and educational networks) were assembled in a focus group to discuss the utility of both the Nuntia tool and the SOK acquisition and presentation technique, in processes not directly related to software development (e.g. product management).

4.5.1 Paint.NET Experiment

APPROACH After examining Paint.NET's change log and road map, we selected version 3.35 to evaluate Nuntia. According to its change log, this version contains a bug causing a program crash when encountered, which is fixed in the subsequent release (Paint.NET 3.36). According to the change log, this particular version would crash 'When using the "Fixed Ratio" feature of the Rectangle Selection tool, if 0 was specified for both the width and height' [Paint.NET

Roadmap and Change Log]. During evaluation we focused on this bug, which we will refer to as 'fixed ratio bug'.

First, we deployed Paint.NET on our test machine. Using Nuntia, we generated SOK assemblies and assembly descriptors for Paint.NET 3.35. $\mathcal{S}_{Paint.NET}$, a set of Paint.NET's class methods used by the tool to weave SOK acquisition logic into the Paint.NET assemblies, was composed based on the Paint.NET change log as well as the assembly metadata Nuntia provides after having read an assembly.

In this particular case, according to the change log, the bug is related to the 'Fixed Ratio' feature of the 'Rectangle Selection' tool. Therefore, we focused on Paint.NET's classes `PaintDotNet.Tools.SelectionTool` and `PaintDotNet.Tools.RectangleSelectTool`. Of those classes, the methods `CreateSelectionPolygon` and `CreateShape` were marked as candidates for $\mathcal{S}_{Paint.NET}$. During analysis of assembly information provided by Nuntia, we discovered the class `PaintDotNet.SelectionDrawModeInfo`. Of this class, we marked the methods `get_Width` and `get_Height`, since the fixed ratio bug crashes Paint.NET when '0' is specified for both the width and the height of the selection. Next, a SOK assembly was generated according to the process described in section 4.3. We reproduced the fixed ratio bug using the SOK assembly and analyzed the resulting SOK print afterwards. Next, we repeated these steps with Paint.NET 3.36 with $\mathcal{S}$ identical to the set that was used with Paint.NET 3.35 and compared the created SOK print with the one of Paint.NET 3.35. During the recording of both versions, identical actions were performed.

RESULTS Figure 4.3 shows the visualization of the SOK print that was created during the fixed ratio bug recording session we performed. The program crash caused by the bug is visualized by two rhombic nodes (which, as described in section 4.4, represent exceptions). The first exception occurs in method `CreateShape`, after the selection width and height both have been requested twice. Analyzing the SOK print using the Nuntia tool, the first exception is an `OverflowException` with the message 'Negating the minimum value of a twos complement number is invalid.', originating from `Math.AbsHelper(Int32 value)`, `PaintDotNet.Utility.PointsToRectangle(Point a, Point b)` and `PaintDotNet.Tools.RectangleSelectTool.CreateShape(List‘1 tracePoints)`. This exception was included in the pdncrash.log file Paint.NET generated when the application crashed. The second exception is a `NullReferenceException` occurring in method `CreateSelectionPolygon`, after `CreateShape` has finished. The second exception originates from `PaintDotNet.Utility.SutherlandHodgmanOneAxis(RectangleF bounds,`

Identifier	Statement	Average (σ)	
		CADComp	Focus group
S1	Nuntia provides new, valuable insight in the behavior of our software in the field	4.44 (0.61)	4.00 (0.82)
S2	Nuntia provides new, valuable insight in the behavior of our end-users in the field	4.31 (0.85)	4.00 (0.94)
S3	Nuntia increases my comprehension of the behavior of our software in the field	3.81 (0.81)	4.44 (0.50)
S4	Nuntia increases my comprehension of the behavior of our end-users in the field	3.75 (0.90)	3.89 (0.87)
S5	Nuntia shortens the time needed to find bugs	3.94 (0.83)	3.00 (0.94)
S6	Nuntia shortens the time needed to solve bugs	3.63 (0.93)	3.67 (0.94)
S7	Nuntia discloses knowledge I did not have before	4.06 (0.56)	4.33 (0.67)
S8	Nuntia supports our software maintenance activities	3.88 (0.78)	3.89 (0.57)
S9	Nuntia supports our software development activities	3.56 (0.93)	3.67 (1.05)
S10	Nuntia supports other activities (please specify which activities)	3.63 (1.11)	3.44 (0.68)

Table 4.2 Questionnaire statements and results

`RectangleEdge edge, List‘1 v)`,`PaintDotNet.Utility.SutherlandHodgman(RectangleF bounds, List‘1 v)` and `PaintDotNet.Tools.SelectionTool.CreateSelection Polygon()` with the message ‘Object reference not set to an instance of an object.’. This exception was not mentioned in the pdncrash.log file.

Figure 4.4 shows the visualization of the SOK print that was created during the recording session of Paint.NET version 3.36. As the fixed ratio bug has been fixed, no exceptions are part of this recording. Furthermore, the return types of methods `CreateShape` and `CreateSelectionPolygon` have changed from `void` in version 3.35 to `Collections.Generic.List‘1[Drawing.PointF]` and `Drawing.PointF[]` in version 3.36, respectively, as became clear after SOK print analysis using the Nuntia tool. In both versions, the return values of all calls to both the `get_Height` and `get_Width` methods were 0 (of type `float64`). In other words, the fixed ratio bug was not solved by altering the representation of the selection size parameters, or by preventing a selection with area zero to be created internally.

Regarding recording execution of the ‘Fixed Ratio’ selection feature with versions 3.35 and 3.36 of Paint.NET using the Nuntia tool, SOK prints resulting from this recording are consistent with the behavior of Paint.NET described in the change log of the software.

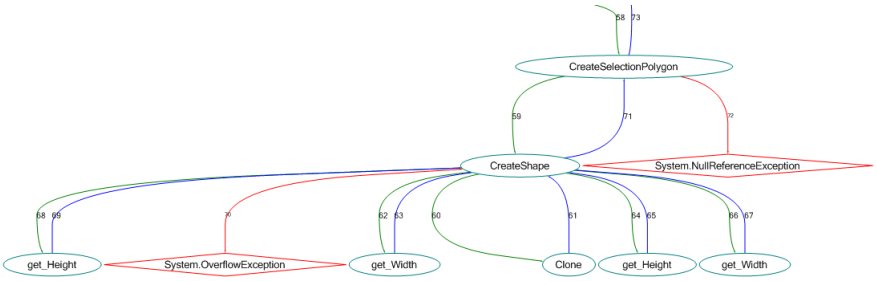


Figure 4.3 Visualization of 'fixed ratio' bug of Paint.NET 3.35

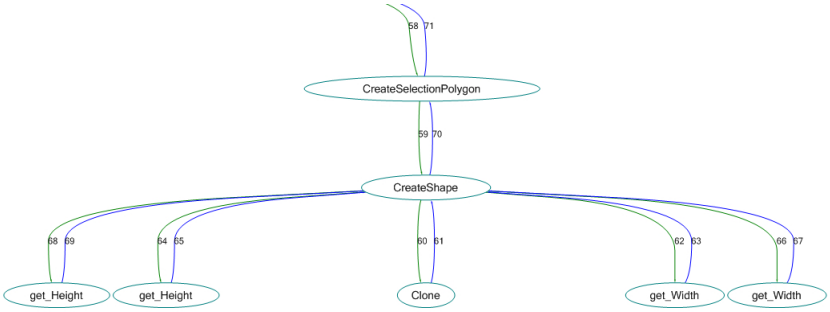


Figure 4.4 No exceptions occur using Paint.NET 3.36: fixed ratio bug fixed

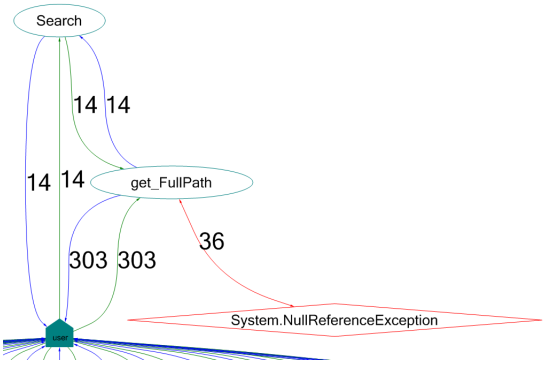


Figure 4.5 Visualizing all acquired CADProd SOK prints simultaneously

4.5.2 Industrial Case Studies

APPROACH

CADProd

First, CADProd's project leader was interviewed, to learn which parts of the software CADComp desires to acquire SOK from. He indicated that he was particularly interested in the data that is entered into CADProd by its end-users. Based on this information, all 'set_*XYZ*' methods of relevant CADProd business objects were marked as candidates for $\mathcal{S}_{CADProd}$, where *XYZ* is a business object property name. Also, methods handling search input and application launch, as well as methods that potentially throw exceptions were added to $\mathcal{S}_{CADProd}$.

Next, operation of CADProd, deployed at one of CADComp's 300 CADProd customers, was recorded by replacing the customer's original CADProd assembly with a SOK assembly generated by the Nuntia tool. Five of the customer's end-users were asked to use the software normally for about twenty minutes. Subsequent to the recording sessions, resulting SOK prints were analyzed and discussed by visualizing and presenting them to sixteen of CADComp's managers, team leaders and developers. Afterwards, these CADComp employees filled out the questionnaire (see table 4.2). Participating employees had an average of 11.6 years experience in information technology (σ = 5.36 years).

ERPProd

ERPComp daily monitors the performance, quality and usage of ERPProd, based on software operation data that is stored in application, error, help, and process logs. ERPProd's product line manager and one of ERPComp's senior research engineers were interviewed to determine if new and valuable SOK could be acquired from ERPProd using the Nuntia tool. Based on interview results, $\mathcal{S}_{ERPProd}$ was determined and Nuntia's generic weaving functionality was evaluated by creating SOK assemblies of certain ERPProd assemblies. Next, the assemblies were deployed in one of the ERPProd testing environments. Operation recording results were evaluated with the senior research engineer afterwards.

RESULTS

CADProd

Figure 4.5 shows a fragment of a simultaneous visualization of all SOK prints resulting from the CADProd operation recording

sessions. The figure illustrates to which extent CADProd usage by the five end-users during the recording sessions resulted in calls of two (of in total 86) methods in $\mathcal{S}_{CADProd}$. Methods `CADProd.Ulo.ULOFolderSearchResults.Search(Ulo.ULOFile File, string strSearch, bool bDescriptions, bool bAllProfiles)` and `CADProd.Ulo.ULOFile.get_FullPath()` are called 14 and 317 times, respectively. Of the 317 calls to the `get_FullPath` method, 36 (11,36%) caused a NullReferenceException. SOK print analysis by CADProd developers showed that these calls form a CADProd operation bug that was unknown to CADComp's CADProd developers. They reasoned that the `get_FullPath` method exception might be caused by too long path names for deeply nested projects. Also, the developers suggested to implement functionality to show a graph consisting of only UI-related methods and events, and proposed to display events in a list, both to increase Nuntia's usability.

Concerning the questionnaire results (see table 4.2 and figure 4.6), the participants tended to agree with the statements, on average answering the statements with 3.9 (σ = 0.83). Participants agreed the most with statements S1 and S2 (these statements were answered with 4.4 (σ = 0.61) and 4.3 (σ = 0.85) on average, respectively) and therewith found Nuntia to provide new, valuable software operation insights. There was most consensus on S7 ('Nuntia discloses knowledge I did not have before', avg. answer 4.1, σ = 0.56) and S1 (4.4, σ = 0.61). Least consensus was reached on S10 (σ = 1.11). Participants agreed the least on statements S9, S10 and S6; these statements were answered with 3.56 (σ = 0.93), 3.63 (σ = 1.11) and 3.63 (σ = 0.93) on average, respectively. In the context of S10, product management, training and customer relationship management were mentioned as other activities supported by Nuntia.

ERPProd

During the interview, the senior research engineer indicated that ERPComp uses a graphing component to visualize and monitor parts of the software operation knowledge they acquire. ERPComp developers have written a library on top of this component to provide it with XML data more easily. The engineer explained that when a graph looks faulty, developers would like to be able to see the data that is provided to the graphing component. Therefore, two of the component methods were added to $\mathcal{S}_{ERPProd}$: `ERPComp.FCharts.SingleSeriesChart.AddMeasurement(string ID, string`

`label, object value)` and `ERPComp.FCharts.SingleSeriesChart.get_XML()`. Generating a SOK assembly of the graphing component assembly, written in Visual Basic, went without problems. When the original assembly was replaced by the SOK assembly locally, a SOK print was created successfully after the first refresh of the local ERPProd environment. The data stored in the print matched the corresponding graphs that were shown in the ASP.NET application. After numerous successful tests, the SOK assemblies were deployed at the vendor's internal testing web servers. While SOK prints were created successfully at first, the environment showed signs of unstableness after some time, possibly related to concurrent requests, threading or file locking. The precise cause of instability could not be identified; future case studies will be carried out to continue the tests. During evaluation, the engineer indicated that although the prototype still needs work, valuable knowledge can be acquired with it. Also, he suggested the prototype to be extended with functionality to easily enable or disable SOK acquisition.

4.5.3 Expert Focus Group Discussions

APPROACH A SOK focus group consisting of nine experts employed by nine different European software vendors was assembled. The group consisted of two CTOs, four product managers and three senior engineers, with 14 years of experience in information technology on average (σ = 4.35 years). First, the Nuntia prototype was introduced to the focus group experts by means of a presentation in which the tool functionality and characteristics were exposed. Next, a tool demo was given during which our SOK acquisition and presentation technique was demonstrated on Paint.NET 3.5.1. Also, during this demo, the resulting SOK print was visualized and analyzed with the participants. Finally, a discussion about the technique and the tool was held and participants were asked to fill out the questionnaire (see table 4.2). Participants were asked to fill out the questionnaire with the assumption that the demonstrated technique was available for their software development platform or language.

RESULTS The nine focus group experts inclined to agree with the questionnaire statements (see table 4.2 and figure 4.7), on average answering the statements with 3.83 (σ = 0.80). Participants agreed the most with statements S3 and S7 (these statements were answered with 4.4 (σ = 0.50) and 4.3 (σ = 0.67) on average, respectively) and therewith found Nuntia to increase their comprehension of in-the-field behavior of their software, as well as to disclose knowledge they did not have before. There was most consensus on S3 (avg. answer

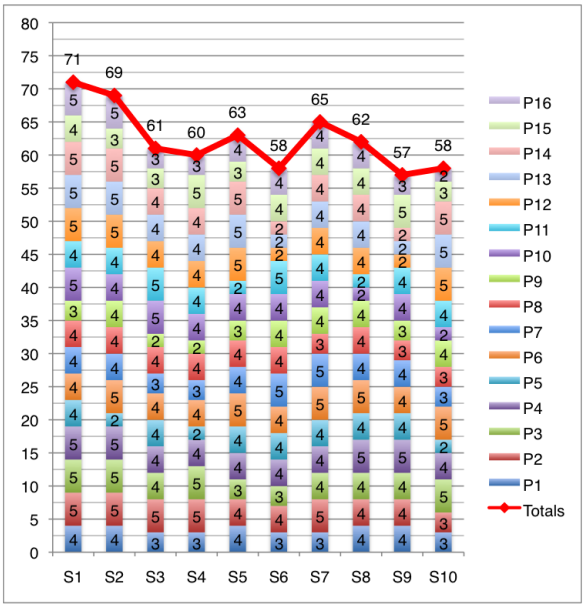


Figure 4.6 Answers to questionnaire statements by CADComp employees

4.4, $\sigma = 0.50$) and S8 ('Nuntia supports our software maintenance activities', avg. answer 3.9, $\sigma = 0.57$). Least consensus was reached on S9 ($\sigma = 1.05$). Participants agreed the least with statements S5 and S10; these statements were answered with 3.0 ($\sigma = 0.94$) and 3.4 ($\sigma = 0.68$) on average, respectively. In the context of statement 10, software testing and usability improvement were mentioned as other activities that are supported by Nuntia. Summarizing, participants found Nuntia to increase their comprehension of in-the-field operation of their software and to disclose knowledge they did not have before. Also, they were in harmony about finding Nuntia significantly supporting their software maintenance activities.

During the discussion, participants stated that both the Nuntia tool as well as the SOK acquisition and presentation technique it implements have high potential. They indicated that insight in behavior of (end-users on) software (1) is frequently required in product management and technical support processes, and (2) is provided by the Nuntia tool. Also, participants saw utility of the tool in software testing and quality assurance processes by simulating realistic software operation during these processes, allowing them to acquire and analyze SOK before actual deployment of the software.

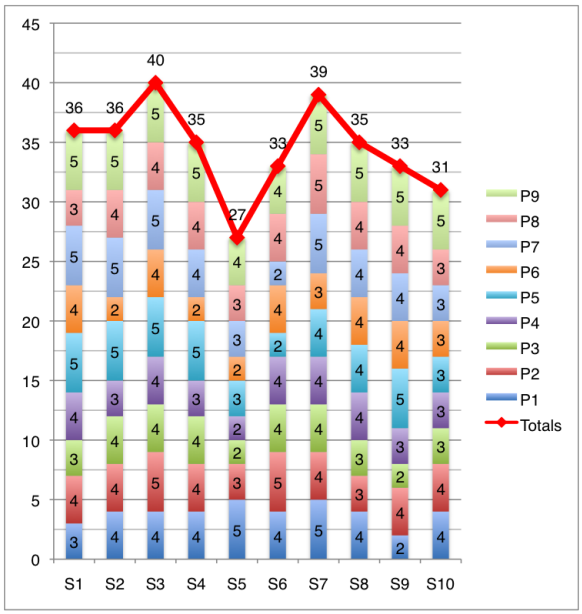


Figure 4.7 Answers to questionnaire statements by focus group experts

The SOK acquisition and presentation technique in particular was valued especially because of its post-compilation and SOK acquisition criteria selection characteristics, which, according to the participants, enable one to quickly acquire valuable SOK without having thorough knowledge of the software source code itself. The Nuntia tool was praised because of (1) its generic weaving and recording functionality, (2) the small footprint of SOK assemblies compared to the original assemblies and (3) the negligible performance loss induced by the SOK acquisition logic. However, the composition of $\mathcal{S}$ was still considered quite labor-intensive. Also, it was suggested to add graph filtering and critical path indication functionality to Nuntia's SOK print visualization features. Finally, participants appreciated the separation of SOK acquisition and presentation by means of a generic recording format.

SUMMARY Empirical evaluation results can be summarized as follows: (1) the technique was considered sound and viable by developers, maintainers and managers, mainly because of its flexibility in defining SOK acquisition criteria and its post-compilation acquisition characteristics; (2) both the technique and the tool were praised by managers expecting to gain further knowledge and insights from the tool and technique (once implemented), helping them

to rapidly increase software quality; (3) although still a prototype, Nuntia was already valued by developers and maintainers because they expect faster bug reproduction and fixing by using the tool.

TECHNICAL DETAILS Operation recording of Paint.NET was performed on a Core 2 Duo T7500, 2.20 GHz, with 2 GB of memory, running Windows XP SP3. The size of the PaintDotNet.exe assembly of Paint.NET 3.35 was 691 kB before and 694 kB after weaving SOK acquisition logic for five methods. The size of the SOK assembly descriptor generated for this version was 720 kB, containing 2,833 method descriptions. Regarding Paint.NET 3.36, the size of the assembly was 692 kB before and 695 kB after weaving. The size of the SOK assembly descriptor (containing 2,835 method descriptions) was 719 kB. Recording of CADProd operation was performed on five different machines, all running Windows XP SP3. Average recording duration was 18 minutes. The size of the CADProd.exe assembly was 1,620 kB before and 1,676 kB after weaving SOK acquisition logic for 86 methods. The size of the generated SOK assembly descriptor was 1,025 kB, containing 3,612 method descriptions.

4.6 THREATS TO VALIDITY

The validity of the results is threatened by several factors. Primary threats to the validity of the questionnaire results are the number of focus group experts as well as the number of participating CADComp employees. Due to the small number of questionnaire participants, (differences between) questionnaire results are not statistically significant. However, considering the total number of participants that contributed to the empirical evaluation as well as their position in their organizations, we consider the questionnaire results indicative and representative. Regarding the questionnaire answers, focus group experts agreed modestly to statement S5 compared to their answers to the other statements and to corresponding answers of CADComp employees. Since there is no obvious explanation for the drop of agreement regarding this statement, further interviews will be needed to clarify this drop.

Internal validity of our empirical evaluation is threatened by the size of $\mathcal{S}$ (i.e., $\mathcal{S}_{Paint.NET}$ and $\mathcal{S}_{CADProd}$): although both the experiment and the field study show that, without a priori knowledge of the software source code and with a relatively small set $\mathcal{S}$, valuable SOK can be acquired, performance effects of weaving acquisition logic at a large number of join points (e.g. when $\mathcal{S} = \mathcal{M}$) still have to be investigated.

External validity of the field study is threatened by the number of experiments and case studies carried out. Although the tool has been evaluated using three widely-used software applications that are based on various techniques (Windows Forms, WPF and ASP.NET), more experiments are needed to establish the robustness of Nuntia's weaving process, as well as the performance of SOK assemblies generated by the tool.

While we believe that the evaluation of our SOK acquisition and presentation technique demonstrates the utility and soundness of the technique, further research is needed to establish utility and soundness of the technique in combination with other binary instrumentation tools.

4.7 CONCLUSIONS AND FUTURE WORK

Although software vendors recognize the relevance and potential of software operation knowledge, such knowledge is frequently acquired ad hoc, impromptu and application-specific. As a consequence, acquired operation data is laborious to analyze and compare, and a vendor's existing practices, processes and products are only limitedly supported and improved by acquired SOK. We presented a technique that (1) enables software vendors to acquire SOK independent of target software, (2) allows vendors to get a uniform insight in operation of their software in the field and (3) contributes to reduction of software maintenance effort. Furthermore, we presented a prototype tool that implements this technique, and demonstrated the utility of the technique in both scientific and industrial contexts through evaluation of the tool using an eclectic set of empirical evaluation instruments. Three research questions and seven propositions were formulated to determine the utility and effectiveness of our technique.

RQ 1 — *Does the technique allow generic software operation recording?*

Although Nuntia has limitations regarding weaving heavily multi-threaded applications and supporting different platforms, SOK acquisition logic was successfully woven into diverse applications from different, independent software vendors (**P1**): Paint.NET (Windows Forms) from dotPDN LLC, ERPProd (ASP.NET) from ERPComp and CADProd (WPF) from CADComp. Furthermore, operation of these applications (of which CADProd was deployed at customer site) was successfully recorded. While a formal proof showing that unwanted or unexpected side effects will never be introduced, can not be given, no such effects were observed during local operation of resulting SOK assemblies (**P2**). Therefore, we consider this question to be answered positively.

RQ 2 *— Does the technique allow accurate recording and replay of software operation?*

As shown by analysis of SOK prints resulting from the experiment and both industrial case studies, events recorded in those prints by Nuntia were consistent with operation of corresponding software during recording (**P3**). Also, empirical evaluation shows that operation visualization and replays of Paint.NET, ERPProd and CADProd corresponded with the actual software operation during recording (**P4**). Given these results, we consider this question to be answered affirmatively. However, work is needed to mature textual representation of complex datatype variables during replay.

RQ 3 *— Does the technique support software maintenance and operation comprehension?*

Questionnaire results show that Nuntia provides valuable insight in, and increases comprehension of, in-the-field software operation as well as in-the-field end-user behavior (**P5**): the 25 questionnaire participants answered questions 1–4 with 4.3 (σ = 0.7), 4.2 (σ = 0.9), 4.0 (σ = 0.8) and 3.8 (σ = 0.9) on average, respectively. Also, those results indicate that Nuntia discloses knowledge that is not available without the tool (**P6**): the participants answered S7 with 4.2 (σ = 0.6) on average. Both propositions are also confirmed by Paint.NET experiment and CADProd case study results. To a lesser extent, the results confirm that Nuntia reduces the time needed to analyze software operation failures (**P7**): participants answered S5 with 3.6 (σ = 1.0) on average. Also, focus group experts and case study evaluation session participants stated that they expect Nuntia to reduce maintenance effort even more when it is (1) used to acquire SOK from pilot customers during the beta stages of their software, and (2) integrated in release versions of their software by default. Additional field evaluation is needed to demonstrate more significant maintenance time reductions.

Evaluation results indicate that the SOK acquisition and presentation technique is considered to effectively reduce maintenance effort, at least by the focus group experts and case study evaluation session participants. Our implementation of this technique, Nuntia, increased comprehension of in-the-field software operation, and is considered to reduce the time needed to analyze software operation failures. Furthermore, the tool disclosed a substantial failure in the CADProd application that was unknown to the CADProd development team until SOK print analysis. Nuntia recorded, visualized and replayed software

operation accurately. Therefore, we consider our technique as an adequate answer to the main research question of this paper, '*How can software maintenance effort be reduced through generic recording and visualization of operation of deployed software?*'.

Future work includes Nuntia development to increase the robustness of the weaving process and SOK assembly operation, and to diminish performance effects induced by this process even further. Also, more refined, post-weaving method selection (e.g. 'all methods that write to disk'), as well as acquisition of end-user feedback knowledge during software operation are part of future work. Additional case studies will be performed to demonstrate more significant maintenance effort reduction. Finally, integration of acquired SOK with existing practices, processes and products will be investigated.

5

Pragmatic Process Improvement through Software Operation Knowledge

ABSTRACT

Knowledge of in-the-field software operation is nowadays acquired by many software-producing organizations. Vendors are effective in acquiring large amounts of valuable software operation data to improve the quality of their software products. For many vendors, however, it remains unclear how their actual product software processes can be advanced through structural integration of such information. In this paper, we present a template method for integration of software operation information with product software processes, and present four lessons learned that are identified based on a canonical action research study of ten months, during which the method was instantiated at a European software vendor. Results show that the template method contributes to significant software quality increase, by pragmatic but measurable improvement of software processes, without adhering to strict requirements from cumbersome maturity models or process improvement frameworks.*

*This work has been published as *If the SOK Fits, Wear It: Pragmatic Process Improvement through Software Operation Knowledge* in the proceedings of the 12th International Conference on Product Focused Software Development and Process Improvement (PROFES 2011) [Van der Schuur *et al.* 2011a]. It is co-authored by Slinger Jansen and Sjaak Brinkkemper.

5.1 INTRODUCTION

Nowadays, software vendors are continuously striving for refinement and improvement of their software processes to achieve and extend competitive advantage. Simultaneously, software vendors are experienced in acquiring in-the-field operation data from their software products and services. It has become common practice, for example, to monitor software operation and identify operation failures by means of acquired operation information [Glerum *et al.* 2009, Van der Schuur *et al.* 2011b].

A wealth of software operation knowledge (SOK), gained from operation information analysis, can improve basically any software process in a product software company, such as software maintenance, software product management and customer support [Van der Schuur *et al.* 2010]. For many vendors, however, it remains unclear how software processes can be improved through integration of such information. As a consequence, acquired operation information is left untouched during execution of product software processes.

The research question we attempt to answer in this paper is '*How can product software processes effectively be improved with acquired information of in-the-field software operation?*'. To answer this question, we introduce a template method for integration of software operation information with product software processes, and present four lessons learned that are identified based on a canonical action research study of ten months, during which the method was instantiated at a European software vendor. Several prescriptive 'one-size-fits-all' software process improvement (SPI) approaches, such as the Capability Maturity Model Integration (CMMI) and ISO/IEC 15504 (SPICE), are considered as too large, too extensive and too expensive to comprehend and implement effectively within small and medium-sized companies [Conradi and Fuggetta 2002, Kuilboer and Ashrafi 2000, Pettersson *et al.* 2008, Smite and Gencel 2009]. As opposed to those approaches, the SOK integration template method presented is pragmatic and inductive, i.e., potential resulting improvements are based on the particular situation of an organization. The template method particularly describes *what* are typical operation information integration activities and concepts; a method instantiation describes *how* these activities should take place and how related concepts are involved, specific to a vendor's situation. This paper continues with placing our work in context. Next, the research approach is detailed, after which the SOK integration template method is presented (section 5.4). Section 5.5 details in-the-field instantiation of the method, as well as an valuation of integration experiences, resulting lessons learned and research limitations. Finally, conclusions and future work are presented (section 5.6).

5.2 RELATED WORK

Many research efforts cover the subject of software process improvement, but only few consider knowledge of in-the-field software operation as an instrument for improving product software processes.

For instance, Pettersson *et al.* [Pettersson *et al.* 2008] have proposed iFLAP, a process improvement framework that is to some extent similar to the SOK integration template method we presented: the framework is inductive in nature and draws on knowledge that is already residing in the organization to improve processes. However, as opposed to our method, the framework is specifically directed towards, and evaluated with, requirements engineering processes. Miler and Górski [Miler and Górski 2004] report on a case study in which they apply a risk-driven software process improvement framework in a real-life software project. Case study results demonstrate that the proposed framework is able to reveal new, previously undetected risks that provide important input for process improvement. Although various undetected risks were identified, the framework requires a high level of process description detail to be applied effectively. Iversen *et al.* [Iversen *et al.* 2004] proposed a framework for understanding risk areas and resolution strategies within software process improvement, as well as a corresponding risk management process. Although both the framework and the process are comprehensive and detailed, the framework was not evaluated through empirical studies and practical use. Moreover, it was left unclear how and to which extent vendors over time actually benefit from it.

Also, many efforts are directed to demonstrating the effectiveness of software process improvement by means of CMM(I) [Dangle *et al.* 2005, Fitzgerald and O'Kane 1999]. For example, Dangle *et al.* [Fitzgerald and O'Kane 1999] analyze the role of process improvement in the context of small organizations through an extensive case study in which CMM is applied. Based on this study, lessons learned are identified. Although one lesson is somewhat analogous to one of the lessons we have identified, it is left unclear to which extent the lessons learned of Dangle et al. are generalizable to vendors that have implemented a different maturity model, or no maturity model at all.

In this paper, we demonstrate pragmatic but measurable improvement of product software processes, and identify lessons learned that are generalizable to similar product software vendors.

5.3 RESEARCH APPROACH

To investigate how product software processes can be effectively improved with acquired information of in-the-field software operation, we designed a template method for integration of such information with product software processes, following a combination of design research and canonical action research principles [Hevner *et al.* 2004, Davison *et al.* 2004]. We conducted an extensive action research study at a European software vendor, during which the method was used to integrate software operation information acquired by the vendor with the vendor's product software processes, and therewith improve those processes. Based on study results, lessons learned are identified that may serve as a guiding substrate for similar vendors in integrating operation information with their product software processes.

5.3.1 Research Site

The action research study presented in this paper was carried out at CADComp[1], a European software vendor founded in 1990. The vendor develops an industrial design application, CADProd, which is targeted on the Microsoft Windows platform and is used daily by more than 4,000 customers in five countries. Since the start of its development in 1995, four major versions of the application have been released. Currently, CADComp employs about 100 people and is established in the Netherlands, Belgium and Romania. From December, 2009 to September, 2010, we were present at the vendor's main development site to integrate acquired SOK in the vendor's product software processes.

Before our study commenced, CADComp already acquired data of the in-the-field operation of its software product CADProd in the form of customized error reports. However, these data were not structurally analyzed, no software operation information was extracted from these data and no operation information was integrated with CADComp's existing product software processes.

5.3.2 Canonical Action Research

The canonical action research method described by Davison *et al.* [Davison *et al.* 2004] was used to structure the research activities. We attempted to satisfy each of their 'canonical action research principles':

[1]Note that for reasons of confidentiality, the names of the vendor and its software products have been anonymized.

1. *Principle of the Researcher-Client Agreement*

   The study was conducted at the European software vendor CADComp. Both the researchers and the vendor agreed on the action research approach; the vendor indicated that it is in need of an approach for improving its product software processes with the use of acquired operation data. We have agreed on a research plan that contains an overview of the shared research objectives as well as the data that was to be collected during the study (e.g. software architecture specifications, process descriptions, memos, semi-structured interviews, etc.)

2. *Principle of the Cyclical Process Model*

   The cyclical process model [Davison *et al.* 2004], originally proposed by Susman and Evered [Susman and Evered 1978], served as a basis for our work at the vendor. Structuring research activities by means of this model ensures adequate action planning, action taking and evaluation, as well as specifying what other vendors and researchers could learn from our study.

3. *Principle of Theory*

   According to Davison *et al.* [Davison *et al.* 2004], action research theory takes the following form: in situation S with salient features $F_1, \cdots, F_n$, outcomes $O_1, \cdots, O_n$ are expected from actions $A_1, \cdots, A_n$. Our (grounded) theory, as reflected by the formulated research question, is consistent with this form: we expect vendors that acquire but not analyze or integrate operation information (S), to improve their product software processes (O) by means of implementing an instantiation of the SOK integration template method (A).

4. *Principle of Change through Action*

   As stated by principle 1 and 3, both the researcher and the vendor were motivated to change the situation at the vendor in terms of operation data and information use. Planned actions were designed and taken to achieve the defined objectives (see section 5.5).

5. *Principle of Learning through Reflection*

   Both the observations made as well as the actions taken were reported to, and evaluated with the vendor's employees and management. The researcher and vendor reflected outcomes of the study by means of semi-structured interview sessions (see section 5.5).

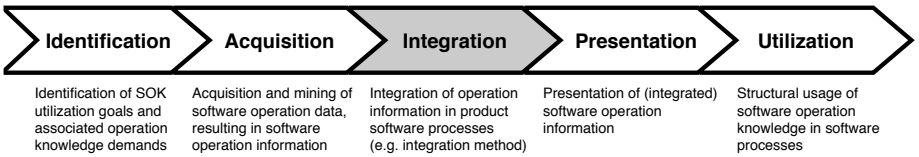


Figure 5.1 Integration of operation information is part of the SOK life cycle [Van der Schuur *et al.* 2010]

Adhering to these principles assisted us in establishing pure validity of our research approach, which contributes to realistic repetition of the study at similar software vendor sites.

5.4 SOK INTEGRATION

We have developed a template method to facilitate integration of acquired SOK into existing product software processes (*target processes*). The method is designed to guide vendors in (1) identification of relevant and valuable operation information, (2) analysis of target processes and their integration environment, (3) integration of selected information in, and transformation of, target processes and (4) presentation of integrated operation information.

When applied successfully, the method enables software vendors to increase the extent to which their practices, processes and tools are supported by SOK, which may result in directed software engineering, informed management and more intimate customer relationships. Figure 5.3 depicts the method as a Process-Deliverable Diagram (PDD) [Van de Weerd and Brinkkemper 2008].

In the context of the SOK framework [Van der Schuur *et al.* 2010], the SOK integration template method is an implementation of the SOK integration process, following identification [Van der Schuur *et al.* 2010] and acquisition [Van der Schuur *et al.* 2011b] processes, and preceding presentation and utilization processes (see figure 5.1). In the SOK acquisition process, software operation data are acquired, mined and analyzed, resulting in software operation information that forms input of the method. After acquisition, operation information is presented on carriers determined during application of the method; software operation knowledge resulting from interpretation of such information is used with the frequency determined during method application.

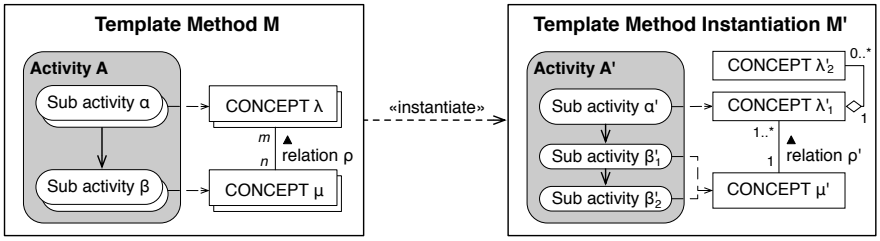


Figure 5.2 Template method instantiation

5.4.1 Instantiation

The SOK integration template method can be used by software vendors that attempt to integrate operation information with product software processes. For each target process, a new method instance is created, each with corresponding object and activity instantiations. The template method prescribes *what* (rather than *how*) activities and concepts are to be implemented by software vendors. Therefore, the method is only composed of *open* activities and concepts [Van de Weerd and Brinkkemper 2008], which should be instantiated with standard, open or closed equivalents. To anchor process improvement in the organization, vendors should instantiate both template activities and concepts consciously and diligently, taking into account daily practices. The method can be typically applied iteratively, continuously, and potentially in parallel with other method instantiations. The method's application duration and frequency depend on INTEGRATION RESOURCES and environment. One or multiple people are responsible for successful execution of (sub) activities. We assume that both the SOFTWARE OPERATION INFORMATION as well as one or more TARGET PROCESSES are available and accessible, before instantiating the method. An instantiation example is visualized in figure 5.2.

5.4.2 Concepts

The eight template concepts the SOK integration template method refers to, depicted at the right side of the PDD (see figure 5.3), are detailed in table 5.1.

Each of the concepts referred to by the method, corresponds to at least one of the (sub) activities of which the template method is composed, which are detailed in the next section.

Concept Name	Concept Description
ACTOR	A human who demands and utilizes SOFTWARE OPERATION INFORMATION with a certain frequency, potentially visualized by a CARRIER, and participates in one or more TARGET PROCESSES
CARRIER	Medium that can convey and visualize SOFTWARE OPERATION INFORMATION
INTEGRATION EVALUATION	A systematic determination of merit, worth, and significance of the performed integration of SOFTWARE OPERATION INFORMATION using criteria against the set of defined INTEGRATION OBJECTIVES involved
INTEGRATION OBJECTIVE	Goal of integration of SOFTWARE OPERATION INFORMATION with a TARGET PROCESS involving one or more INTEGRATION REQUIREMENTS
INTEGRATION REQUIREMENT	A SOFTWARE OPERATION INFORMATION integration necessity, demanding an amount of INTEGRATION RESOURCES
INTEGRATION RESOURCE	The (human) resources available for performing integration of SOFTWARE OPERATION INFORMATION
SOFTWARE OPERATION INFORMATION	Information resulting from data mining and abstraction of software operation data acquired from software operating in the field, possibly presented on a CARRIER
TARGET PROCESS	A collection of related, structured activities, tasks, tools and ideas that produce a specific service or product for (a) customer(s), possibly dependent on other processes or activities, with which SOFTWARE OPERATION INFORMATION is integrated

Table 5.1 Concepts of the SOK integration template method

5.4.3 Activities

The method's activities and sub activities, depicted at the left side of the PDD (see figure 5.3) are detailed below.

1. Operation information selection

In the first activity of the method, a selection of relevant operation information resulting from data mining and abstraction of software operation data is made. First, the TARGET PROCESS is analyzed (*Analyze target process*). Questions like 'How does the process actually work?', 'How is the process used?', and 'What are process dependencies?' are answered during this activity. *Analyze target process* may be performed from a specific perspective (e.g. human interaction, data dependencies, etc.), which results in a comprehensive view of the process. Based on this process analysis, INTEGRATION OBJECTIVES are determined (*Determine integration objectives*). In this activity, integration incentives and goals are identified, for example by defining the role and functioning of the TARGET PROCESS after integration. Second, software operation information demands of the vendor are identified (*Identify operation information demands*). Next, operation infor-

mation that is considered relevant and valuable to integrate with the TARGET PROCESS, is selected (*Select relevant operation information*)[2], based on the process analysis results and identified operation information demands.

2. *Integration requirements identification*

   First, current and future actors involved with the TARGET PROCESS and acquisition, integration or presentation of operation information are identified (*Identify SOK actors*). Second, it is estimated how often SOK resulting from integration or presentation of operation information will be used by ACTORS within the TARGET PROCESS after integration (*Estimate SOK utilization frequency*). Third, CARRIERS for presentation of operation information integrated with the TARGET PROCESS are determined (*Determine operation information carriers*). Finally, based on the sub activities prior to *Identify integration requirements*, INTEGRATION RESOURCES and INTEGRATION REQUIREMENTS for effective integration of acquired SOFTWARE OPERATION INFORMATION are determined. Resulting requirements serve as input for the subsequent 'Operation information integration' activity.

3. *Operation information integration*

   The TARGET PROCESS is altered to allow integration of relevant SOFTWARE OPERATION INFORMATION (*Integrate software operation information*) selected in 'Operation information selection'. Integration of selected operation information is dependent on, and constrained by, the available INTEGRATION RESOURCES identified earlier (e.g. external data sources, time, people, knowledge, etc.) Operation information integration results are evaluated in the second activity (*Evaluate integration results*). If, based on the subsequent INTEGRATION EVALUATION, can be concluded that INTEGRATION OBJECTIVES are met, the result of this activity (and therewith of the SOK integration template method) is a process that is effectively supported by acquired operation information. Otherwise, the method is reinitiated by starting the 'Operation information selection' activity.

In the following section, we demonstrate instantiation of the template method.

[2] If operation information is missing, application of the method can be interrupted to first iterate through the identification and acquisition phases again (see figure 5.1), and therewith make sure that desired information is available and accessible.

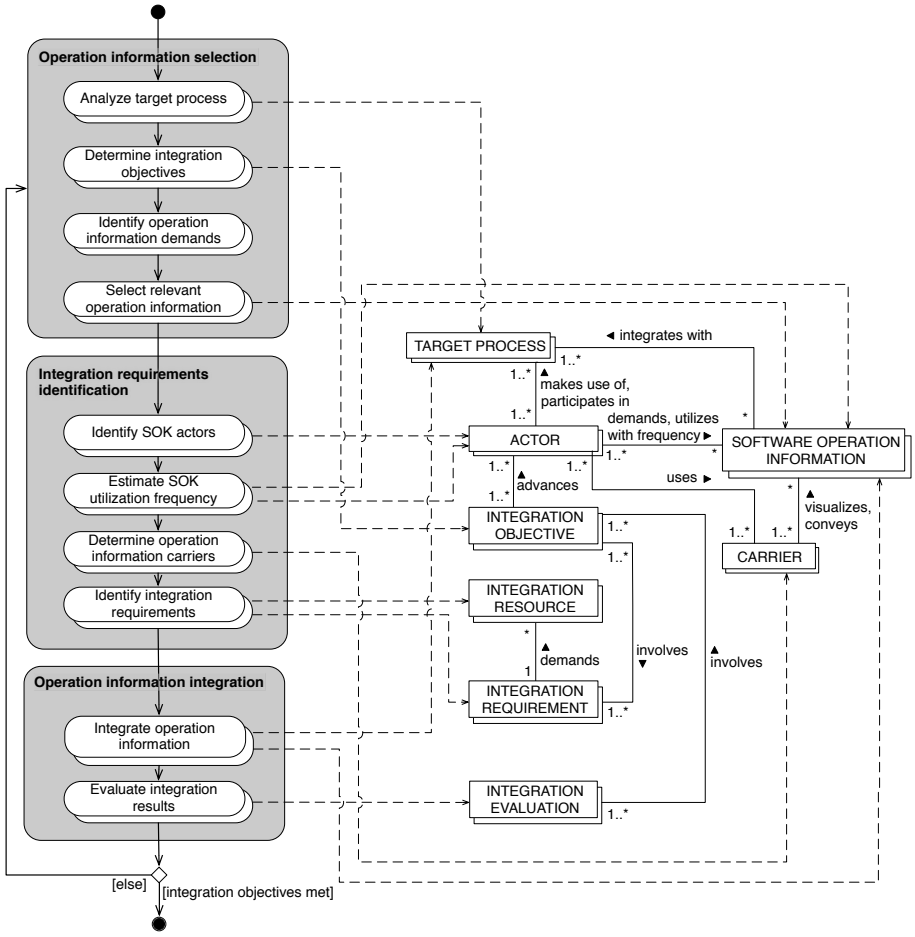


Figure 5.3 SOK integration template method

5.5 THREE PRAGMATIC IN-THE-FIELD METHOD INSTANTIATIONS

CADComp uses its own software operation data mining and analysis tool called Denerr, to extract operation information from the acquired CADProd operation data. Denerr was deployed on CADComp's intranet, and is accessible to all CADComp employees. The tool presents acquired error reports, and provides comprehensive data filtering functionality allowing developers and maintainers to define constraints on each of the error report properties, for example to analyze a particular set of error reports. The SOK integration template method was used to integrate operation information extracted by the Denerr tool in three of CADComp's product software processes: (1) software maintenance, (2) software product management and (3) customer support. Each process was selected based on CADComps needs expressed by its CEO, and corresponds to a particular SOK framework perspective [Van der Schuur *et al.* 2010].

Tables 5.2 and 5.3 respectively list how the template activities and concepts of the integration method were instantiated for each process. As part of the integration process, adjustments were made to the Denerr tool and to CADComp product software processes. CADComp employees involved in the target processes were introduced to new Denerr functionality by means of e-mails describing the new functionality, meetings of both software development and customer support departments during which new functionality was presented, and short hands-on evaluation sessions during which new functionality was demonstrated to and evaluated by particular employees. The SOK integration template method activities were performed in parallel. All integration activities were performed by the researchers and were overseen by the CADComp CEO.

5.5.1 Observations

As stated in table 5.2, weekly one-hour 'Denerr harvest meetings' were introduced to frequently analyze and delegate received error reports six months after initiation of our study. The meetings were organized with two maintenance team leaders and one researcher, and were led by the product manager made responsible for all Denerr-related issues. We attended the first three meetings in preparation of the interviews. With around 8200 error reports received since the start of error report acquisition seven months earlier, aims of the first meeting were to (1) investigate which error reports could be considered old or ir-

Template activity	Target process (corresponding perspective)					
	Software maintenance (development)		**Software product management** (company)		**Customer support** (customer)	
	Time period	Activity instantiation	Time period	Activity instantiation	Time period	Activity instantiation
Analyze target process	12/09–01/10	Software maintenance is dependent on testing, customer support, customer training and sales feedback: error reports are just acquired and stored, progress regarding repairing software failures during software maintenance is not registered. CADProd developers are not used to involving end-user error reports in their maintenance work	01/10	The software product management process relies on operation information stored in customer support (CRM) software, as well as end-user feature requests and bug fix wishes reported by salesmen in planning features and bug fixes for upcoming releases. Error reports are not taken into account by product management in the context of these activities	01/10	Customer support is used to answer end-user questions, aid customers with software usage, identify software failures and sustain customer contact. Customer supporters have limited knowledge of the in-the-field operation of their software at customers, and are dependent on the customer's willingness to explain its situation and reason for calling
Determine integration objectives		Integration objectives (e.g. accelerated software maintenance) based on analysis of the imperfections of the current software maintenance process (e.g. process dependencies slowing down the maintenance process)		Integration objectives (e.g. directed software product (release) management) were based on analysis of the imperfections of the current software product management process (e.g. limited use of operation information)		Integration objectives (e.g. increased customer intimacy) were based on analysis of the imperfections of the current customer support process (e.g. little a priori knowledge of customer's reason for calling)
Identify operation information demands		No significant demand identification was performed. Software engineers requested for maintenance status information of error reports by their own efforts		During evaluation of the Denerr tool, product managers and CEO requested for aggregated operation information and software operation trends		Through short talks with customer supporters, operation information supporting identification of particular customers was identified as information requested for
Select relevant operation information		Error report meta information was identified as relevant information, based on information demands of software engineers as well as analysis of the current software maintenance process		Aggregated error report properties and software operation trends were identified as relevant operation information based on information demands of product managers as well as analysis of the current software product management process		Operation information which enables identification of customers was identified as relevant operation information based on information demands of customer supporters as well as analysis of the current customer support process
Identify SOK actors	01/10	Actors were selected from the employees that requested for operation information, or were appointed by CADComp's CEO	02/10	Actors were selected from the employees that requested for operation information, or were appointed by CADComp's CEO	01/10–02/10	Actors were selected from the employees that requested for operation information, or were appointed by CADComp's CEO
Estimate SOK utilization frequency		By analyzing the frequency of current software maintenance process activities, it was estimated that SOK would be used at least once a week		By analyzing the frequency of current software product management process activities, it was estimated that SOK would be used at least every Scrum sprint		By analyzing the frequency of current customer support process activities, it was estimated that SOK would be used potentially with every customer support call
Determine operation information carriers		Based on (meta) operation information demands and feedback from CADComp employees, a Denner error report list view with status information was selected as carrier		Based on (meta) operation information demands and feedback from CADComp employees, a Denner aggregated error report view was was selected as carrier		Based on (meta) operation information demands and feedback from CADComp employees, Denner customer-specific error report views were selected as carrier
Identify integration requirements		Based on results of previous activities as well as discussions with CADComp's CEO, creating and introducing error report labeling functionality were identified as integration requirements.		Based on results of previous activities as well as discussions with CADComp's CEO. Also, it became clear that creating and introducing error report aggregation functionality were identified as integration requirements. Also, it became clear that frequent meetings had to be organized for analyzing, discussing and delegating aggregated error reports		Extending the error report format and Denerr functionality with customer-specific elements were identified as integration requirements, based on results of previous activities as well as discussions with CADComp's CEO
Integrate operation information	02-10–03/10	Denerr was extended to show error report status, people responsible were assigned and team leader responsibilities were expanded	03/10–09/10	Various graphs have been designed and implemented (e.g., error cause modules, error report submission time, etc.), as well as an aggregation view containing the most frequent occurring errors. Also, 'Denerr harvest meetings' were introduced to frequently analyze and delegate received error reports	02/10–06/10	IP-based location determination was implemented by means of IP geolocation library, customer-specific error report analysis view was created, integration with external tools was realized
Evaluate integration results		After deployment of new Denerr functionality, short evaluation talks were held with the involved software engineers		After deployment of new Denerr functionality, short evaluation talks were held with the involved product managers		After deployment of new Denerr functionality, short evaluation talks were held with the involved customer supporters

Table 5.2 Template activity instantiations

Template concept	Target process (corresponding perspective)		
	Software maintenance (development)	**Software product management** (company)	**Customer support** (customer)
ACTOR	2 software engineers, 4 team leaders, 3 product managers	3 product managers, CEO	3 customer supporters, 2 software engineers
CARRIER	Error report list view with status information	Statistics, graphs, aggregation error reports view	Customer-specific error report view, acquisition, mining and analysis of subjective operation information (e.g. end-user comments)
INTEGRATION EVALUATION	Employees suggested that error reports could (and should) be automatically be assigned a status, particularly when at least 95% of the equivalent error reports is assigned one and the same status, to streamline maintenance work and reduce repetitive administration tasks. Also, it was suggested to tighten integration of acquired operation knowledge with the software maintenance process by keeping track of error reports status change(r)s over time, to analyze past and delegate future maintenance tasks. Both ideas were implemented. Finally, it was suggested to integrate error report data with bug tracking software, to align error reports with identified bugs.	Initially, the graph- and aggregation views were always generated for all error reports. Later, it appeared that more specific queries were requested for generating these views, resulting in views generated for error reports with a particular build number, status or IP address. Therefore, both views were implemented as a search result view. Also, it appeared convenient to delegate errors to teams using 'error report bundle' URLs instead of URLs to individual reports: teams are provided with insight in the scope and 'in-the-field severity' of software failures. 'Bundle URLs' were created to support this form of communication.	A basic customer view was first accessible via the detail view of each individual error report. Supporters made very little use of this customer view. A global customer search was implemented to alleviate the task of associating customer support details with operation information and tighten integration of operation information in the customer support process. Also, it was suggested to extend CADComp's Salesforce 'customer prepsheet' with operation information of the particular customer, to provide salesmen and supporters with insight in the customer's recent in-the-field software operation
INTEGRATION OBJECTIVE	Increase error report status awareness; enable error report status updating and monitoring; faster software maintenance	Gain insight in trending error report characteristics; directed software product (release) management	Gain insight in software operation at particular customers; increased customer intimacy
INTEGRATION REQUIREMENT	Labeling error reports upon receive with 'New' status, visualize labels within Denerr tool, implement label update functionality	Adding aggregated error report view and trending graphs to Denerr tool	Extending error report format and Denerr mining, analysis and reporting functionality (include customer profile and comment data)
INTEGRATION RESOURCE	1 software engineer, 3 team leaders, 1 product manager	1 software engineer, 3 team leaders, 1 product manager	1 software engineer, 3 team leaders, 1 product manager
SOFTWARE OPERATION INFORMATION	Error report status labels	Aggregated operation information, software operation trends	Operation information which enables identification of customers, such as IP address, customer and user name, license number, etc.
TARGET PROCESS	Software maintenance	Software product management	Customer support

Table 5.3 Template concept instantiations

relevant and put their status to 'Ignore', and therewith get a recent, realistic view of the status of all error reports, (2) delegate each aggregated report in the resulting top 10 of aggregated error reports to a software development team, and (3) investigate reports received in the last week to identify potential bugs introduced recently.

During the second and third meeting, respectively, the status of tasks identified during first harvest were discussed again with the team leaders, and reports with automatically assigned statuses (see table 5.3) were verified to check if the correct status was assigned. Based on our attendance of the meetings, three main observations were made. First, during analysis of the error reports, employees were surprised about the amount of reports being submitted, particularly from versions that were considered old versions by the employees. As a consequence, questions like '*When can we ignore error reports originating from a particular release build?*' and '*To which extent is it desirable to see the number of error reports received from a new software product significantly increase month over*

Interviewee type	Software maintenance	Software product management	Customer support	Years experience in IT (average)
Senior supporters			3	6.3
Senior software engineers	2			12
Team leaders	3			8.5
Product managers		3		16.5
CEO		1		25

Table 5.4 Interviewees per target process

month?' were discussed. Second, although employees understood the significance of the reports ('*Those error reports represent the unhandled exceptions that are experienced by our end-users*)', occasionally, insufficient data was available to gain a clear understanding of an end-user's software operation. As a result, end-user comments accompanying the error reports were frequently analyzed to identify the usage history and goals of end-users. Also, team leaders formulated more accurate operation information demands. Third, we observed a demand for fine-grained report analysis. After aggregation and statistic views were implemented for all error reports, employees desired to show these views only for reports originating from release builds, internal builds and per (build) version. The aggregation view was used to quickly identify which bugs were not under investigation for the current sprint. Bug fix work items were created, and engineers were managed based on the aggregation view.

5.5.2 Experience Evaluation

Twelve semi-structured interviews consisting of 34 questions divided over seven sections[3] (*Integration Objectives*, *Process Improvement*, *Integration Challenges*, *Return On Investment*, *Future*, *Lessons Learned* and *Final Remarks*), were performed after application of the SOK integration template method. Interviews were conducted with CADComp employees that are involved in a particular product software process (see table 5.4), to reflect on the SOK integration process and identify lessons learned. The method, tables 5.2 and 5.3 as well as observations of the attended Denerr harvest meetings served as input for the interviews. The interviews took 1.5 hour on average and were conducted over

[3]Interview questions can be found in appendix B.

Area	Software maintenance	Software product management	Customer support
Integration objectives	Quickly identify software failures most customers are experiencing frequently, faster improve software quality and performance, increase customer satisfaction	Gain more precise insight in software failures and weak spots in the software code base, efficiently increase software quality	Increase customer intimacy, increase efficiency of product software processes, get insight in software usage of customers that do not call for support
Process improvements	Software maintenance (e.g. bug prioritization): time was saved because software failures are faster reproducible, better insight in and awareness of in-the-field software operation and quality was gained	Software maintenance, software product management: processes have been accelerated because software failures are bundled (clustered) and prioritized automatically and discussed on a weekly basis (instead of manual, subjective bundling)	Software maintenance, customer support: more detailed information of in-the-field software operation is available which helps to faster determine failure causes in collaboration with the software development department
Integration challenges	Procrastination of developers during identification and reparation of software failures based on software operation information, coping with large amounts of acquired operation information (identifying what information is most relevant in which situations and cope with diversity of information during analysis)	Assigning responsibilities to employees in involving operation information in product software processes, while ensuring a balance between (1) the liberty of an employee and its team, and (2) improving the performance and efficiency of a department as a whole	Based on large amounts of acquired operation information, correctly determine what are actual causes of software failures and what additional (operation environment) information is needed to do so if those causes can not be correctly determined
Integration side effects	Operation information can be used to convince development management in taking release planning decisions, exception handling mechanism of the software was extensively refactored to increase software quality	Information on software usage is also gained, e.g. insight in a customer's software update policy can be gained	Customers feel taken seriously, especially when they are contacted after providing information regarding their software operation
Return on Investment[a]	Software quality (robustness) increase of 25%, decrease of software maintenance time of 50%.[b]	Unhandled exception occurrence decrease of 40%, customer satisfaction increase of 25%	Customer support time decrease of 50%, software quality increase of 25%
Main future challenge	Knowing what are the functional requirements of the main customer (end-user) types, and to ensure that relevant, reliable operation information is extracted while data acquisition increases	Realizing a customer-specific approach in terms of software licensing and customer support, and finding an optimal balance between steering processes through operation information, and sustaining a leading role in industry by implementing a software product vision	Making sure that operation information is used and prioritized as effective as possible, while data acquisition sources and resulting operation data amounts increase

[a] All percentages are averages of rough estimations made by interviewees

[b] Before operation information integration, software failures were frequently unreproducible and were never repaired

Table 5.5 Interview results

a period of 68 days. Interview results of the first five sections are summarized in table 5.5; lessons learned are presented separately in section 5.5.3.

Although increase of software quality was considered a significant return on investment, a particular rival hypothesis regarding software quality increase was postulated often during the action research study and the reflective interviews. Various employees pondered over the actual cause of the decrease of received error reports: had the software quality actually been improved, or was there a random downward trend in software usage (or, more particularly, error report submission)? Error report submission history (see figure 5.4) indicates that software quality actually has been improved during period our study was conducted. While the number of submitted error reports increased about linearly from February, 2009 until June, 2010, this trend was broken in September, 2010 (the drop during July and August could well caused by summer vacation). In this month, a new major release of CADProd was released and delivered to customers, causing a slight increase in number of CADProd users.

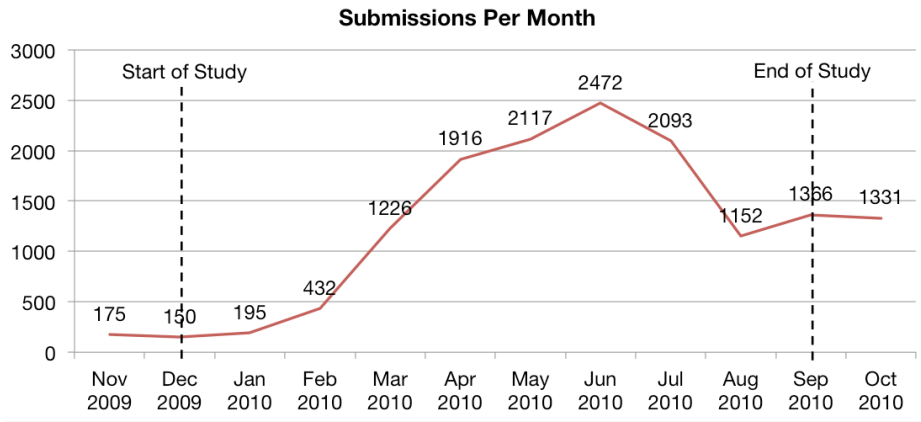


Figure 5.4 CADProd error report submission decrease of about 45% during our study

5.5.3 Lessons Learned

The lessons learned listed below have been identified based on interview session results as well as observations during our presence at the vendor.

1. Integration Processes Should be Lean

The effects of integrating operation information in product software processes should not be underestimated. Additional processes with corresponding responsibilities may be required to ensure effective and continuous integration of acquired operation information (for example, registering software maintenance tasks based this information and delegating those tasks to the right employee(s)). Vendors should ensure that additional (administrative) tasks caused by integration of operation information are handled in a pragmatic and lean way: additional administration may negate the time gain caused by integration of operation information.

2. Integration Responsibilities and Results Should Be Evangelized

An internal manager that has affiliation and experience with development-, business- and customer-related processes should be made responsible for integration of acquired operation information with those processes, since acquired operation information will not integrate automatically: during our action research study at CADComp, we observed that making integration of operation information everyone's shared responsibility, is effectively equivalent to making no one respon-

sible. Inter alia, such a manager should ensure that the potential and results of the 'SOK-supported' process both are communicated clearly and frequently: this increases awareness and acceptance of the new way of working, both among employees as well as at management level. Evangelism of SOK integration potential and results is key in integrating operation information.

3. *SOK Integration Opens Up Black Boxes*

   In line with expectations of CADComp employees, integration of acquired operation information improved software maintenance, software product management and customer support processes: the time needed to reproduce software failures was decreased, deeper insight into (and awareness of) in-the-field software operation and end-user behavior was gained, and customer satisfaction was increased. Operation information is used for prioritization of fixes for software failures that are actually experienced by end-users, which may decrease the time required for software maintenance and customer support. However, as became clear after integrating CADProd operation information with CADComp processes, unanticipated improvements may result from effective SOK integration. For example, since developers are made aware of in-the-field software operation quality, SOK integration may result in a more customer-central, pro-active development mentality ('*build what the customer will use, before the customer asked for it*'). Also, integration of operation information in product software processes may clarify and speed-up interdepartmental communication as well as communication between employees and management. Improvement areas that were unnoticed before, are highlighted as such after (and potentially as a side effect of) SOK integration.

4. *Continuous Refinement of SOK Integration Objectives and Requirements Leads to Optimization of Integration Results*

   Integration results are dependent on integration objectives and requirements. Since product software processes and activities change, as well as software operation environments and customer demands, integration objectives and requirements should be evaluated and refined continuously to correspond to both a vendor's product strategy as well as a vendor's customers needs. On the long term, software vendors should attain a balance between using operation information to steer their product software processes, and adhering to their product vision and strategy.

These lessons learned may serve as a guiding substrate for similar vendors in integrating information of in-the-field software operation.

5.5.4 Threats to Validity

The validity of the study results is threatened by several factors. First, construct validity of our study is threatened by the fact that the researchers conducting the study were involved in objects of study (e.g. product software processes implemented at CADComp): observations or conclusions could be biased. This threat is addressed by adhering to the principles for canonical action research elicited by Davison *et al.* [Davison *et al.* 2004] (see section 5.3.2). For example, the researchers and software vendor being studied agreed on a research plan describing the shared objectives and data that was to be collected during the study. Also, both the researcher and the vendor reflected upon the outcomes of the study by means of semi-structured interview sessions.

Second, primary threat to the internal validity of the study is the relation between the instantiation of the SOK integration template method at CADComp and the subsequent software quality increase perceived by CADComp interviewees. Although it is a challenge to isolate the particular influence of template method instantiation on improvement of CADComp's product software processes, we regard CADComp's error report submission history (as analyzed in section 5.5) as representative of the extent to which CADComp's software maintenance, software product management and customer support processes are improved through instantiation of the SOK integration template method.

Third, external validity is threatened by the fact that the SOK integration template method was instantiated at only one software vendor, during a limited period of time. While we acknowledge this threat, we regard the study as repeatable with the same results, presuming similar circumstances (e.g. similar operation information, processes, software vendors, etc.)

5.6 CONCLUSIONS AND FUTURE WORK

All too often in industry, software vendors acquire large amounts of valuable software operation data, without effectively using these data in advancement of their processes. Operation information extracted from operation data is not structurally integrated with product software processes, leaving vendors in the dark regarding in-the-field software performance, quality and usage, as well as end-user feedback. Vendors are in need of an approach that supports them in accomplishing such integration.

We presented a template method that aids product software vendors in (1) identification of relevant and valuable operation information, (2) analysis of target processes and their integration environment, (3) integration of selected information in, and transformation of, target product software processes, and (4) presentation of integrated operation information. During an action research study of ten months performed at a European software vendor, the template method was instantiated to improve the vendor's product software processes through integration of acquired operation information.

Evaluation of the study shows that typical product software processes like software maintenance, software product management and customer support benefit from structural integration of operation information in terms of software quality, operation knowledge and customer intimacy. Based on this evaluation, four lessons learned are identified that may serve as a guiding substrate for similar vendors, in integrating information of in-the-field software operation with their product software processes. We regard the SOK integration template method and lessons learned as an adequate early answer to the main research question of this paper, '*How can product software processes effectively be improved with acquired information of in-the-field software operation?*'. We demonstrated how product software processes can be improved pragmatically but measurably, without adhering to strict requirements from cumbersome maturity models or process improvement frameworks.

Future work will include additional action research or case studies to instantiate and evaluate the SOK integration template method in industry, and therewith further demonstrate its soundness and utility. Further research is also needed to mature the identified lessons learned towards generic guidelines or principles for effective integration of software operation information.

6

Leveraging Software Operation Knowledge for Maintenance Task Prioritization

ABSTRACT

Knowledge of in-the-field software operation is acquired by many software-producing organizations nowadays. While vendors are effective in acquiring large amounts of valuable software operation information, many are lacking methods to improve their software processes with such information. In this paper, we attempt to improve the software maintenance process by proposing a software operation summary: an overview of a vendor's recent in-the-field software operation, designed to support software processes by providing software operation knowledge. Particularly, we strive to improve prioritization of software maintenance tasks by fostering the reach of consensus between involved employees on such prioritization. Through an extensive survey among product software vendors in the Netherlands, we confirm the need for a software operation summary, and identify which crash report data are considered relevant as a basis for the summary when it is used for prioritization of software maintenance tasks. By means of a case study at a European software vendor, a software operation summary composed using crash report data is empirically evaluated. Results confirm the lack of consensus experienced between engineers and managers, and illustrate the value of the summary in fostering reach of consensus between employee roles, particularly in preparation of sprint planning and bug fixing activities.*

*This has been published as *Sending Out a Software Operation Summary: Leveraging Software Operation Knowledge for Prioritization of Maintenance Tasks* in the proceedings of the 6th International Con-

6.1 INTRODUCTION

One of the most time-consuming and challenging tasks in software maintenance is to reach consensus on the prioritization of software maintenance tasks. Many factors are involved in the process of software maintenance task prioritization, such as the number and names of customers that have reported a particular software failure, the severity and frequency of the failure as well as a software vendor's roadmap and available resources [Lehtola and Kauppinen 2006]. All too often, prioritization discussions and decisions of smaller software vendors are led by strong emotions of employees involved in the prioritization process, rather than by (f)actual knowledge. Employees may strongly focus on one particular aspect of software maintenance, and for example base their prioritization preferences on the interests and wishes of a large customer or on their image of the software code quality. Many software vendors are lacking in methods to improve such processes with objective data and knowledge, for example knowledge of in-the-field operation (and failures) of their software (i.e., software operation knowledge or SOK) [Van der Schuur *et al.* 2011c]. As a consequence, reaching consensus on software maintenance task prioritization frequently is a time-consuming process, resulting in flawed prioritization decisions, unsatisfied customers and significant technical debt. The ISO standard on the software maintenance process [ISO/IEC 2006] defines software maintenance as comprised of six activities, being *Process Implementation; Problem and Modification Analysis; Modification Implementation; Maintenance Review/Acceptance; Migration; Retirement*. In the context of this standard, software maintenance task prioritization activities as well as reaching consensus on such prioritization are covered by the *Process Implementation* activity.

The main contribution of this paper is the Software Operation Summary (SOS) concept: an overview of recent in-the-field software operation, which can be used for improvement of software processes. We improve software maintenance *process implementations* by fostering the reach of consensus in software maintenance task prioritization through this software operation summary concept.

The main research question is therefore '*Can prioritization of software maintenance tasks be improved through the concept of a software operation summary?*', which is answered through an extensive software maintenance survey held among product software vendors in the Netherlands. Based on survey results, we (1) confirm the lack of consensus on software maintenance prioritization experi-

ference on Software Process and Product Measurement (MENSURA 2011) [Van der Schuur *et al.* 2011c]. It is co-authored by Slinger Jansen and Sjaak Brinkkemper.

enced between engineering, management, and customer support, (2) establish the need for an SOS by these employee roles, and (3) identify which crash report data are considered relevant as a basis for an SOS that fosters reaching consensus on prioritization of software maintenance tasks [ISO/IEC 2006, Pigoski 1997]. To qualitatively support survey results, we conducted a case study at a European software vendor, through which we composed an SOS based on crash report data and empirically evaluated its soundness and validity.

This paper is organized as follows. Section 6.2 further details the SOS concept. Section 6.3 describes our research approach; section 6.4 details survey structure and contents. In section 6.5, survey results are analyzed, while in section 6.6 the case study results are presented. Next, we discuss research limitations (section 6.7) and place our work in context (section 6.8). Finally, conclusions and future work are presented in section 6.9.

6.2 SOFTWARE OPERATION SUMMARY

We define a software operation summary as *an overview providing knowledge of recent in-the-field software operation, based on acquired software operation information*. The operation information needed for composition of the summary can be acquired by vendors from their software operating in the field, in a generic manner [Van der Schuur *et al.* 2011b]. Both the operation information on which an SOS is based, as well as the operation knowledge provided by the summary are related to the in-the-field performance, quality or usage of the software, or to the feedback of end-users of the software [Van der Schuur *et al.* 2011c]. An abstract SOS is provided in figure 6.1.

The software operation history, which forms the main component of the SOS, can be complemented with operation meta data such as the operation history timespan and software operation objectives that have been defined. As we show later in this paper, frequent use of a software operation summary (particularly, the software operation knowledge it provides) may foster reaching consensus between employees on software process issues. For example, it helps achieving consensus on prioritization of software maintenance tasks (i.e., determining when which bugs should be fixed, by whom).

In industry, at least three types of employees are involved in software maintenance activities: customer supporters (filing bug reports based on feedback of customers experiencing in-the-field software failures), software engineers (developing and testing fixes for the reported failures) and development / product managers (guiding the maintenance process, being responsible for prioritizing maintenance tasks). Each of these employees may advocate a particular focus

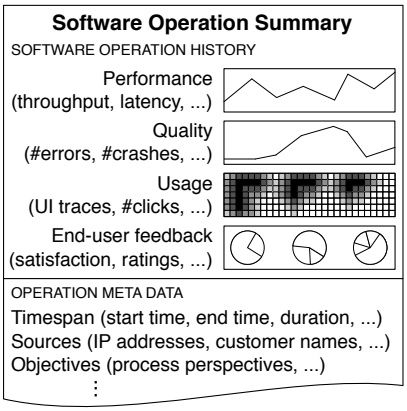


Figure 6.1 Software operation summary concept

in prioritizing software maintenance tasks, and strive to improve a particular aspect within this focus. Customer supporters may be focused on increasing the satisfaction of end-users using the software. When prioritizing software maintenance tasks, customer supporters prioritize bugs that are experienced by the most influential and demanding customers. Software engineers, on the other hand, may be concerned with the quality of their software architecture and therefore prioritize bugs that are caused by poor or non-standard software design. Third, development / product managers may strive to increase the efficiency of the maintenance process itself, by maximizing the number of completed maintenance tasks and minimizing task duplication. By mapping those three employee foci to the SOK framework [Van der Schuur *et al.* 2011c], three perspectives on software maintenance task prioritization can be observed. See table 6.1.

In this paper, it is identified which crash report data are considered relevant as a basis for an SOS that fosters reaching consensus on prioritization of corrective and adaptive maintenance tasks [ISO/IEC 2006, Pigoski 1997]. Crash report data are successfully used by Microsoft [Microsoft Online Crash Analysis 2005, Microsoft Error Reporting 2006], Apple [Mac OS X Reference Library 2010], Canonical [Ubuntu Wiki 2010], and Google [Google Breakpad 2010] to improve their software architecture quality, end-users satisfaction and maintenance process efficiency [Glerum *et al.* 2009]. Which data are considered relevant, however, is situational and depends on many factors, such as the type of product, the type of customers and the type of employee using the data.

	Engineering	Management	Support
Employee role	Software engineer	Development / Product manager	Customer supporter
Maintenance focus	Software architecture	Maintenance process	End-user
Improvement aspect	Quality	Efficiency	Satisfaction
SOK framework perspective	Development	Company	Customer

Table 6.1 Perspectives on software maintenance task prioritization

6.3 RESEARCH APPROACH

We structured the research approach using the SOK integration template method [Van der Schuur *et al.* 2011a]. The method is designed for integration of software operation information in software processes (e.g. software maintenance). The instantiated method details the main research activities and related concepts, and therewith describes how our study can be repeated [Kitchenham and Pfleeger 2002]. It is presented as a process-deliverable diagram (PDD) [Van de Weerd and Brinkkemper 2008] in figure 6.2. Significant research activities and concepts are described below and in table 6.2, respectively.

Operation information selection

The first activity is concerned with determining which SOFTWARE OPERATION INFORMATION in crash reports is considered relevant in supporting the SOFTWARE MAINTENANCE TASK PRIORITIZATION process through a SOFTWARE OPERATION SUMMARY. First, we acquired empirical understanding of this process by analyzing SOFTWARE MAINTENANCE TASK PRIORITIZATION processes in industry (see section 6.2, as well as [Van der Schuur *et al.* 2011a]) and answering questions like 'How is the process implemented in the organization?' and 'What are process dependencies?' (*Analyze target process*). Next, we used industry experiences and literature study results to identify issues with prioritization of software maintenance tasks, on which we based INTEGRATION OBJECTIVES (*Identify task prioritization issues in industry and literature*). Third, we categorized crash report data originating from various crash reporters (*Categorize crash report data*; see table 6.4) to identify SOFTWARE OPERATION INFORMATION demands of vendors acquiring crash reports. Based on the process analysis results and identified information demands, we have designed and conducted a SOFTWARE

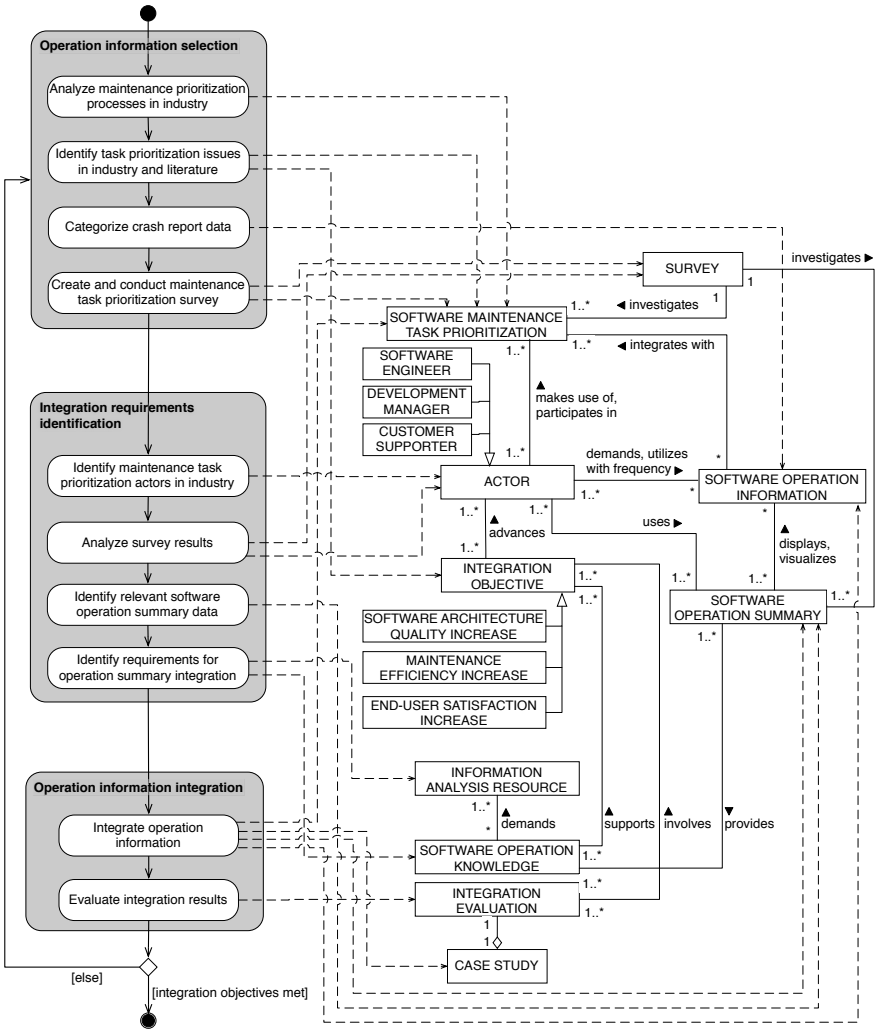


Figure 6.2 Research approach based on SOK integration template method

MAINTENANCE TASK PRIORITIZATION SURVEY (*Create and conduct maintenance task prioritization survey*; see section 6.4).

Integration requirements identification

concerns identifying requirements for successful integration of the SOFTWARE OPERATION SUMMARY with SOFTWARE MAINTENANCE TASK PRIORITIZATION processes. First, we identified ACTORS involved with SOFTWARE MAINTENANCE TASK PRIORITIZATION by visiting software vendor sites (*Identify main-*

Concept name	Concept description
ACTOR	A person who demands and utilizes SOFTWARE OPERATION INFORMATION with a certain frequency, potentially visualized by a SOFTWARE OPERATION SUMMARY, and participates in one or more SOFTWARE MAINTENANCE TASK PRIORITIZATION processes
CUSTOMER SUPPORTER	An ACTOR providing (technical) assistance with one or more software products or services
DEVELOPMENT MANAGER	An ACTOR managing SOFTWARE ENGINEERS of a software development department
END-USER SATISFACTION INCREASE	One of the INTEGRATION OBJECTIVES: an increase of the extent to which end-users believe the software available to them meets their requirements
INFORMATION ANALYSIS RESOURCE	A physical or virtual entity of limited availability needed to analyze and interpret SOFTWARE OPERATION INFORMATION and gain SOFTWARE OPERATION KNOWLEDGE
INTEGRATION EVALUATION	A systematic determination of merit, worth, and significance of the performed integration of SOFTWARE OPERATION INFORMATION using criteria against the set of defined INTEGRATION OBJECTIVES involved
INTEGRATION OBJECTIVE	Goal of integration of SOFTWARE OPERATION INFORMATION with SOFTWARE MAINTENANCE TASK PRIORITIZATION involving SOFTWARE OPERATION KNOWLEDGE; generalization of the SOFTWARE ARCHITECTURE QUALITY INCREASE, MAINTENANCE EFFICIENCY INCREASE and END-USER SATISFACTION INCREASE INTEGRATION OBJECTIVES
MAINTENANCE EFFICIENCY INCREASE	One of the INTEGRATION OBJECTIVES: an increase of the extent to which wasted time and effort in the SOFTWARE MAINTENANCE TASK PRIORITIZATION process are avoided
SOFTWARE DEVELOPER	An ACTOR concerned with facets of the software development process, primarily (but wider than) design and coding.
SOFTWARE MAINTENANCE TASK PRIORITIZATION	The process of assigning priorities to software maintenance tasks by one or more ACTORS, which is part of the *Process Implementation* activity of the ISO standard on the software maintenance process [ISO/IEC 2006]
SOFTWARE OPERATION INFORMATION	Information resulting from data mining and abstraction of software operation data acquired from software operating in the field, possibly presented on a SOFTWARE OPERATION SUMMARY
SOFTWARE OPERATION KNOWLEDGE	Knowledge of in-the-field performance, quality and usage of software, and knowledge of in-the-field end-user software experience feedback [Van der Schuur *et al.* 2010]
SOFTWARE OPERATION SUMMARY	An overview of recent in-the-field software operation. Based on recent SOFTWARE OPERATION INFORMATION, the summary provides SOFTWARE OPERATION KNOWLEDGE
SOFTWARE ARCHITECTURE QUALITY INCREASE	One of the INTEGRATION OBJECTIVES: an increase of the extent to which software is designed well, and how well the software conforms to that design

Table 6.2 Names and descriptions of significant concepts

tenance task prioritization actors in industry; see table 6.1). Next, we analyzed SURVEY results, for example to estimate how often SOFTWARE OPERATION INFORMATION would be used by the identified ACTORS (*Analyze survey results*; see section 6.5). Based on SURVEY results and prior sub activities, relevant crash report data on which a SOFTWARE OPERATION SUMMARY can be based, were identified (*Identify relevant software operation summary data*; see section 6.5.4). Finally, requirements for successful integration of the summary in SOFTWARE MAINTENANCE TASK PRIORITIZATION processes were

identified (*Identify requirements for operation summary integration*; see section 6.6). For example, a requirement could be that valuable SOFTWARE OPERATION KNOWLEDGE is gained after integration of acquired operation information, demanding INFORMATION ANALYSIS RESOURCES).

Operation information integration
concerns integration of the SOFTWARE OPERATION SUMMARY with SOFTWARE MAINTENANCE TASK PRIORITIZATION processes. We conducted a CASE STUDY at a European software vendor to compose and evaluate a SOFTWARE OPERATION SUMMARY. First, the SOFTWARE MAINTENANCE TASK PRIORITIZATION process was altered to allow integration of relevant SOFTWARE OPERATION INFORMATION selected in 'Operation information selection' (*Integrate software operation information*; see section 6.6). Next, as part of the CASE STUDY, integration results were evaluated (*Evaluate integration results*; see section 6.6). If, based on a subsequent INTEGRATION EVALUATION, can be concluded that INTEGRATION OBJECTIVES are met, the result of this activity is a SOFTWARE MAINTENANCE TASK PRIORITIZATION process that is effectively supported by acquired operation information. Otherwise, the method is reinitiated with the 'Operation information selection' activity.

This paper's main research question ('*Can prioritization of software maintenance tasks be improved through the concept of a software operation summary?'*) is answered based on evaluation of the following six propositions:

P1 A software operation summary is expected to integrate with current software maintenance practices.

P2 A software operation summary is expected to increase knowledge of software architecture quality.

P3 A software operation summary is expected to increase knowledge of end-user satisfaction.

P4 A software operation summary is expected to increase knowledge of maintenance process efficiency.

P5 A software operation summary is expected to foster achieving consensus on software maintenance task prioritization.

P6 A software operation summary is expected to reduce the time needed for software maintenance task prioritization.

The propositions are evaluated based on results of the software maintenance task prioritization survey we conducted.

Sections	Questions	Propositions
General	1. How many people are employed at your company? [Less than 5; 5 to 10; 10 to 20; 20 to 50; 50 to 100; 100 to 200; More than 200]	N/A
	2. How many end-users are using software that is produced by the company you are employed by? [Less than 50; 50 to 200; 200 to 500; 500 to 2,000; 2,000 to 10,000; 10,000 to 100,000; More than 100,000]	N/A
	3. How many crash reports are received weekly by the company you are employed by? [Less than 50; 50 to 200; 200 to 500; 500 to 2,000; 2,000 to 5,000; 5,000 to 10,000; N/A]	N/A
	4. What is your role within the company you are employed by? [Engineering (software developer, software engineer, software architect, etc.); Management (Development manager, product manager, etc.); Support (customer supporter, help desk employee, etc.)]	N/A
	5. How many years of experience do you have in the field of information technology? [0..75]	N/A
Current situation	6. How much time do you and your colleagues spend weekly on prioritization of software maintenance tasks? [Less than 1 hour; 1 to 5 hours; 5 to 10 hours; More than 10 hours]	P1
	7. How frequent do you experience a lack of consensus on prioritizing software maintenance tasks with each of the following colleagues? [Never (1); Rarely (2); Monthly (3); Weekly (4); Daily (5); N/A] • Engineering • Management • Support	P1
	8. To which extent does knowledge of your software architecture play a role in prioritizing software maintenance tasks within your organization? [1 (Very minor role); 2; 3; 4; 5 (Very major role); N/A]	P1
	9. To which extent does knowledge of end-user satisfaction play a role in prioritizing software maintenance tasks within your organization? [1 (Very minor role); 2; 3; 4; 5 (Very major role); N/A]	P1
	10. To which extent does knowledge of the efficiency of the maintenance process within your organization play a role in prioritizing software maintenance tasks within your organization? [1 (Very minor role); 2; 3; 4; 5 (Very major role); N/A]	P1
Expectations	11. How often could you, in your current role, well use a software operation summary? [Never (1); Rarely (2); Monthly (3); Weekly (4); Daily (5); N/A]	P1
	12. With which of your activities is a software operation summary of most use to you? [...]	P1
	13. To which extent do you expect that a software operation summary increases your knowledge of each of the following: [Certainly not (1); probably not (2); Possibly (3); Probably (4); Certainly (5); N/A] • Software architecture quality • End-user satisfaction • Maintenance process efficiency	P2 P3 P4
	14. To which extent do you expect to save time on prioritization of software maintenance tasks with a software maintenance summary? [Certainly not (1); probably not (2); Possibly (3); Probably (4); Certainly (5); N/A]	P6
	15. To which extent do you expect information contained in a software operation summary to foster the reach of consensus on prioritization of software maintenance tasks, with each of the following colleagues? [Certainly not (1); probably not (2); Possibly (3); Probably (4); Certainly (5); N/A] • Engineering • Management • Support	P5
Crash report data	16. Indicate which data types in your opinion contribute most to successful execution of the following activities: [List of data types listed in table 6.4] • Determining which bug will result in the largest increase of software architecture quality, once fixed • Determining which bug will result in the largest increase of end-user satisfaction, once fixed • Determining which bug will result in the largest progress in the software maintenance process, once fixed	P2 P3 P4

Table 6.3 Survey questions and propositions

Vendor name		Microsoft [Microsoft Online Crash Analysis 2005, Microsoft Error Reporting 2006]	Apple [Mac OS X Reference Library 2010]	Canonical [Ubuntu Wiki 2010]	Google [Google Breakpad 2010]	ERPComp	CADComp
Crash reporter form		Operating system service			External library	Product software feature	
Category	**Unified data type**	**Data type instance**					
Software	Application name	Identifier	General AppName	Package	–	–	Cause
	Build version	Build Info	Version Information	StackTrace, Package	MODULE name	–	BuildVersion
	Code file and line number causing the crash	Backtrace	–	StackTrace	FILE name	–	CauseLocation
	Code file and line number presenting the crash (to end-user)	Backtrace	–	–	FILE name	–	–
	Database	–	–	–	–	Database	–
	Edition	–	–	–	–	–	Edition
	Error code	Exception Codes	Exception Code	–	–	–	–
	Error message	–	–	–	–	Exception details	–
	Error type	Exception Type	–	–	–	Exception Type	ExceptionType
	Last user action (click, command, etc.)	Backtrace	–	StackTrace	FUNC name	Action name	–
	Last operation context (page, screen, etc.)	–	–	–	–	Page name	–
	Localization	–	–	–	–	–	Localization
	Memory address	Exception Codes	Exception Address	–	FUNC address	–	MemoryLocation
	Module causing the crash	Backtrace	–	StackTrace	–		CauseModule
	Module presenting the crash (to end-user)	Backtrace	–	–	–	Page name	–
	Operation environment (operating system, browser)	OS Version	System Information	DistroRelease, ProcEnviron	MODULE operatingsystem	OS, Browser, page data	–
	Process name	Process Identifier	–	–	–	–	–
	Processor architecture	Code Type	System Information	–	MODULE architecture	–	ProcessorType
	Session type	–	–	–	–	Session data	–
	Thread name	Crashed Thread	–	–	–	–	–
	Version	Version	Version Information	Package	–	–	–
End-user	Comments accompanying report	–	–	–	–	–	EnduserComments
	Company name	–	–	–	–	Customer	CompanyName
	Country	–	–	–	–	–	SourceCountry
	IP address	IP address	–	–	–	IP address	IPAddress
	Language	–	–	–	–	–	Language
	License number	–	–	–	–	–	LicenseNumber
	User name	–	–	–	–	User	UserName

Table 6.4 Categorization of crash report data types

6.4 MAINTENANCE TASK PRIORITIZATION SURVEY

As part of the *Operation information selection* activity described in section 6.3, a software maintenance task prioritization survey was designed and conducted to evaluate the six propositions and to establish which operation information from crash reports is considered relevant by the three employee roles. The survey was reviewed and tested by peer researchers and target respondents before publication. To acquire survey respondents, the survey was announced to our educational and professional networks, for example via an expert focus group consisting of chief technology officers, product managers and senior team leaders from industry. Also, the survey was noticed and advertised by press in the Netherlands [Nap 2011]. The survey was composed of 16 questions divided over four sections (see table 6.3):

General

The survey started with five questions to identify the respondent, as well as the software vendor the respondent is employed by. The first three questions were asked to get insight in the size of the company the respondent is employed at, in terms of number of employees, end-users and received crash reports. The answer sets of these questions (as denoted between square brackets in table 6.3) were deducted from the definition for small and medium-sized enterprises (SMEs) of the European Commission [OOPEC 2005]. The answer options of question 3 contained a 'Not Applicable' option for vendors not receiving any crash reports. Question 4 was asked to identify the role of the respondent within its employing company and could be answered with three options that correspond to the roles in table 6.1. Finally, question 5 was an open question querying the respondent's experience in information technology. This section's questions correspond to none of the propositions.

Current situation

This section is composed of five questions that were asked to establish the situation at the respondent's organization regarding prioritization of software maintenance tasks, and identify potential improvement areas. The first two questions respectively queried the amount of time that is spent on software maintenance prioritization tasks by the respondent, as well as the frequency with which the respondent experiences a lack of consensus with colleagues on prioritizing software maintenance tasks. Questions 8, 9 and 10 each refer to a particular prioritization focus and improvement aspect in table 6.1 and were asked to confirm the importance of the main-

tenance foci within the respondent's company. Each of these questions correspond to an identical set of answer options that is equivalent to a five-point Likert item (see table 6.3). All questions in the *Current situation* section were asked to evaluate proposition **P1**.

Expectations

As an introduction to the questions in this section, the software operation summary concept was introduced to respondents before asking the questions by providing a basic definition of the concept, as well as a description of three of its goals (providing knowledge of (1) software architecture quality, (2) end-user satisfaction, (3) maintenance process efficiency). Five questions related to expected or potential use of a software operation summary were asked in this section to establish the need for, and potential of such a summary based on crash report data. Particularly, questions 11 and 12 were asked to establish expected integration of a software operation summary with the respondent's current activities and requirements. We chose to let question 12 be an open question to prevent bias of respondents, which, for example, could be induced by providing a predefined set of activities. Questions 13, 14 and 15 query the respondent's expectations concerning the effects of using a software operation summary, respectively in terms of knowledge increase, time saving and consensus reach. The answer options of these three questions represent a five-point Likert item (see table 6.3). Questions 11 and 12 were asked to evaluate proposition **P1**, question 13 to evaluate propositions **P2**, **P3** and **P4** and questions 14 and 15 were asked to evaluate propositions **P5** and **P6**, respectively.

Crash report data

The last section of the survey is concerned with what crash report data are considered relevant input for prioritization of corrective and adaptive software maintenance tasks. A list of answer options in the form of unified crash report data types was assembled based on crash report data types of existing, widely-used and widely-known crash reporting techniques. See table 6.4.

Crash reporting techniques from Microsoft [Microsoft Online Crash Analysis 2005, Microsoft Error Reporting 2006], Apple [Mac OS X Reference Library 2010], Canonical [Ubuntu Wiki 2010] and Google [Google Breakpad 2010], as well as those of two European software vendors called ERPComp and CADComp (actual vendor names have been anonymized

for confidentiality reasons), were analyzed and compared to assemble the answer list with data types that was presented to respondents as an introduction to this section. ERPComp produces an online ERP solution that is used by about 17,000 customers; CADComp produces an industrial drawing application targeted on the Microsoft Windows platform and is used by more than 4,000 customers in five countries. Both vendors are headquartered in the Netherlands and were visited on-site.

To ease understanding and answering of this question for respondents, data types were categorized into two categories: software and end-user. One additional answer option 'Other' was added to the list of 28 crash report data type answer options, to allow respondents to add additional data types matching their crash report data structure. Question 16 was asked to evaluate propositions **P2**, **P3** and **P4**.

As listed in table 6.3, questions 7, 13, 15 and 16 had to be answered for three question components: questions 7 and 15 had to be answered for each of the software maintenance task prioritization perspectives (Engineering, Management and Support; see table 6.1) and question 13 and 16 had to be answered for each of the prioritization foci and improvement aspects of these perspectives. These questions were designed as such to expose relations between employee roles, maintenance foci and improvement aspects in reaching consensus on prioritization of software maintenance tasks.

6.5 ANALYSIS OF SURVEY RESULTS

In total, 136 respondents from about 72 different software-producing companies[1] participated in our survey (n = 136). All of the respondents completed section *General*, 111 (81.6%) completed *Current situation*, 97 (71.3%) completed *Expectations* and 79 (58.1%) completed the final section of the survey, *Crash report data*. Those percentages should be taken into account in further analysis of survey results, which is provided in the following sections.

6.5.1 General

Most respondents are employed by a vendor that employs more than 200 people (49.3%) or 20 to 50 people (15.4%). Almost three-quarter of the respondents (72.1%) is employed by a vendor that serves more than 100,000 end-users (35.3%), 10,000 to 100,000 end-users (18.4%) or 2,000 to 10,000 end-users

[1]Survey respondents participated with 72 unique IP addresses (addresses that only differed on third or fourth octet were considered identical).

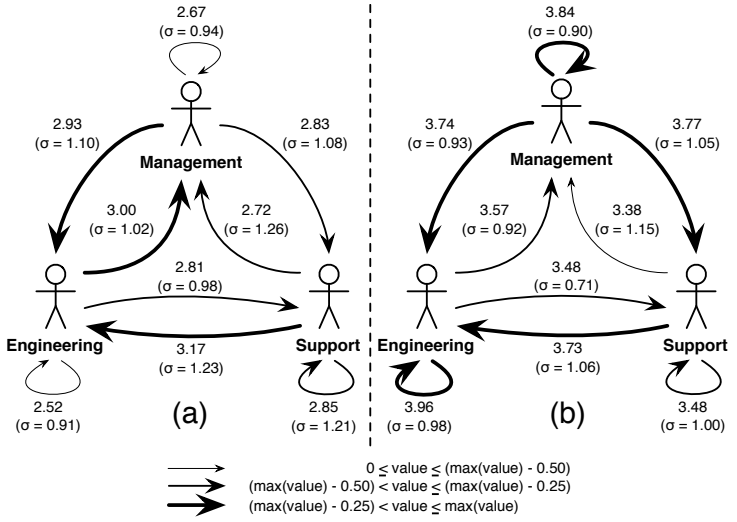


Figure 6.3 (a): average frequency with which lack of consensus is experienced between the different employee roles (question 7); (b): average extent to which the employee roles expect SOS information to foster the reach of consensus (question 15). Thicker arrows indicate higher frequencies and expectations

(18.4%). Most vendors (66.9%) receive less than 50 crash reports (52.9%) or between 50 and 200 (14.0%) crash reports per week (19.1% of the respondents answered 'N/A' to question 3). Respondents were evenly distributed over employee roles: 28.7% identified its role as part of Engineering (12.4 years of experience on average, $\sigma = 7.6$), 36.0% as part of Management (15 years of experience on average, $\sigma = 7.2$ years), and 35.3% as part of Support (9.6 years of experience on average, $\sigma = 8.9$ years). In total, respondents have 1676 years of experience (12.3 years on average, $\sigma = 8.3$ years).

6.5.2 Current Situation

Of the 111 respondents that have answered the first question of this section, most (55.9%) spend 1 to 5 hours weekly on prioritization of software maintenance tasks. 24.3% indicated to spend less than 1 hour on these tasks weekly, while 10.8% indicated to spend more than 10 hours weekly on prioritization of software maintenance tasks.

Figure 6.3a visualizes the lack of consensus experienced by the responding software engineers, customer supporters and managers, as inquired through question 7. Lack of consensus is most frequently experienced by (1) support

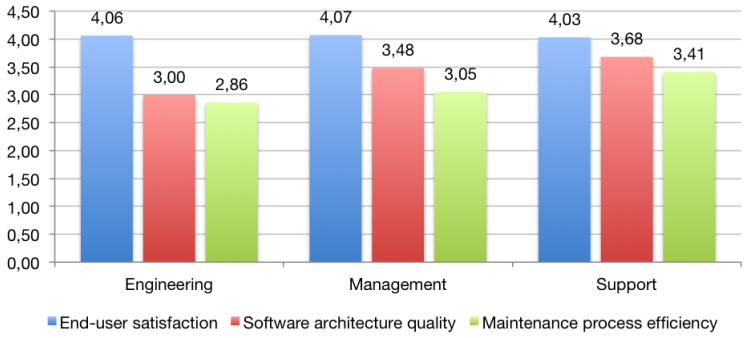


Figure 6.4 Importance of the maintenance foci for the different employee roles (questions 8–10)

with engineering (3.17 on average: more often than monthly), (2) engineering with management (3.00 on average: monthly) and (3) management with engineering (2.93 on average: less often than monthly). Also, engineering and management experience lack of consensus with respectively management and engineering most frequently, compared to other roles. We believe the cause of this outcome is related to the nature of the challenges that are faced by the two roles (i.e., technical versus managerial). Except for support, employees least frequently experience lack of consensus with employees having the same role.

Results of questions 8 to 10 (role of maintenance foci and corresponding improvement aspects) are displayed in figure 6.4. It is remarkable that for all employee roles, end-user satisfaction plays the largest role in prioritizing software maintenance tasks, software architecture quality the second-largest, and maintenance process efficiency relatively the smallest role. Comparing employee roles per maintenance focus, it should be noted that software architecture quality plays the lowest role for engineering. This may be caused by the confidence software engineers have in their work: of the three employee roles, engineers have the most knowledge of, and may be the most confident about the quality of their software architecture. Maintenance process efficiency plays the highest role for support (possibly because they wish as much in-the-field software failures reported by customers as possible to be fixed as soon as possible), and end-user satisfaction is approximately equally important for all employee roles.

Although the lack of consensus between the three employee roles can therefore not be justified by *absolute* differences in maintenance foci, it could be justified by *relative* differences. For example, the role software architecture qual-

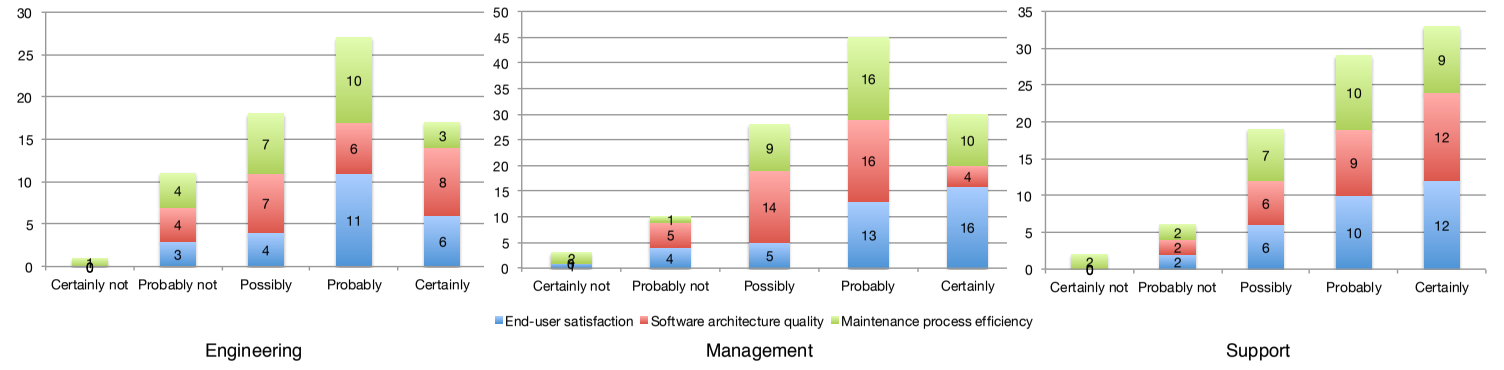


Figure 6.5 Expectations per employee role regarding the extent to which a software operation summary would increase their knowledge (question 13)

ity plays in prioritization of software maintenance tasks increases as employee roles have more direct communication with customer's end-users. In software-producing organizations, customer supporters are typically directly confronted with bugs and crashes experienced by end-users than management. Although engineering departments typically provide third line support, communication with end-users is less direct and frequent than between management and end-users.

6.5.3 Expectations

All employee roles would like to use a software operation summary about every three weeks (average usage frequencies are all more often than monthly: engineering 3.46 (σ = 0.76), management 3.59 (σ = 0.78), support 3.39 (σ = 0.90)). To illustrate the potential applications of such a summary, we created a frequency list of all responses to question 12. Based on analysis of this list, a software operation summary is expected to be most of use during activities related to sprint and release planning, bug fixing and software development, as well as during evaluations (of, for example, software operation at a particular customer, or the most recent sprint).

Expectations of engineering, management and support employees regarding the extent to which a software operation summary could increase their knowledge of software architecture quality, end-user satisfaction and maintenance process efficiency (question 13) is visualized by figure 6.5. With an SOS, all employee roles expect to save time on software maintenance task prioritization to reasonable extent: the modus of the answers to question 14 was 'Possibly' for engineering (average: 3.5, σ = 1.00), and 'Probably' for management (average: 3.38, σ = 0.92) and support (average: 3.24, σ = 0.97). Figure 6.3b visualizes the extent to which the different employee roles expect information in a software operation summary to foster the reach of consensus (question 15). First of all, the parties that experience a lack of consensus the most (support with engineering, engineering with management, management with engineering), expect an SOS to foster reach of consensus with the colleagues with which they experience this lack of consensus. As opposed to support, engineering and management both expect an SOS to foster reach of consensus the most with colleagues that have the same role. Of all employee roles, management has the highest expectations in reaching consensus with other employee roles. In absolute terms, highest expectations are expressed by engineers, expecting an SOS to foster reach of consensus with other engineers with an average of 3.96 (σ = 0.98, modus = 'Certainly'). Finally, most employees consider an SOS fostering

the reach consensus with colleagues on software maintenance task prioritization as probable (modus = 'Probably').

6.5.4 Crash Report Data

As stated in section 6.4, the last survey section of the survey is concerned with identification of which crash report data are considered relevant as a basis for an SOS that fosters reaching consensus on prioritization of software maintenance tasks. In view of this fact, we focus on the most-selected data types per maintenance focus *per employee role*. The more employees differ in data type selection, the more data types are involved in determining which bugs should be fixed to reach the largest increase of the three maintenance foci. See figure 6.6.

Regarding the software architecture quality focus, the most-selected data types are all related to software failure cause identification: data types 'Code file and line number causing the crash' and 'Module causing the crash' were most selected by engineering and management, while support selected data types 'Application name' and 'Thread name' the most. End-user satisfaction data types are more related to identification of the customer: 'Company name' and 'Application name' were data types most selected by engineering and management, while support selected data types 'Company name' and 'Comments accompanying report' the most. Finally, concerning maintenance process progress, some most-selected data types are related to software releases over time (i.e., 'Build version' and 'Version'). 'Build version' and 'Code file and line number presenting the crash' were data types most selected by management, 'Build version', 'Database' and 'Module cause crash' were most selected by engineering, and support selected 'Code file and line number presenting the crash', 'Code file and line number causing the crash' and 'Version' the most.

In total, four additional data types were suggested by respondents using the 'Other' field: (number of crash reports per) 'Performance peak, Slow request', 'Code file / class', 'GUI element' and 'Software configuration'. While the first suggestion is out of the scope of this paper (not being part of typical crash report data) and the second was already part of the presented data types (code file and line number, module), the latter two are taken into account for future research.

Focussing on the expectations for a software operation summary that is designed for fostering consensus on prioritization of software maintenance tasks, survey results can be summarized as follows:

- All employee roles would like to use such an SOS about every three weeks, particularly in preparation of (sprint) planning, (software) development and bug fixing

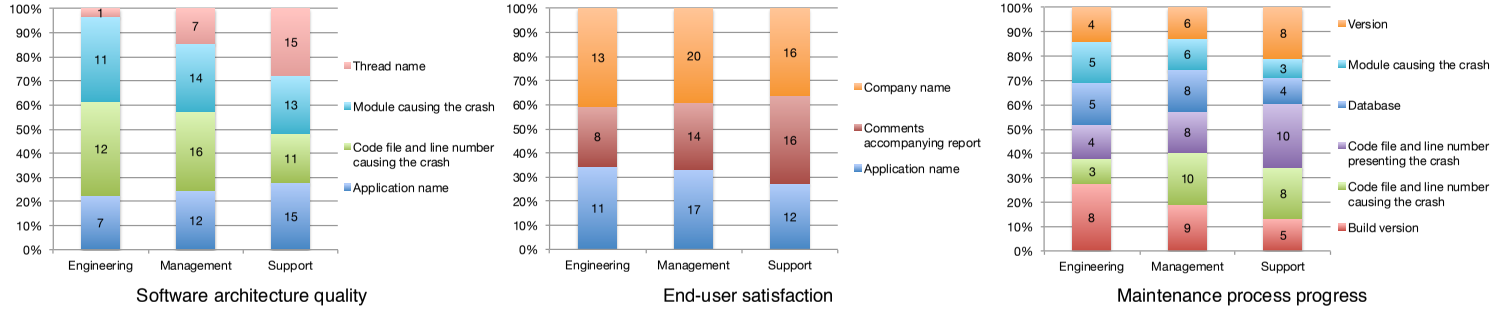


Figure 6.6 Most-selected crash report data types per maintenance focus per employee role. The data types are expected to contribute the most to successful determination of which bugs will result in the largest increase of software architecture quality, end-user satisfaction and maintenance process progress, once fixed (question 16). The data types form the basis of an SOS that fosters consensus on prioritization of software maintenance tasks

- Such an SOS is expected to particularly foster reach of consensus between managers and other employees, engineers themselves and between supporters and engineers

- Crash report data supporting identification of software failure causes, customers experiencing the failures, and software releases used by customers over time, form the basis of such an SOS.

6.6 SENDING OUT AN SOS: CASE STUDY RESULTS

To further establish the soundness and validity of the SOS concept, in addition to survey results we conducted a case study [Yin 2009] of one month at CADComp, during which we composed an SOS based on crash report data. Having attended development meetings and plenary company meetings, we observed lack of consensus between (product) management, and both development and support employees on prioritization of software engineering work items: based on received crash reports, the former employees believed that in-the-field product quality was increasing and implementation of new features should have first priority, while both latter employee groups contrarily believed that product quality was decreasing and bug fixing should have first priority.

Based on our observations at CADComp, we created an SOS in the form of an ASP.NET MVC web application that was deployed at the vendor's intranet. Derived from CADComp crash report data, the SOS presents software operation information in the form of graphs that visualize recent software operation history. Figure 6.7 shows one of the graphs that visualizes recent software quality: crash reports sent by various in-the-field releases of the CADComp software are distributed over the code file names (partly hidden for confidentiality reasons) and line numbers that caused the particular crash.

Apart from which code is causing in-the-field software failures in which version of the software, this SOS graph provides knowledge about crash report acquisition service outages (A), how in-the-field software quality improves over time (B), which code forms structural weaknesses in the software (C), when the software is barely used by end-users (D) and when new major bugs are introduced (E). Furthermore, the SOS provides operation meta data such as operation environment statistics and customer names corresponding to the crash.

The soundness and validity of the SOS in the context of CADComp's work item prioritization were evaluated through seven unstructured interview sessions [Yin 2009] with three product managers, two senior software engineers and two customer supporters. Although the product managers initially asked

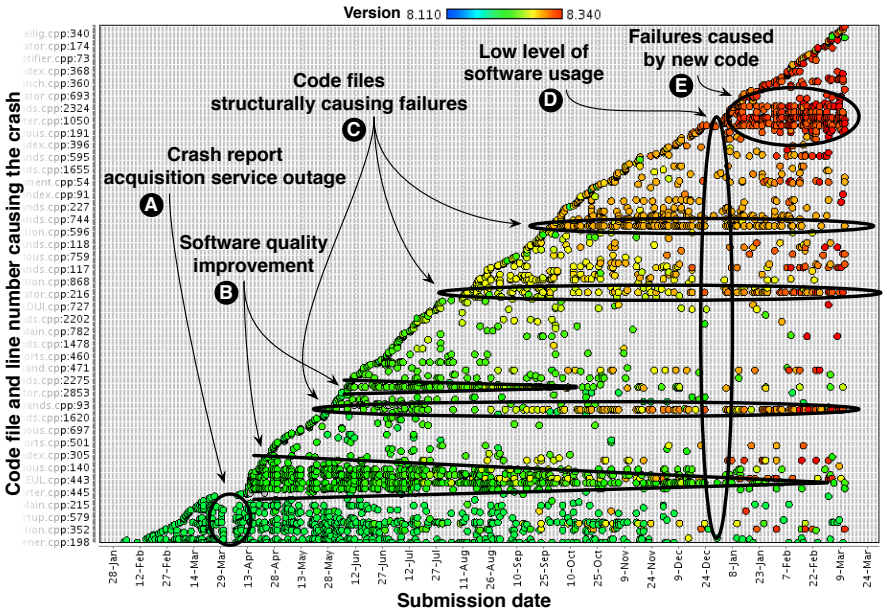


Figure 6.7 A software quality graph of the SOS implemented at CADComp, showing crashes per 'code file and line number causing the crash' over time

for help with interpreting the SOS graph depicted in figure 6.7 (they assumed the graph illustrated a linear increase of crash report submissions), all seven interviewees indicated that the graph increased awareness of in-the-field software operation throughout the organization, and expected the SOS application to contribute to the reach of consensus on work item prioritization (new features versus maintenance tasks). Also, software engineers suggested to only include crash reports in the SOS graph that form structural problems (since those problems cause the most crashes, and result in the highest increase in software architecture quality, once solved). Finally, customer supporters expected that on the long term, frequent usage of the software operation summary would result in an increase of customer satisfaction.

Based on results of both the CADComp case study as well as the maintenance task prioritization survey, the six propositions defined in section 6.3 are evaluated. Survey results show that most employees spend 1 to 5 hours weekly on software maintenance task prioritization, during which lack of consensus is frequently experienced, particularly with employees from engineering (monthly on average). All employee roles could well use a software operation summary

about every three weeks, particularly during in preparation of (sprint) planning, (software) development, and bug fixing activities. Therefore, we consider proposition **P1** (*A software operation summary is expected to integrate with current software maintenance practices*) to be correct. The next three propositions (*A software operation summary is expected to increase knowledge of software architecture quality* (**P2**), *end-user satisfaction* (**P3**), *maintenance process efficiency* (**P4**)) are also confirmed by survey results: employees from engineering, management and support mostly consider a software operation summary to increase knowledge of software architecture quality, end-user satisfaction and maintenance process efficiency as probable. Also, most respondents (28%) consider an SOS fostering the reach consensus with colleagues in terms of software maintenance task prioritization as probable. Proposition **P5** (*A software operation summary is expected to foster achieving consensus on software maintenance task prioritization*) is therefore considered to be correct. Proposition **P6** (*A software operation summary is expected to reduce the time needed for software maintenance task prioritization*) appears to be relatively difficult to evaluate: while on average, saving time on prioritization of software maintenance tasks through SOS usage is considered 'possible' by nearly all respondents, further research is needed to demonstrate reduction of the time needed for software maintenance task prioritization through SOS usage. We therefore consider the correctness of this proposition to be undetermined until further research on this subject has been performed. Finally, case study results support propositions **P1**, **P2**, **P3** and **P5**.

6.7 THREATS TO VALIDITY

The validity of the study results is threatened by several factors. First, survey-related aspects (as described by Kitchenham and Pfleeger [Kitchenham and Pfleeger 2002]) threaten construct validity of the research. For example, although 136 respondents initiated the survey, 79 (58.1%) completed it. This may be grounded in the type, order and number of survey questions. Also, as a consequence of the fact that the survey was anonymous, we can not exactly determine the number of distinct software-producing organizations that have responded to the survey, as well as the distribution of employee roles per organization. Furthermore, while concepts (such as the SOS) were introduced to respondents before questions related to these concepts were asked, it might be that common understanding among respondents was not fully reached. Although statistical relevance of (differences in) survey results may be limited by these factors, we believe the results are indicative and representative for the product software industry in the Netherlands.

Second, internal validity of the study may be threatened by the fact that in this study, we focus on corrective and adaptive software maintenance tasks, while preventive and perfective tasks are considered outside the scope of the study considering the types of data generally provided by crash reports. Conclusions based on the survey results might appear to be biased to certain extent. We acknowledge, however, that different types of operation information can be used to meet different requirements in optimizing maintenance activities and processes (as expressed in section 6.6).

Third, external validity of our research may be threatened by two factors. First, survey results could be dominated by respondents of a small number of software vendors. Second, in this study we focus on corrective and adaptive maintenance tasks. Both factors may influence the extent to which the results are applicable to other (types of) vendors: survey results might to certain extent be typical for software vendors in the Netherlands, and therefore are limitedly generalizable to vendors outside these categories. Although we believe that the results of this study are representative for many vendors, further research is needed to mitigate these threats.

6.8 RELATED WORK

Many research efforts cover the use of information fragments to improve people's practices, processes and workflows. However, to improve those through software operation knowledge is only considered by few. Kim *et al.* [Kim *et al.* 2011] propose a machine learning approach for predicting top software failures to prioritize maintenance efforts. Although their approach appears to be promising (75%–90% accuracy), it only considers the frequency of a crash as a prioritization factor: it does not consider software operation at particular customers, or involve the efficiency of the maintenance process, for example.

To determine the contents and investigate the quality of *bug* reports (as opposed to *crash* reports, which are researched in this paper), Zimmerman et al. conducted a survey among developers of Apache, Eclipse and Mozilla [Zimmermann *et al.* 2010]. Their conclusions (e.g. well-written, complete reports are likely to get more attention than poorly written or incomplete ones) may also apply to the SOS concept. However, their study only involves software engineers and does not consider management or customer support employees.

As an attempt to answer questions asked by software developers during their daily work, Fritz and Murphy [Fritz and Murphy 2010] have introduced an information fragment model that supports composition of information from multiple sources (among others, bug reports) and supports the presentation

of composed information in various ways. The authors show that the model can support 78 questions developers want to ask about a development project. While we recognize the value of combining information from multiple sources for answering diverse questions asked by one type of people, in our study we use one source of information (*crash reports*) to answer one question (*how should software maintenance tasks be prioritized?*) asked by multiple types of employees (*engineering*, *management* and *customer support*). Involving and combining multiple sources of information could lead to lengthy discussions regarding reason of information involvement as well as weight of the information in the resulting combination, which could delay the reach of consensus between different employee roles.

6.9 CONCLUSIONS AND FUTURE WORK

Software-producing organizations increasingly recognize the relevance and potential of software operation information visualizations on operation dashboards, reports and other media. However, all too often in industry, vendors experience difficulties in selecting and presenting operation information in such a way that besides providing software operation knowledge, visualized software operation information actually contributes to improvement of software process execution. Software maintenance task prioritization, for instance, is often time-consuming because reaching consensus between involved employees regarding this prioritization is tedious: involved employees have different roles, each with corresponding concerns, maintenance foci and improvement aspects.

In this paper, we propose the concept of software operation summary (SOS): an overview of recent in-the-field software operation based on acquired software operation information, which can be used to improve execution of software processes. For example, improving the process of software maintenance task prioritization by fostering the reach of consensus between employees on such prioritization.

We report on an extensive software maintenance task prioritization survey, held among 136 product / development managers, software developers and customer supporters. Survey results confirm the demand for an SOS that contributes to reach of consensus on software maintenance task prioritization, particularly in preparation of (sprint) planning, (software) development and bug fixing activities. As a basis for such an SOS, particularly crash report data supporting identification of (1) software failure causes, (2) customers experiencing the failures, and (3) software releases used by customers over time, are con-

sidered relevant. Furthermore, we report on a case study during which we compose an SOS based on crash report data, and empirically evaluate it at a European software vendor. Case study results show that an SOS increases awareness of in-the-field software operation throughout software-producing organizations, and indicate that it contributes to the reach of consensus on work item prioritization (e.g., new features versus maintenance tasks). Based on survey and case study results (through which the six propositions, defined to establish the soundness and validity of the SOS concept, are evaluated), we consider the main research question of this paper (*Can prioritization of software maintenance tasks be improved through the concept of a software operation summary?*) to be answered positively.

While this paper focuses on the use of a software operation summary in the context of software maintenance task prioritization, an SOS can be used to support numerous practices and processes. Using the SOK integration template method [Van der Schuur *et al.* 2011a], software vendors can tailor the form and contents of a software operation summary to their needs and requirements, and therewith optimize integration and presentation of software operation information. Vendors should ensure that recent operation information is actually available at the time and with the frequency an SOS is used. Also, selected underlying data types should correspond with the needs of the involved employees. Vendors should be aware, however, that too many data types underlying their SOS may limit the extent to which reach of consensus between their employees is fostered.

Future research activities include extension and concretization of the software operation summary concept: concrete SOSes that are tailored to processes other than software maintenance, will be developed and empirically validated, to further show its viability in terms of measurable process efficiency increases. Finally, it will be researched which visualization techniques are most appropriate for presenting (operation) data on software operation summaries and other media.

7

Becoming Responsive to Service Usage and Performance Changes

ABSTRACT

Software vendors are unaware of how their software performs in the field. They do not know what parts of their software are used and appreciated most and have little knowledge about the behavior of the software and its environment. In this paper we present a metrics-based approach that is used by software vendors to create real-time usage reports, based on data gathered by leveraging aspect-oriented programming techniques. This approach enables software vendors to respond quickly to performance and usage changes in their service software, both at specific customers and concerning the service software in general. We show that by using this approach, vendors can make informed decisions with respect to software requirements management and maintenance. The metrics and usage reports are validated by way of a case study at a Dutch software vendor. While validation shows high potential of the approach, a successful implementation will require change management at the software vendor.*

*This work has been published as *Becoming Responsive to Service Usage and Performance Changes by Applying Service Feedback Metrics to Software Maintenance* in the workshop proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008) [Van der Schuur *et al.* 2008]. It is co-authored by Slinger Jansen and Sjaak Brinkkemper.

7.1 INTRODUCTION

Nowadays, a lot is asked from software vendors. Their organizations have to be dynamic and agile, highly responsive to changes, while developing software at high speed, low cost and according to high quality standards. Two trends can be observed. First, a shift from Object-Oriented Development [Booch 2004] to Service-Oriented Development [Newcomer and Lomow 2004] can be perceived: software vendors focus on the development of (SOA) Web services and online, rich internet applications instead of offline, component-based desktop applications [Ibrahim *et al.* 2007]. In past years companies like Google, Yahoo and Microsoft have developed successful online applications like Gmail, Flickr, and Office Live. Second, while software vendors may have implemented a particular form of software usage data gathering mechanism analogue to Microsoft's error reporting functionality [Brelsford *et al.* 2002], few software vendors actually use the data that is collected. For example, still only very few software vendors make use of usage reports. The Dutch National Customer Configuration Benchmark Survey 2007 [Jansen *et al.* 2008] concludes that only 28% of the software vendors sell software that automatically composes usage reports and shows that only 30% of the software vendors automatically compose and send bug reports.

In this paper we introduce Service Knowledge Utilization (SKU) as an approach to increase a software vendor's flexibility, and responsiveness to changes in the performance and usage of its service-based, online software, at specific customers and concerning its software in general. Furthermore, we present the SKU report, a report that quantifies the usage and feedback of a software vendor's service-based software and contains three metrics-based indices that express software quality. In literature, metrics are used to measure the quality of a service (QoS) [Menascé *et al.* 2001, Yu and Lin 2005] and suggested as useful information that an infrastructure for managing and monitoring software can collect [Farrell and Kreger 2002].

In the SKU report presented in section 7.2, metrics are used to illustrate and summarize changes in software performance and usage, based on gathered software usage and feedback data. Section 7.5 details Nuntia, a software prototype based on aspect-oriented programming that provides a concrete SKU implementation, including data gathering and SKU report generation functionality. A case study at a Dutch software vendor was performed to validate both the report and the prototype. While software usage data gathering may easily result in significant performance loss of the software being traced, results show that the integration of our prototype with target software has a negli-

gible effect on the performance of the target software. Furthermore, research validation shows that the SKU report is considered valuable and supportive in management meetings on release management, requirements management and software maintenance. All case study results are presented in section 7.6. Finally, section 7.7 details conclusions and future research.

7.2 SERVICE KNOWLEDGE UTILIZATION

To enable a software vendor developing service-based software to become more flexible and responsive to changes in the performance and usage of its software, knowledge about the usage of the software by its end-users, as well as the utilization of this knowledge is of high importance. We define SKU as follows:

> ***Service Knowledge Utilization*** — Using service performance, usage and feedback knowledge to support the software development and maintenance processes and make software vendors more flexible and responsive to service performance and usage changes, both at specific customers and concerning their software in general.

As already argued by Mendelson and Ziegler [Mendelson and Ziegler 1999], *information awareness* (knowing what is going on in the surrounding world) enables enterprises to anticipate changes in the demands to their products and services. The authors define 'Listening to the Customer' as the first process that can be utilized to increase the information awareness of an enterprise [Mendelson and Ziegler 1999]. In order to improve their customer and information awareness, enterprises should consider their customers as partners and as the primary source of information, innovation and renewal.

In this paper, we present and validate the SKU report as an instrument to contribute to a software vendor's information awareness increase. The SKU report quantifies the service usage and feedback of all participants in a software vendor's software supply network that have licensed one or more of the vendor's services, and contains three indices that express the service quality in terms of performance, usability and customer usage.

7.2.1 SKU Report

As mentioned, the SKU report quantifies the service usage and feedback of all participants in a software vendor's software supply network that have licensed one or more of the vendor's services. All content of the report is generated from (service) software usage and feedback data gathered during a pre-determined

time period. The SKU report contains data and information about a number of concepts:

Customers
: The software vendor's customers of which the usage and feedback data are gathered, are represented by the set C, which is defined as $C = \{c_1, ..., c_n\}$.

Services
: The services developed and published by the vendor are represented by the set S, which is defined as $S = \{s_1, ..., s_p\}$. Each customer $c \in C$ has licenses for a collection of services. The services licensed by customer c_i are represented by the set S_i, which is defined as $S_i = \{s_{i1}, ..., s_{iv}\}$.

Methods
: Every service $s_j \in S$ has a set of published methods. The methods of a service s_j are represented by the set M_j, which is defined as $M_j = \{m_{j1}, ..., m_{jq}\}$.

Events
: The set of all events that took place during the execution of all services $s \in S$ is defined as $E = \{E_1, \cdots, E_p\}$. The set of events that took place during the execution of a service s_j is represented by the set E_j, which is defined as $E_j = \{e_{j1}, ..., e_{jr}\}$.

Event Types
: A number of event types exist, which are defined as $T = \{t_1, \cdots, t_w\}$.

The concepts above are subject to three constraints. First, the number of services a software vendor's customer c_i has licensed can not be larger than the number of services the software vendor has published. Second, the SKU report only contains data and information about licensed services: $S_i \subseteq S$. Third, every event $e \in E$ has an unique type $t \in T$: $\forall e \in E : t(e) \in T$.

For each method of each service, the report contains seven metrics, calculated in order to quantify and measure the status and quality of the services licensed by a vendor's customers.

7.2.2 SKU Metrics

The research community has developed a wide range of metrics to measure and quantify the quality of services [Kalepu *et al.* 2003]. The metrics listed below were selected by way of a literature study focussing on the extent the metrics

express service performance, usability and usage. This range of metrics (Reliability, Throughput, Latency, Accuracy, Availability, Usability, Reputation) is suggested by Farrell and Kreger [Farrell and Kreger 2002] as useful information that a software infrastructure for managing and monitoring Web services can collect for a Web service, and is also used in QoS-related research [Menascé *et al.* 2001, Yu and Lin 2005]. In the context of this paper, the metrics are used to construct three service indices that express the status of service-based software.

Reliability

The term 'Reliability' has a number of definitions. It is defined as '*the probability that a request is correctly responded within a maximum expected time frame*' [Zeng *et al.* 2003] or '*the quality aspect of a Web service that represents the degree of being capable of maintaining the service and service quality*' [Sahai *et al.* 2001]. Analogue to [Mani and Nagarajan 2002], we define reliability as '*The rate of successful responded requests '*.

Throughput

We define the throughput of a service as '*the number of successfully completed service requests over a time period*' [Ran 2003]. Often, throughput is measured in requests per time unit [Von Bochmann *et al.* 2001].

Latency

Latency is defined as the '*time taken between the moment the service request arrives and the moment the request is being serviced*' [Ran 2003]. We define the latency of a service as '*the round-trip time between sending a request and receiving the response*' [Sahai *et al.* 2001]. The throughput of a system (service) is related to its latency.

Accuracy

In a generic way, 'accuracy' is defined as '*percentage of objects without data errors such as misspellings, out-of-range values, etc.*' [Naumann *et al.* 1999]. More specifically, we define accuracy as '*the error rate produced by a service*' [Ran 2003]. In practice, this metric indicates the amount of exceptions (both handled and unhandled) a service or service method generates.

Availability

The term 'Availability' has various definitions: '*The quality aspect of whether the Web service is present or ready for immediate use, and represents the probability that a service is available.*' [Sahai *et al.* 2001], '*The percentage of time a source is accessible based on technical equipment and statistics*' [Naumann *et al.* 1999] or '*The probability a system is up*' [Ran 2003, Von Bochmann *et al.* 2001]. In this paper, we use the latter definition.

Usability

Seffah *et al.* [Seffah *et al.* 2006] developed a consolidated usability measurement model based on the research of Shackel [Shackel 1991] and Nielsen [Nielsen 1993]. Among other criteria, the 'minimal action' criterion is identified: '*The capability of the software product to help users achieve their tasks in a minimum number of steps*'. Given the nature and role of Web services in the context of this research — users actively 'interact' with a (process-centric or enterprise level) Web service via its graphical user interface — the usability of Web services is defined as the 'minimal action' criterion: the number of steps a user has to perform to achieve a particular task.

Reputation

'*The reputation of a service is a measure of its trustworthiness. It mainly depends on end user's experiences of using the service. Different end users may have different opinions on the same service*' [Zeng *et al.* 2003]. In the context of this research, reputation of a service is expressed in a '*user grade from 1 to 10 based on personal preferences and professional experience*' [Naumann *et al.* 1999].

The notation of the metrics defined above is depicted in table 7.1. m_{jk} represents a method of a service s_j published by a software vendor.

Notation	Description
$q_{Rel}(m_{jk})$	The reliability of a method m_{jk}
$q_T(m_{jk})$	The throughput of a method m_{jk}
$q_L(m_{jk})$	The latency of a method m_{jk}
$q_{Acc}(m_{jk})$	The accuracy of a method m_{jk}
$q_{Av}(m_{jk})$	The availability of a method m_{jk}
$q_U(m_{jk})$	The usability of the graphical interface of a method m_{jk}
$q_{Rep}(m_{jk})$	The reputation of a method m_{jk}
$q_{\{Rel,\cdots,Rep\}SLA}(s_j)$	The minimal (or maximal) value of a metric (specified as a part of an SLA or service contract)

Table 7.1 Metrics notation

Before further detailing how the service indices are constructed, it is important to observe that services may be bound to service contracts and corresponding service levels, specifying the required performance of a service. Specifically, such a contract is a Service Level Agreement (SLA) between the service provider and the service requester, inter alia describing (un)acceptable service

levels [Verma 1999, Lewis 2001]. A service level is defined as some mark by which to qualify acceptability of a service parameter [Lewis 2001], for example a guaranteed minimal availability of a service.

Concerning the construction of the service indices, for all metrics, service levels are used to calculate the metric *scores*. Depending on the type of service level (minimal or maximal), two types of metric scores can be identified. The first type applies to the metrics Reliability, Throughput, Availability and Reputation. If the Reliability, Throughput, Availability or Reputation of a service method is equal to or greater than the corresponding minimal value of that metric prescribed in the SLA, $q_{\{Rel,T,Av,Rep\}_{SLA}}(s_j)$, the metric score of the service method on that metric $r_{\{Rel,T,Av,Rep\}}(m_{jk})$ is equal to 1.0 — the method is operating according to its service level. Otherwise, the metric score is equal to the Reliability, Throughput, Availability or Reputation metric value, divided by the corresponding SLA value for that metric. If the metric score of a method is equal to 0.0, the method is failing maximally (because its service is offline, or always throwing an unhandled exception, for example) with respect to its SLA.

The second metric score type applies to the metrics Latency, Accuracy and Usability. If the Latency, Accuracy or Usability of a service method is equal to or less than the corresponding maximal value of that metric prescribed in the SLA, $q_{\{L,Acc,U\}_{SLA}}(s_j)$, the metric score of the service method on that metric $r_{\{L,Acc,U\}}(m_{jk})$ is equal to 1.0 — the method is operating according to its service level. Otherwise, the metric score is equal to the Latency, Accuracy or Usability metric SLA value, divided by the value of that metric. Again, if the metric score of a method is equal to 0.0, the method is failing.

7.3 SKU REPORT INDICES

The metric scores $r_{\{Rel,T,L,Acc,Av,U,Rep\}}(m_{jk})$ are calculated to eliminate the differences between the domains of the metrics listed in this section, and to enable easy service status and quality comparison. Based on the resulting metric score values, three service indices are constructed.

Service Performance Index

Represents the average performance of the services the software vendor has published.

Service Usability Index

Represents the average usability of the services the software vendor has published. Gives an indication of to which extent users value and rate the

services of a software vendor, and to which extent they are satisfied with using those services.

Service Client Utilization Index
: Represents to which extent the services the software vendor has published, are utilized by the clients that have licensed the particular services.

All indices are calculated for each service s_j that is contained in the SKU report, as well as for each method $m_{jk} \in M_j$ of each service s_j, as well as for all services s_{ix} of a customer c_i. Furthermore, three 'aggregate' service index values are calculated, representing the status of all services that are licensed by a customer. The construction of the service indices is described below.

7.3.1 Service Performance Index

The Service Performance Index expresses the quality of a method (or service, or collection of services) in terms of its reliability, capacity, response time, error rate and on-line time. $\pi(m_{jk})$ represents the value of the Service Performance Index for a service method m_{jk}. The Service Performance Index of a method m_{jk} is calculated as follows:

$$\pi(m_{jk})) = r_{Rel}(m_{jk}) \cdot r_T(m_{jk}) \cdot r_L(m_{jk}) \cdot r_{Acc}(m_{jk}) \cdot r_{Av}(m_{jk}) \tag{7.1}$$

Equation 7.2 shows how the Service Performance Index is calculated for a service $s_{ix} \in S_i$ — where S_i is the set of services licensed by customer c_i — as the average performance of its methods.

$$\pi(s_{ix}) = \frac{\sum_{k=1}^{q} \pi(m_{ixk})}{q} \tag{7.2}$$

The equation below illustrates how to calculate the Service Performance Index for a customer c_i. This index expresses the average performance of the set of services that this customer has licensed. The constant factor M is added to stress the aggregate character of the customer-level service indices, and improve the comparability of customers based on those indices. With this constant factor included, customer-level service indices 'rate' the software vendor's customers based on the services they have licensed, on a scale of 0 to M. $M = 5$, $M = 7$ and $M = 10$ are proven intuitive by research [Dawes].

$$\pi(c_i) = M \cdot \frac{\sum_{x=1}^{v} \pi(s_{ix})}{v} \tag{7.3}$$

7.3.2 Service Usability Index

The Service Usability Index expresses the quality of a method (or service, or collection of services) in terms of the number of actions a user has to perform to activate the method, error rate, response time, and user feedback. $\upsilon(m_{jk})$ represents the value of the Service Usability Index for a service method m_{jk}. Given a service method m_{jk}, its Service Usability Index is constructed as follows:

$$\upsilon(m_{jk}) = r_U(m_{jk}) \cdot r_{Acc}(m_{jk}) \cdot r_L(m_{jk}) \cdot r_{Rep}(m_{jk}) \tag{7.4}$$

Equation 7.5 shows how the Service Usability Index is calculated for a service $s_{ix} \in S_i$ — where again, S_i is the set of services licensed by customer c_i — as the average usability of its methods.

$$\upsilon(s_{ix}) = \frac{\sum_{k=1}^{q} \upsilon(m_{ixk})}{q} \tag{7.5}$$

Equation 7.6 illustrates how to calculate the Service Usability Index for a customer c_i. This index expresses the average usability of the set of services that this customer has licensed.

$$\upsilon(c_i) = M \cdot \frac{\sum_{x=1}^{v} \upsilon(s_{ix})}{v} \tag{7.6}$$

7.3.3 Service Client Utilization Index

The Service Client Utilization Index expresses the quality of a method (or service, or collection of services) in terms of the number of method requests over a predefined period of time, response time, and user feedback. $\kappa(m_{jk})$ represents the value of the Service Client Utilization Index for a service method m_{jk}. Given a service method m_{jk}, its Service Client Utilization Index is constructed as follows:

$$\kappa(m_{jk}) = r_T(m_{jk}) \cdot r_L(m_{jk}) \cdot r_{Rep}(m_{jk}) \tag{7.7}$$

Equation 7.8 shows how the Service Client Utilization Index is calculated for a service $s_{ix} \in S_i$ — where again, S_i is the set of services licensed by customer c_i — as the average client utilization of its methods.

$$\kappa(s_{ix}) = \frac{\sum_{k=1}^{q} \kappa(m_{ixk})}{q} \tag{7.8}$$

Equation 7.9 illustrates how to calculate the Service Client Utilization Index for a customer c_i. This index expresses the average utilization of the set of services that this customer has licensed.

$$\kappa(c_i) = M \cdot \frac{\sum_{x=1}^{v} \kappa(s_{ix})}{v} \tag{7.9}$$

Having presented all service index constructions on all levels (methods, services and customers), note that all metric scores $r_{\{Rel,T,L,Acc,Av,U,Rep\}}(m_{jk})$ have equal weight in the construction of each of the service indices. As detailed in section 7.6, a software vendor may assign different weights to each of the metric scores, in line with its strategy focus. A number of observations can be made. First, the domains of the index values $\pi(s_{ix}), \upsilon(s_{ix}), \kappa(s_{ix}), \pi(m_{jk}), \upsilon(m_{jk})$ and $\kappa(m_{jk})$ are equal to the domains of the metric scores.

In other words, when the value of, for example, $\pi(s_{ix})$ for a service s is equal to 1.0, the service is performing according to the metric values in its SLA. A value of 0.0 indicates that a service is maximally failing in meeting its SLA. Second, the domains of the service indices on customer level are as follows:

$$\{\pi(c_i), \upsilon(c_i), \kappa(c_i)\} \in [0, M]$$

Obviously, the change in domains is caused by the constant factor M included in the formulas of all customer-level service indices. Third, note that of all concepts, the 'event' concept is not mentioned within the construction of the indices. This is because events are used to calculate the numerous metric values. Similarly, the concept of an event type is introduced to enable event categorization. In section 7.5, an example SKU report implementation is presented, as part of the description of the software prototype Nuntia that provides SKU report generation functionality.

7.3.4 Related Work

In research, QoS metrics are often used in the process of web service discovery, selection or composition. Yu and Lin [Yu and Lin 2005] propose a set of service selection algorithms for web services with QoS constraints. Compared to the work of Yu and Lin and others, our research focuses on the application of metrics as a means to achieve software quality goals and increase the responsiveness of software vendors to performance and usage changes in their software. Farrell and Kreger [Farrell and Kreger 2002] propose types of information that can be collected with respect to web service management, and indicate that

this information could serve as a basis for service usage monitoring. However, the solutions the authors provide do not feature any form of (summarizing) report functionality similar to the SKU report presented in this paper. Finally, Papapetrou and Papadopoulos [Papapetrou and Papadopoulos 2004] present an AOP-based logging tool for a component-based software. While their application of aspect-oriented programming is similar, our solution is also utilized with service-based software and environments.

7.4 RESEARCH APPROACH

Part of the research was the development of the SKU approach and the SKU report, including its service metrics and indices. Furthermore, a software prototype (section 7.5) was developed to provide an service knowledge utilization implementation for software vendors developing service-based software. A case study at a Dutch software vendor was conducted to validate the service metrics and indices, as well as the SKU reports generated by the prototype.

7.4.1 Company Identification

Validation of the research outputs mentioned above was done by means of a case study at a Dutch software vendor. The vendor develops and distributes CAD software products for the building services industry, creating designs in the '.dwg' file format. The software vendor's market leading product is now used by more than 7000 draftsmen every day in the Netherlands and Belgium. Founded in 1990, the vendor has set the basis for CAD software for contractors in the Netherlands and Belgium. With over 3800 customers and more than 8000 licenses sold, the vendor is market leader in its segment. The software vendor approximately employs 100 employees. Recently, the vendor made its product line available abroad as well. Since August, 2008, the first localized editions were released for Belgium, France and the United Kingdom. In the future, other localizations may be developed and released. The vendor's software development activities are mainly performed in the Netherlands.

7.4.2 Case Study Approach

In order to identify the case study company and gather facts on which the results of the case study are based, four case study techniques have been used. Yin [Yin 2009] has defined six sources of case study evidence. Numerous sources are utilized as part of this research.

First, 'direct observations' were made during the presence at the software vendor. Development meetings where software developers and project leaders presented details and progress concerning the development of the vendor's software products and services, where attended, for example. Second, a document study was done. The vendor provided software architecture specifications, software manuals, process descriptions and numerous memos. Third, the vendor's software was studied. A training session was attended in order to get familiar with the software, end-users of the software and the vendor's training sessions. Furthermore, the source code of the software was examined. Finally, twenty semi-structured interviews have been conducted. An SKU survey was designed specifically for each of the vendor's departments that employed people invited to the case study interview. Interviewees were asked to elaborate on their survey answers and describe the 'current' situation with respect to SKU at their department.

7.4.3 Hypotheses

Having implemented the software prototype successfully, it is expected that the effects of the utilization of the Nuntia-generated SKU report by the software vendor are twofold.

Decrease of Time to Market

It is expected that the software vendor's time to market will be shortened. Three causes can be identified. First, the internal communication of the software vendor will be improved because the SKU report provides an informed view on the performance, usability and usage of the software, easing feature-related decision making and thus shortening the related management meetings. Second, frequent analysis of the the SKU reports potentially discloses end-user's configuration and severe bugs earlier and may help to reproduce bugs, possibly resulting in shorter overall development cycles. Third, by effectively utilizing the software (usage) knowledge gathered, performance and usage changes in the vendor's software potentially are expected and incorporated in the vendor's projects at an earlier stage, in a shorter amount of time.

Responsiveness Increase

The software vendor's responsiveness (the speed at which changes are implemented based on performance and feedback data), for example to issues reported to its help desk, may increase when the vendor has implemented the SKU prototype. With the SKU presented, the usage of soft-

ware is logged continuously. Furthermore, actions can be defined that have to be executed when particular events occur and are logged. For example, an action could describe that as soon as a severe exception occurs, the help desk of the software vendor has to be informed.

7.4.4 Validation Methods

The research outputs (hypotheses, the software prototype and the SKU report) were validated in three ways.

Interviews

To determine the correctness of the information interviewees provided and the information gathered during direct observations, reflective questions were asked during the interviews, especially during the interviews with key people of a department or the total organization. Furthermore, project leaders, developers and help desk employees were asked to give their opinion about the final prototype and its fit within the vendor's organization.

Expert Validation

Before implementing the prototype, both the metrics and formulas used to calculate the service indices of the SKU report generated by the software prototype were reviewed and validated by the product managers of the software vendor. Furthermore, the vendor's product managers and senior developers were asked to review the Nuntia-generated SKU report. Specifically, they were asked to validate the value, relevance and usefulness of the information contained in the report, as well as the hypotheses listed before). All experts received a set of statements related to the SKU report and the hypotheses, which they were asked to confirm or reject using a Likert scale (1: strongly disagree, 5: strongly agree).

Testing

Lead developers and testers of the software vendor, as well as the vendor's CEO were asked to test the software prototype in practice, inter alia in terms of stability and performance.

7.5 SKU SOFTWARE PROTOTYPE

The goal of the software prototype, Nuntia, is to gather software usage and feedback data from software products and services that are developed by software vendors and are licensed by one or more participants in the vendor's soft-

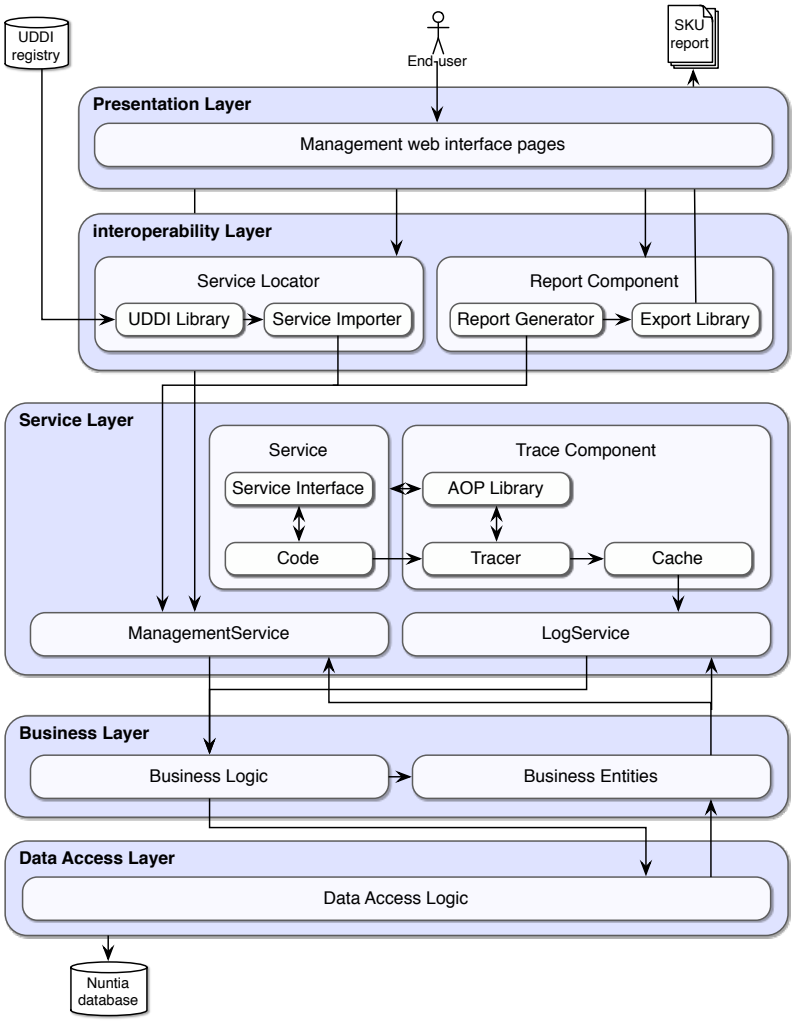


Figure 7.1 Architecture of Nuntia

ware supply network, and to provide means to extract valuable real-time software (usage) knowledge from the data gathered, in order to increase a software vendor's flexibility and responsiveness to changes in software usage and performance.

In order to gather service usage data of a service oriented architecture's (SOA) services, Nuntia can be easily integrated in a SOA by using the UDDI-based Service Locator. To actually gather service usage and feedback data,

Nuntia's trace component is integrated in the target service. In addition, Nuntia provides an SKU report generation implementation: when service usage and feedback data is gathered, SKU reports based on this data are generated with the Report Generator. Finally, two web services are part of Nuntia. First, the ManagementService forms the interface to the Nuntia database and is used by Nuntia's web interface to enable data manipulation. Second, the LogService is used to gather, cache and store service usage and feedback data. While the ManagementService is used by the software vendor itself to register its customers and services, the LogService is also accessible for (pilot) customers and other partners in the vendor's software supply network, in order to register their service usage and feedback. See figure 7.1.

Nuntia was developed using Visual Studio 2008 and the .NET framework 3.5, both from Microsoft [VS2008]. All code was written in C# 3.0. Concerning Nuntia's trace component, one technique is utilized extensively: Aspect-Oriented Programming (AOP), created by Kiczales *et al.* [Papapetrou and Papadopoulos 2004]. AOP is a programming paradigm that contributes to the area of *separation of concerns*. Some concerns, *cross-cutting* concerns, cannot be separated cleanly, because they are related to numerous, distinct modules of a program (the concerns 'cut' across different program modules). Examples of cross-cutting concerns are security and logging. Both concepts are often needed throughout all modules a program consists of, because the need to log exceptions exists program-wide, for example. In AOP, code that is needed program-wide (a so-called *advice*), is injected into the existing code of the program, at locations (*point-cuts*) determined by the programmer. The combination of an advice and a point-cut is called an *aspect* [Filman *et al.* 2005]. For the process of advice injection (also referred to as *weaving*) to take place, the programmer does not have to adjust the program code structurally. Next, the trace component is described in more detail.

7.5.1 Tracing

Nuntia's trace component consists of three elements: an aspect-oriented programming (AOP) library, a tracer and a cache component.

AOP Library

Nuntia utilizes the (compile-time) weaving functionality of the PostsharpAspNet to trace the usage and performance of services. PostsharpAspNet was released in February, 2008 and is an experimental library based on the PostSharp Laos aspect weaver [Fraiteur 2008].

Tracer

The tracer contains advices that are inserted at the point-cuts in the 'target' service code. Specifically, the tracer contains four aspects that cover generic events related to the web service's methods: OnEntry, OnSucces, OnException and OnExit. In those aspects, relevant service environment and status information is saved in an Event object (in the case of an exception, for example, the most recent stack trace and the exception message are stored in this object). Next, the Event object is sent to the cache.

Cache

When a service has to process a lot of requests from a great number of end-users or external services, both database load and network traffic may raise to high levels. In order to prevent network congestion, a cache is part of the trace component. When the cache capacity is reached, the cached is flushed: using Nuntia's LogService, all Event objects are inserted in the Nuntia database and the cache is cleared.

7.5.2 Report Generation

When receiving a report generation request, the report generator requests all relevant data related to the time frame, customers and services from the Nuntia database. Next, the report generator calculates metric scores and service indices for the selected services (including the services' web methods) of a selected customer in the selected time period. When no data is available for a particular metric, the metric score is set to 1.0 (the SLA requirement is met). This is done to prevent negative influences on the metric scores and service index values by methods that are not called within the selected time frame (and for which no data is stored)[1]. When this is the case, a 'not representative' annotation is added to the data. All metric score values have a color that indicates their status. Green indicates that the SLA value is met, orange indicates that the SLA value is not met and red indicates that the metric score is lower than half the SLA value.

Concerning the customer-level service index calculation: the constant M is equal for all reports can be set via Nuntia's web interface. Apart from the metric score and service indices calculation, a list of all events occurred within the selected time frame is included in the SKU report. For example, for each exception event, the list contains exception details and stack trace and detailed environment information. Furthermore, the list contains data of all request and

[1] In other words: no conclusions concerning performance, usability, availability, etc. can be drawn.

SKU Indices Prototype Calculation Sheet of 5/14/2008 - 5/20/2008

Customer 'Vendor B.V.'

		Rel		T		L		Acc		Av		U		Rep		Index factors			SLA	
Service	**Method**	abs	score	abs	score	abs	score	abs	score	abs	score	abs	score	abs	score	P	U	C	metric	value
StabiBASE Web																**0.82**	**0.84**	**0.85**	**Rel**	0.90
	GetChildrenFolders	1.00	1.00	72.00	1.00	0.38	1.00	0.00	1.00	0.93	1.00	3.00	**1.00**	3.50	1.00	1.00	1.00	1.00	**T**	2.00
	GetLogo	0.87	0.97	80.00	1.00	2.09	0.24	0.07	0.15	0.93	1.00	2.00	1.00	3.50	1.00	0.04	0.04	0.24	**L**	0.50
	GetFreeFieldsById	0.90	**1.00**	2.00	**1.00**	0.50	**1.00**	0.01	**1.00**	0.93	1.00	3.00	**1.00**	3.50	1.00	1.00	1.00	1.00	**Av**	0.90
	GetReservationsByFileId	0.99	1.00	277.00	1.00	0.06	1.00	0.00	1.00	0.93	1.00	3.00	**1.00**	3.50	1.00	1.00	1.00	1.00		
	GetProfileByFileId	0.90	**1.00**	2.00	**1.00**	0.50	**1.00**	0.01	**1.00**	0.93	1.00	3.00	**1.00**	3.50	1.00	1.00	1.00	1.00		
	GetPhasesByFileId	1.00	1.00	1.00	0.50	0.03	1.00	0.00	1.00	0.93	1.00	3.00	**1.00**	3.50	1.00	0.50	1.00	0.50		
	SetFileFormat	1.00	1.00	11.00	1.00	0.58	0.86	0.00	1.00	0.93	1.00	3.00	**1.00**	3.50	1.00	0.86	0.86	0.86		
	SetSystemLanguage	1.00	1.00	50.00	1.00	0.31	1.00	0.00	1.00	0.93	1.00	3.00	**1.00**	3.50	1.00	1.00	1.00	1.00		
	GetFileFormat	1.00	1.00	32.00	1.00	0.42	1.00	0.00	1.00	0.93	1.00	3.00	**1.00**	3.50	1.00	1.00	1.00	1.00		
	HasPreview	0.90	**1.00**	2.00	**1.00**	0.50	**1.00**	0.01	**1.00**	0.93	1.00	3.00	**1.00**	3.50	1.00	1.00	1.00	1.00		
	GetLanguageIndex	0.90	**1.00**	2.00	**1.00**	0.50	**1.00**	0.01	**1.00**	0.93	1.00	3.00	**1.00**	3.50	1.00	1.00	1.00	1.00		
	IsValidLogin	0.96	1.00	50.00	1.00	0.50	1.00	0.04	0.26	0.93	1.00	3.00	**1.00**	3.50	1.00	0.26	0.26	1.00		
	GetPhase	1.00	1.00	2.00	1.00	0.02	1.00	0.00	1.00	0.93	1.00	3.00	**1.00**	3.50	1.00	1.00	1.00	1.00		
	ShowDocuments	0.90	**1.00**	2.00	**1.00**	0.50	**1.00**	0.01	**1.00**	0.93	1.00	3.00	**1.00**	3.50	1.00	1.00	1.00	1.00		
	GetSelectedLanguage	0.46	0.51	34.00	1.00	0.99	0.51	0.00	1.00	0.93	1.00	3.00	**1.00**	3.50	1.00	0.26	0.51	0.51		
													SKU Index			**4.50**	**4.50**	**4.60**		

Figure 7.2 Screenshot of a SKU report generated by Nuntia

response events, as well as user interface paths for all user interface events. When all required data is gathered, the data is sent to an export library which compiles all data into a report that is presentable to the end-user. Figure 7.2 depicts a screenshot of a Nuntia-generated SKU report ($M = 5$).

7.6 RESULTS

To gather the results presented in this section, a number of tests were conducted with a customized version of the web edition of the vendor's CAD software. Performance measurement and exception gathering code, feedback gathering functionality (to calculate the Reputation metric) and a user interface step count algorithm (to calculate the Usability metric) were implemented in this version. As mentioned, experts were interviewed to validate the results. Next, validation results of two main research outputs are presented.

7.6.1 Prototype Performance

The performance of a software usage and feedback data gathering solution is key to the value and success of its integration with live software. A number of performance tests were conducted to measure the influence of Nuntia on the performance of the target software. The total delay Δ caused by the tracing process during the execution of a service s_j can be expressed as follows:

$$\Delta = \sum_{z=1}^{r} \delta_1(e_{jz}) + \delta_2(e_{jz}) = \sum_{z=1}^{r} \rho_T(e_{jz}) - \rho_O(e_{jz}) \tag{7.10}$$

... where e_{jz} represents an event that occurred during the execution of service s_j (e_{jz} is an element of the set that represents all events that occurred during the execution of s_j: $e_{jz} \in \{e_{j1}, \cdots, e_{jr}\}$, or $e_{jz} \in E_j$), δ_1 represents the time needed to process the advice code associated with the entry of a web method (δ_1) and δ_2 is the time needed to process the advice code associated with the exiting of the web method. Furthermore, ρ_T and ρ_O represent the total web method execution time with and without Nuntia integrated, respectively. Δ then is equal to the sum of both part delays caused by all events $e_{jz} \in E_j$, or $\Delta = \sum_{z=1}^{r} \delta_1(e_{jz}) + \delta_2(e_{jz})$. Moreover, Δ can be expressed in terms of total web method execution times. Defining ρ_T and ρ_O as the total web method execution time with and without the prototype integrated, respectively, Δ is defined as the sum of the differences between those times for all events that occurred during the service execution, or $\Delta = \sum_{z=1}^{r} \rho_T(e_{jz}) - \rho_O(e_{jz})$. Numerous test sessions have been conducted. The results of four recent sessions are presented[2] in table 7.2. T represents the total test session duration in minutes. To effectively measure performance differences, all tests contained the same steps and were conducted in the same order. All tests conducted within one session (A, B, C, D) were based on identical test plans.

	$\sum_{z=1}^{r} \rho_O(e_{jz})$	$\sum_{z=1}^{r} \rho_T(e_{jz})$	Δ	T
A	36.15	47.44	11.29	10 min.
B	23.56	40.68	17.13	5 min.
C	30.94	36.91	5.98	5 min.
D	15.87	23.06	7.19	5 min.

Table 7.2 Performance test results

Tests were executed by the authors, as well as testers and product managers of the CAD vendor. When measuring $\rho_T(e_{jz})$, both the Nuntia database and the LogService were installed on an external machine (when only one of those components is located externally, Δ has a lower value). No performance optimizations were done: no indexes were added to tables in the Nuntia database and both the target software and Nuntia were compiled and running in debug mode. Cache capacity was set to 30 events.

All test results show a low Δ/T rate: the delay caused by the tracing process is spread over a relatively long time. Furthermore, in all test sessions except B, $5 < \Delta < 15$. In test B, the test machine was completely restarted after each test. As detailed in table 7.2, this increases Δ significantly, due to required connection

[2] All values represent seconds, except when indicated differently.

and plugin initialization. Test results also show a low average delay per event (in seconds, not listed in table 7.2):

$$\frac{\Delta_A + \Delta_B + \Delta_C + \Delta_D}{r_A + r_B + r_C + r_D} = 0.008 \tag{7.11}$$

This is in line with the opinion of the experts that were asked to validate the prototype: while all experts recognized that the integration of the prototype with the target software entails some performance loss and that the tracing effects during heavy usage are unknown, they consider the performance effects as negligible. Furthermore, both product and project managers expect Nuntia to be integrated with their software. However, two project managers indicated that Nuntia will not improve the vendor's responsiveness *on its own*: the vendor will have to implement a process to take action based on the gathered knowledge (fix bugs revealed by exception data in the SKU report, for example) and assign employees to this process, also to prevent a 'software usage data flood' towards the organization.

7.6.2 SKU Report

Overall, the experts indicated that the report contains data that is valuable to software vendors and that the report supports informed decision making in the areas of software development and software maintenance. Concerning the report's service indices, experts indicated all service indices should be included in the SKU report. In their opinion, especially the service client utilization index was considered a valuable indication. This is in line with earlier interviews and observations: both developers and project managers indicated that the vendor has difficulties in determining whether or not an existing feature is still used (and should not be removed yet) or whether a new feature will be used by their end-users.

Except for the throughput, availability and reputation metrics, the experts agreed that all metrics proposed should be included in the report. They denoted that the throughput and availability metrics especially are of importance with respect to process-centric services. Unlike the vendor's service software (which can be characterized as a public enterprise service), services of this type have to meet a specific demand, and thus guarantee a particular level of throughput or availability. Project and product managers also indicated that only proper and serious feedback will be valuable. Therefore, they proposed to include the reputation metric primarily with pilot customer tests. Furthermore, the availability metric was considered as less valuable because the soft-

ware may not be the cause of a low availability score, the experts reasoned. The events overview in the report was considered optional and only valuable for debugging purposes. It was suggested to calculate the metrics on more levels of detail and to base the report structure on the structure of the target software.

While the data contained in the report was expected to be taken into account when taking decisions related to operational, tactical and strategical management, product managers indicated that the SKU report will be primarily used in tactical management meetings (release management meetings, for example) and added that the report will have to be summarized in order to be of use in strategical meetings. It was also denoted that a successful SKU implementation would require some form of change management: people will have to get used to involve the knowledge contained in the report in their daily work, for example. The report will be generated once a month.

7.7 CONCLUSIONS AND FUTURE RESEARCH

Case study results show that Nuntia is expected to contribute to a software vendor's responsiveness to software performance and usage changes, and thus is an effective SKU implementation. The vendor has decided to integrate Nuntia with its live software in the near future because of the negligible performance loss this integration involves and the valuable information the SKU report provides.

The SKU implementation is expected to contribute to a software quality increase on the short term and a time-to-market decrease on the long term: the vendor noted that it will first have to invest into the process of acting upon the knowledge, before it is able to structurally improve its responsiveness based on the gathered knowledge. The vendor expects to base software maintenance processes (requirements elicitation and bug fix priority assignment) on the SKU reports generated by Nuntia, and be able to react before an end-user calls for support. Finally, Nuntia's architecture (loosely-coupled services, use of AOP) enables easy integration with other software. We consider both hypotheses (decrease of software vendor's time to market and increase of vendor's responsiveness) supported by the research validation.

While the results of this research are promising, further research is needed to improve the applicability of the SKU report and the performance of the prototype. We will research data mining techniques and software tomography [Bowring *et al.* 2003] to extract sophisticated statistics from large software knowledge repositories and integrate these in the report, and keep performance loss of the target software negligible under heavy use circumstances. Future research will

also focus on integration of service knowledge in software development environments, SKU implementation change management requirements, and feedback visualization techniques.

Part IV

Conclusion

8

Conclusion

In the introduction of this dissertation, the main research question is formulated as follows:

> ***MRQ*** *— How can software processes of product software vendors be improved through software operation knowledge?*

This dissertation addresses this question through construction of the software operation knowledge reference framework, which defines the parties, perspectives and processes that constitute the software operation knowledge life cycle. In the chapters of this work, the software operation knowledge concept and framework are defined and empirically evaluated. Each of the five software operation knowledge life cycle processes is addressed by one or more chapters (see table 1.1).

In the following sections, each of the six sub research questions are answered and findings are evaluated. Next, limitations of this research are discussed. Finally, a reflection on this research is provided by placing research results and implications in a broader context, and directions for future research are described.

8.1 CONTRIBUTIONS AND EVALUATION

The research in this dissertation is structured around six research questions. This section answers each of the questions, and briefly evaluates the findings that correspond to each of the questions.

RQ 1 *— What is the concept of software operation knowledge?*

In answer to this research question, a software operation knowledge definition and reference framework are proposed. The definition is composed of four types of operation knowledge that encompass and categorize the plethora of definitions and metrics that exist to respectively describe and measure software operation. Furthermore, the software operation knowledge reference framework defines the life cycle of software operation knowledge and illustrates (perspectives on) potential uses of such knowledge in the improvement of a vendor's products and internal software processes. Both the software operation knowledge definition and reference framework were evaluated and refined through the results of focus group discussions and extensive case studies at three software-producing organizations. The utility of the SOK framework is demonstrated, and the demand for such a guiding substrate regarding SOK utilization is confirmed. With the SOK definition and reference framework, meaning and potential of the software operation knowledge concept are positioned.

RQ 2 *— How can software operation knowledge practices within software ecosystems be classified?*

A classification of successful operational software ecosystem practices that may help software-producing organizations to effectively utilize and propagate knowledge of the in-the-field operation of their software, and therewith address challenges that result from ecosystem participation, provides the answer to this research question. The classification is constructed based on case studies with four vendors that successfully address challenges resulting from software ecosystem participation through utilization and propagation of software operation knowledge. Based on key derivations from these case studies, software ecosystem orchestration practices and software operation knowledge propagation practices are classified along dimensions of software ecosystem scope level and ecosystem life cycle phase. The classification illustrates the value and importance of software operation knowledge within the context of software ecosystems.

RQ 3 *— How can software maintenance effort be reduced through generic recording and visualization of operation of deployed software?*

The answer to this research question is a technique for generic acquisition and presentation of software operation knowledge. The technique enables soft-

ware vendors to acquire SOK independent of target software, allows vendors to get a uniform insight in operation of their software in the field, and contributes to reduction of software maintenance effort. Furthermore, a prototype tool is developed that implements this technique. Utility and effectiveness of the technique are evaluated based on a field study (consisting of two case studies and an experiment) with the tool, as well as focus group discussions regarding the technique and the tool. Based on empirical evaluation results, the SOK acquisition and presentation technique is considered to effectively reduce maintenance effort. The tool demonstrates that software operation can be recorded, visualized and replayed independently from target software. Furthermore, the tool increases comprehension of in-the-field software operation, and is considered to reduce the time needed to analyze software operation failures.

RQ 4 *— How can product software processes effectively be improved with acquired information of in-the-field software operation?*

This research question is answered through a template method for integration of software operation knowledge in existing software processes, and a set of four lessons learned that was created based on an action research study of ten months, during which the method was instantiated for three software processes of a European software vendor. Using the method, vendors are supported in (1) identification of relevant and valuable operation information, (2) analysis of target processes and their integration environment, (3) integration of selected information in, and transformation of, target product software processes, and (4) presentation of integrated operation information. The method demonstrates how product software processes can be improved pragmatically but measurably, without adhering to strict requirements from cumbersome maturity models or process improvement frameworks. The resulting lessons learned serve as guiding directions for future uses of the template method.

RQ 5 *— Can prioritization of software maintenance tasks be improved through the concept of a software operation summary?*

In answer to this research question, the concept of a software operation summary (abbreviated SOS; a SOK presentation medium that is based on recent in-the-field operation of a vendor's software) is introduced as a strive to support software processes by providing software operation knowledge, more specifically, to improve prioritization of software maintenance tasks by fostering the reach of consensus on such prioritization. Soundness and validity of the summary have been evaluated through an extensive survey among Dutch product

software vendors as well as a case study at a European software vendor. Results confirm (1) the lack of consensus experienced between different employee roles, (2) the demand for a software operation summary, particularly in preparation of (sprint) planning, (software) development and bug fixing activities, and (3) that a software operation summary is expected to foster reach of consensus. We demonstrate how crash report data can be identified, such that a software operation summary based on these data fosters the reach of consensus on prioritization of software maintenance tasks. Software vendors can instantiate the SOK integration template method (see chapter 5) to tailor the contents of an SOS, and its underlying data types, to their needs and preferences.

> ***RQ 6*** — *How can responsiveness to changes in service performance and usage be increased through utilization of knowledge of in-the-field software operation?*

In answer to this research question, the concept of service knowledge utilization (SKU) is introduced as an approach to increase a software vendor's flexibility, and responsiveness to changes in the performance and usage of its service-based, online software. Furthermore, a SOK presentation tool is presented that generates SKU reports, which are composed of three SKU indices that express service performance, usability and client utilization. Both the tool and SKU reports are empirically evaluated through an extensive case study at a Dutch software vendor. Evaluation results show that the SKU report is composed of metrics and indices that are valuable to software vendors. More specifically, results indicate that the tool contributes to the responsiveness of a software vendor to changes in software performance and usage: the reports it generates, support informed decision making in the areas of software development and software maintenance. The SKU report, including its metrics and indices, therefore is our answer to this research question.

8.2 IMPLICATIONS

This dissertation research contributes to several areas in both research and industry. Two main implications of this research are described next.

8.2.1 Exposing Software Operation Knowledge in the Product Software Domain

The research carried out in this dissertation is an important step in positioning the concept of software operation knowledge in both the industrial and scien-

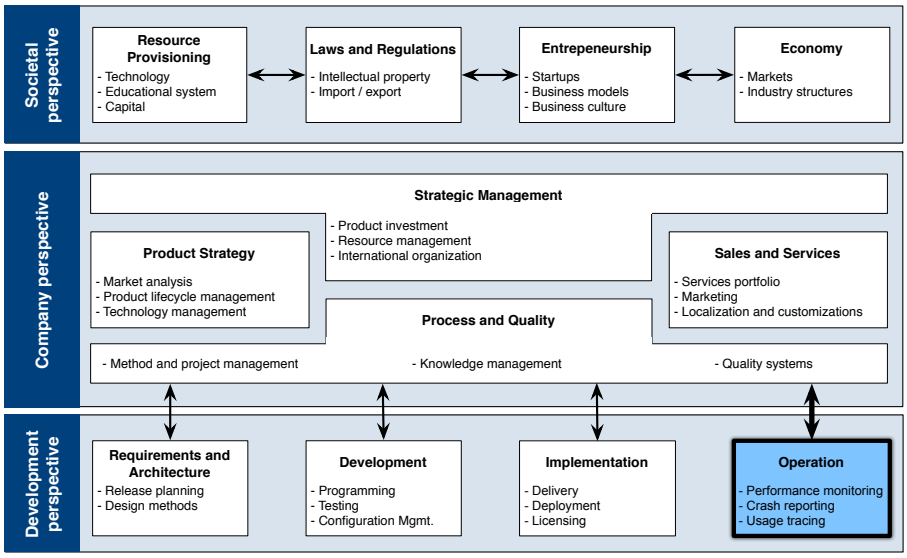


Figure 8.1 Updated research framework for product software (adapted from [Xu and Brinkkemper 2007, Jansen 2007])

tific domains of product software. By relating parties, perspectives and life cycle processes with software operation knowledge, and by positioning those in the software operation knowledge framework, the software operation knowledge concept has been exposed in the scientific and industrial domains of product software.

First, the software operation knowledge framework has contributed to product software vendors' awareness of the potential of software operation knowledge. The framework enables vendors to position functioning software processes in the life cycle of software operation knowledge, and identify which life cycle process implementations are lacking or need to be improved. This research provides guiding incentives that may encourage and inspire vendors to implement processes of the software operation knowledge life cycle (e.g., the software operation knowledge acquisition and presentation technique and tool, the software operation summary and the SKU report). This research demonstrates that software operation knowledge forms a source of impulses for process (e.g. in terms of efficiency) and product software improvement (e.g. in terms of performance, quality, usability, appreciation, etc.).

Second, the software operation knowledge framework has contributed to awareness of the concept of software operation knowledge in the product software research domain: in-the-field software operation forms a new and emerging, distinct source of research opportunities and industry innovations. As a recognition of the potential of the software operation knowledge in future research initiatives and industry practices, the research framework for product software has been updated to include the 'operation' state of software in its development perspective (see figure 8.1). Further improvements will be added to the framework, to cover and reflect future industry developments as well as information systems research areas.

8.2.2 Improving Process Improvement

Apart from positioning software operation knowledge in the product software domains, this research positions the concept of software operation knowledge in the domain of software process improvement by establishing it as an instrument for process improvement. As stated in section 1.3, several prescriptive software process improvement approaches such as CMMI are found too time-consuming or too complex to comprehend, implement and maintain effectively [Kuilboer and Ashrafi 2000, Staples *et al.* 2007, Smite and Gencel 2009].

The SOK integration template method describes what (rather than prescribing how) software vendors can do to structurally integrate operation information in their software processes. By distinguishing and describing eight template activities preparatory to actual integration of software operation information, and by incorporating these activities and corresponding template concepts in the template method for integration of operation information, the software operation knowledge concept has been exposed in the domain of process improvement.

We have demonstrated that through instantiation of the template method, vendors can improve their functioning software processes through operation information that is already available — in a pragmatic way but with measurable results, without introducing significant overhead and without adhering to strict requirements from cumbersome maturity models or process improvement frameworks. For example, using the template method, the number of incoming crash reports at CADComp was reduced with about 45% (see chapter 5). Concluding, the template method is therefore a powerful structure for software process improvement.

8.3 REFLECTION

When comparing the research approaches in this dissertation, particular patterns can be observed. First, half of the research questions are answered through multiple research methods. Second, to a certain extent, each of the artifacts detailed in these chapters are the result of interpretive research: we have studied real-world phenomena, and developed artifacts that help to understand or address certain phenomena.

As a reflection on this dissertation, we highlight strengths and potential risks of using multiple research methods for (empirical) evaluation of artifacts. Furthermore, the concept of hermeneutics is positioned in the context of interpretive information systems research, and stages of constructivist hermeneutics are identified based on the research that forms the basis of the artifacts presented in this dissertation.

8.3.1 Mixed-method Studies in Information Systems Research

A fundamental question for information systems researchers is when to combine different research methods to perform their studies [Falconer and Mackay 1999]. Researchers both inside and outside the information systems research community have attempted to answer this question [Gable 1994, Sandelowski 2000, Jansen and Brinkkemper 2008]. As a reflection on the mixed-method research that is part of this dissertation, we highlight strengths and risks of the chapters that present such research. Table 8.1 lists the strengths and risks of combining focus group and case study research methods (SOK framework), field study and focus group research methods (SOK acquisition and presentation technique and tool) as well as survey and case study research methods (software operation summary).

As illustrated by this reflection on mixed-method research, we have observed that combining quantitative and qualitative research methods allows for early establishment of artifact viability and soundness, as well as fine-grained estimation of artifact feasibility. However, both the quantitative and qualitative methods used should respectively involve enough respondents and cases to limit bias and reach broad evaluation of the artifact. Also, researchers should be aware that artifact feasibility may be found insufficient at a too late stage, particularly when a quantitative evaluation is performed prior to qualitative evaluation. We conclude that combination of different research methods allows for broad evaluation of artifacts, which may increase viability, soundness and validity of these artifacts.

Artifact	Research methods	Strengths	Risks
SOK framework (chapter 2)	a) Focus group b) Case study	• Viability and soundness of artifact are confirmed at an early stage • Validity and applicability of artifact are further refined through input from industry	• Bias or experience lack of focus group participants might lead to distorted case selection • Limited number of cases might result in a conflict between confirmed viability and resulting applicability
SOK acquisition and presentation technique, SOK acquisition and presentation tool (chapter 4)	a) Field study b) Focus group	• Diverse artifact flaws are discovered and can be resolved at an early stage • Real-world feasibility of stable artifact is subsequently estimated by management-level opinions	• Feasibility of stable artifact may be considered insufficient at a too late stage • Artifact stability requirements of focus group participants might not correspond with those of the researchers
Software operation summary (chapter 6)	a) Survey b) Case study	• Need for artifact, as well as the issues it addresses are confirmed at a broad scale • Validity and applicability of artifact are further refined through input from industry	• Clustered or limited number of survey respondents might result in disproportional confirmation of issues, leading to insignificant case study results • Artifact validity might be restricted because of limited representation of survey outcomes by case study

Table 8.1 Strengths and risks of mixed-method research in this dissertation

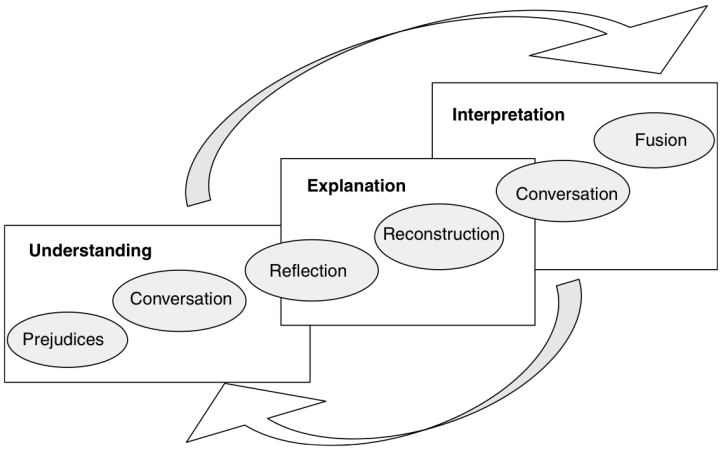


Figure 8.2 Hermeneutic framework for practical research [Cole and Avison 2007]

8.3.2 Constructivist Hermeneutics in Information Systems Research

Put simply, hermeneutics is a theory of interpreting texts. It is engaged in two tasks: ascertaining the exact meaning-content of a word or phrase, and defining guidelines to facilitate explication of interpretive research [Bleicher 1980]. As with many philosophical traditions, hermeneutics is an umbrella term for many different approaches. Whereas *phenomenological* hermeneutics aims at a faithful description of the lived experience and is accomplished by a bracketing of the researcher's frame of reference [Van Manen 1997], *constructivist* hermeneutics acknowledges the embedded nature of the researcher's frame as the beginning point in the process of coming to understand and interpret the phenomena under study [Cole and Avison 2007]. One of the primary aims of constructivist hermeneutics is to enact a methodology based on the recognition that every research act is an act of interpretation [Maturana 1980]. Unlike grounded theory, constructivist hermeneutics does not aspire to build a theory that can explain the relationships between a set of propositions about some phenomena that have been repeatedly tested, or that are widely accepted. Instead, attempts are made to develop a framework of understanding that outlines a set of assumptions, concepts, and practices that constitute a way of viewing reality, such as the hermeneutic framework for practical research of Cole and Avison [Cole and Avison 2007] (see figure 8.2).

The benefits of a detailed hermeneutic reflection as part of information system research are twofold: first, it supports researchers in achieving deeper levels of understanding of their research efforts, as well as the phenomena being studied. Second, such a reflection makes the researcher's interpretive process more transparent to other researchers, and therewith increases repeatability of interpretive research. Despite these benefits, hermeneutics is neither well-accepted nor much practiced in information systems research [Cole and Avison 2007]. The lack of formal structures for conducting hermeneutic research, the difficulty in understanding and correctly using its technical language, and the time needed to learn how to perform hermeneutic reflection have reduced its attractiveness compared with, or as a component of, case study research, action research and other established information systems research methods.

8.3.3 Stages of the Constructivist Hermeneutic Process

To reflect in a hermeneutic manner on the constructive process of the artifacts presented in this dissertation research, we used the hermeneutic framework for practical research of Cole and Avison that is depicted in figure 8.2. During this reflective process, however, we discovered several imperfections in their framework. We made several adaptions to the framework, which are represented by our constructivist hermeneutic framework depicted in figure 8.3.

In the first place, similar to the framework of Cole and Avison, three rectangular areas cover the so-called hermeneutic circle of unconscious understanding and situated behavior as suggested by Heidegger [Heidegger 1927]. Spirals of understanding arise from interpretations of executed actions or comments [Cole and Avison 2007]. The purpose of this hermeneutic circle is to reflectively accept or reject aspects of 'fore-knowledge', which can be respecified for theoretical development [Butler 1998, Cole and Avison 2007], ultimately resulting in theory.

Our adaptions to the framework of Cole and Avison are threefold. First, the constructivist hermeneutic framework explicates the concepts of fore-knowledge (e.g. observations, opinions, prejudices, etc.) and theory (e.g. systemization, categorization, classification, etc.). The fore-knowledge and theory concepts are respectively positioned as inputs and outputs of the hermeneutic circle, which is composed of stages of constructivist hermeneutics. Through these stages, using reasoning, theory is built to understand, predict or control new phenomena in terms of what we already understand.

Second, the explication stage represents the initial explication of considered fore-knowledge, i.e., the position of the researcher with relation to the phenom-

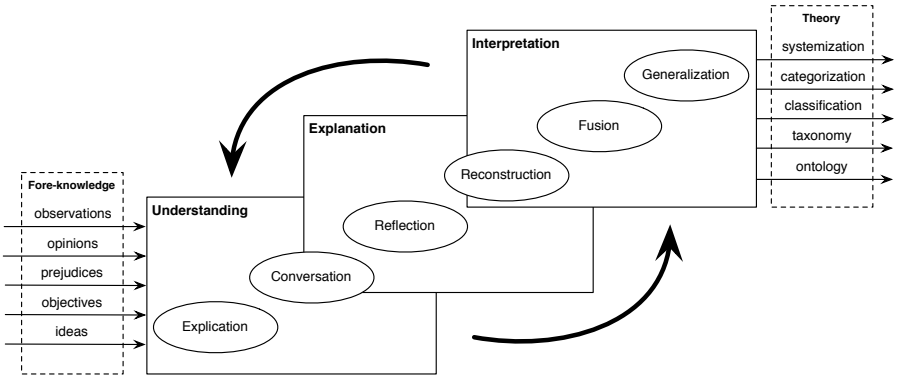


Figure 8.3 Constructivist hermeneutic framework (adapted from [Heidegger 1927, Cole and Avison 2007])

ena of interest, research motivations as well as own values and perceptions. The explication stage furthermore covers formulating lines of inquiry, structuring the research approach and choosing data sources and analytical strategies (in their article, the authors describe these steps as the 'preparation' phase, but this phase is missing in their figure). After the explication stage, initial understanding of fore-knowledge is established.

Third, the generalization stage represents generalization of interpreted observations, ideas, etc. into new concepts constituting the resulting theory. Generalization is an essential property of valuable theory: it refers to the validity of a theory in a setting different from the one where it was empirically tested and confirmed. Since the field of information systems covers both science and profession, the generalizability of an information systems theory to different settings is important not only for purposes of basic research, but also for purposes of managing and solving problems that corporations and other organizations experience in society [Lee and Baskerville 2003].

The constructivist hermeneutic framework reflects the constructivist hermeneutic process that we experienced throughout the construction of the artifacts that are presented in this dissertation. Furthermore, it clarifies the relation with the information systems research framework [Hevner *et al.* 2004]: using fore-knowledge from the knowledge base, the hermeneutic cycle is executed in parallel with development and justification of information systems artifacts and theory, which can be applied in the appropriate environment and potentially can be added to the knowledge base (see figure 1.4).

Through the framework, we attempt to increase maturity and transparency of reflections on future information systems research, and therewith the repeatability of such research. We are currently validating an alternative hermeneutic cycle (*dimensioning*, *categorization*, *naming*, *instantiation*, *presentation and beautification*, *revising*) that is based on the constructivist hermeneutic process we experienced throughout the construction of the classification of identified orchestration and propagation practices (see table 3.3). Based on two versions of this classification, figure 8.4 illustrates *dimensioning*, *categorization* and *naming* phases of this alternative hermeneutic cycle. Dimensioning refers to identification and revision of dimensions (e.g. the 'Focus' dimension in figure 8.4), categorization refers to identification and revision of (sub) categories (e.g. distribution of the 'Ecosystem' and 'Actor' categories in figure 8.4) and naming refers to establishing correct, consistent, complete and esthetically pleasing textual representations of dimensions, categories and other aspects (e.g. renaming of 'Orchestration Phase' in 'SECO Orchestration Phase' in figure 8.4). Empirical validation of our hermeneutic cycle will be subject of future research.

8.4 LIMITATIONS AND FUTURE RESEARCH

Despite the attractiveness of the findings of this dissertation research, the research is subject to some limitations that allude to carefulness when interpreting these findings. Research limitations give rise to further research directions, which are addressed subsequently.

8.4.1 Scientific Scope

Two main limitations of this dissertation research can be identified in terms of the empirical evaluation of the research.

First, the empirical evaluation of the artifacts presented in this research is limited to software vendors headquartered in the Netherlands. Although several of those vendors have offices outside the Netherlands (some even outside Europe), and although we believe this scope is sufficiently large for our study purposes, additional empirical evaluation at software vendors outside the Netherlands is needed to further demonstrate the value of the artifacts in other countries. Comparing observations across countries, markets and software application types, might well contribute to the overall soundness and general applicability of the software operation knowledge acquisition and presentation tool, as well as the software operation knowledge integration template

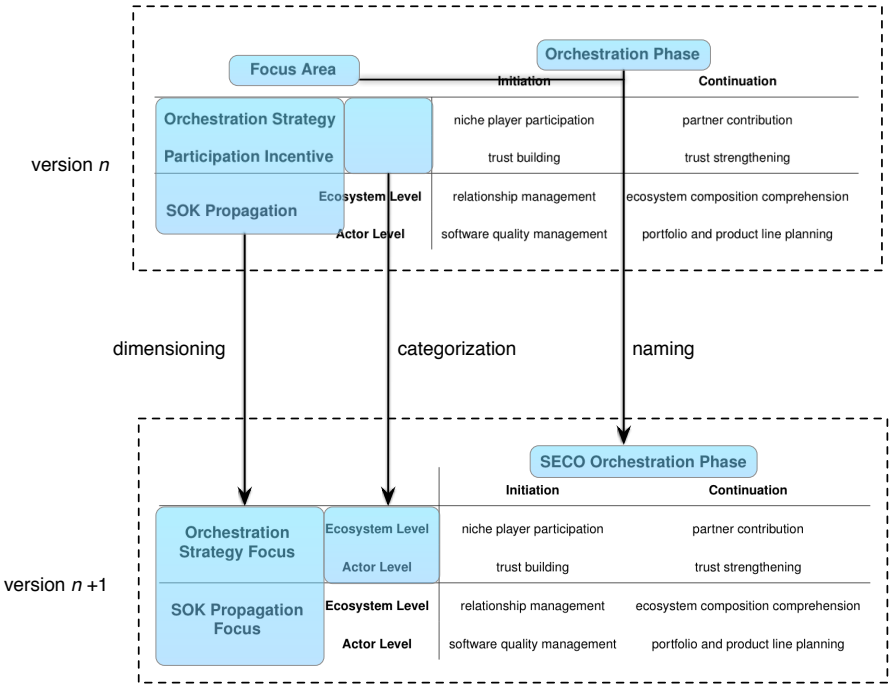


Figure 8.4 Two versions of the classification in table 3.3 illustrating dimensioning, categorization and naming phases

method. Simultaneously, this provides opportunities for further refinement, extension and specification of the artifacts presented in this dissertation.

The second limitation of this dissertation research is related to its explorative nature. Although this dissertation forms an important and significant step in (1) defining the concept of software operation knowledge and its life cycle, (2) exposing the viability and feasibility of process improvement through knowledge of in-the-field software operation and (3) demonstrating and evaluating such improvement empirically, we have only scratched the surface of quantifying the value of software operation knowledge in terms of measurable increases in inter alia process efficiency, employee productivity and software vendor profitability. Further research, particularly development and empirical evaluation of existing and new artifacts, is needed to mitigate this limitation.

An interesting future research avenue would be the establishment of situational factors [Bekkers *et al.* 2008] for SOK identification, acquisition, integration, presentation and utilization, to identify what are ingredients for mature

implementations of these SOK life cycle processes. Based on these situational factors, a software operation knowledge maturity quick scan can then be developed, which allows for situational (1) measurement of a software vendor's maturity in terms of each of the software operation knowledge life cycle processes and (2) identification of a vendor's processes that can be improved the most through integration of acquired software operation information.

8.4.2 Industrial Scope

From a practical perspective, the artifacts presented in this dissertation research require careful implementations, and are subject to limitations. Concerning the software operation knowledge framework, for example, the transformations between software operation data, operation information and operation knowledge might be hard to distinguish in certain circumstances (e.g. when a software vendor implementing the framework decides not to implement the optional integration process, but directly visualize and present acquired software operation data). Furthermore, the software operation knowledge integration template method may initially be found challenging to instantiate by software vendors, since vendors may need to adopt the concept of a template method, and because the method requires both operation information and target processes to be available before instantiating. Larger software vendors in particular might initially expect more prescription from a process improvement approach, and might need to accommodate to delegating template method instantiation to smaller teams that are situated lower in the organization.

An interesting avenue for further research from a practical perspective is the development of a generic, and thus situational, software operation knowledge infrastructure that integrates all SOK life cycle processes. The infrastructure should provide methodical and tool support for SOK identification, acquisition, integration, presentation and utilization, to allow software vendors to implement the full SOK life cycle independently from their software and software processes. As an example, the infrastructure could be composed of an extended Nuntia tool that allows for on-the-fly instrumentation of software (i.e., during in-the-field operation). Software feedback instrumentation could therewith be automated, further eliminating the difference between software requiring deployment and online, service-based software (e.g. web applications). Finally, the infrastructure could support vendors in propagation of acquired operation knowledge to software ecosystem partner organizations.

Bibliography

[Abran *et al.* 2004] Abran, A., Moore, J., Bourque, P., Dupuis, R., and Tripp, L. (2004). Guide to the Software Engineering Body of Knowledge — SWEBOK. (Cited on page 4)

[Van Angeren *et al.* 2011] Van Angeren, J., Kabbedijk, J., Jansen, S., and Popp, K. M. (2011). A Survey of Associate Models used within Large Software Ecosystems. In *IWSECO '11: Proceedings of the 3rd International Workshop on Software Ecosystems*, (pp. 1–13). (Cited on page 46)

[Barry *et al.* 2007] Barry, E. J., Kemerer, C. F., and Slaughter, S. A. (2007). How software process automation affects software evolution: a longitudinal empirical analysis. *Journal of Software Maintenance and Evolution: Research and Practice*, *19*, 1–31. (Cited on page 9)

[Baskerville and Pries-Heje 1999] Baskerville, R. L. and Pries-Heje, J. (1999). Grounded action research: a method for understanding it in practice. *Accounting, Management and Information Technologies*, *9*(1), 1–23. (Cited on page 15)

[Baskerville 1999] Baskerville, R. L. (1999). Investigating information systems with action research. *Communications of the AIS*, *2*. (Cited on page 15)

[Beecham *et al.* 2003] Beecham, S., Hall, T., and Rainer, A. (2003). Software process improvement problems in twelve software companies: An empirical analysis. *Empirical Software Engineering*, *8*, 7–42. (Cited on page 8)

[Bekkers *et al.* 2008] Bekkers, W., Van de Weerd, I., Brinkkemper, S., and Mahieu, A. (2008). The Influence of Situational Factors in Software Product Management: An Empirical Study. (pp. 41–48). IEEE Computer Society Press. (Cited on page 179)

[Beynon-Davies 2010] Beynon-Davies, P. (2010). The enactment of significance: a unified conception of information, systems and technology. *European Journal of Information Systems*, *19*(4), 389–408. (Cited on page 12)

[Bleicher 1980] Bleicher, J. (1980). *Contemporary Hermeneutics: Hermeneutics as Method, Philosophy, and Critique*. Routledge. (Cited on page 175)

[Von Bochmann *et al.* 2001] Von Bochmann, G., Kerhervé, B., Lutfiyya, H., Salem, M.-V. M., and Ye, H. (2001). Introducing QoS to Electronic Commerce Applications. In *ISEC '01: Proceedings of the Second International Symposium on Topics in Electronic Commerce*, (pp. 138–147). Springer. (Cited on page 147)

[Bøegh 2008] Bøegh, J. (2008). A New Standard for Quality Requirements. *IEEE Software*, *25*(2), 57–63. (Cited on page 27)

[Booch 2004] Booch, G. (2004). *Object-Oriented Analysis and Design with Applications (3rd Edition)*. Addison-Wesley Longman Publishing. (Cited on page 144)

[Bosch and Bosch-Sijtsema 2010] Bosch, J. and Bosch-Sijtsema, P. (2010). From integration to composition: On the impact of software product lines, global development and ecosystems. *Journal of Systems and Software*, *83*(1), 67–76. (Cited on page 47)

[Bosch 2009] Bosch, J. (2009). From Software Product Lines to Software Ecosystems. In *SPLC '09: Proceedings of the 13th International Conference on Software Product Lines*. Springer. (Cited on pages 46 and 51)

[Bowring *et al.* 2003] Bowring, J., Orso, A., and Harrold, M. J. (2003). Monitoring Deployed Software Using Software Tomography. *SIGSOFT Software Engineering Notes*, *28*(1), 2–9. (Cited on pages 24, 25, 31, 73, and 162)

[Brelsford *et al.* 2002] Brelsford, H., Toot, M., Kiri, K., and Van Steenburgh, R. (2002). *Connecting to Customers*. (Cited on pages 24 and 144)

[Brown *et al.* 2002] Brown, J. S., Durchslag, S., and Hagel, J. (2002). Loosening up: How process networks unlock the power of specialization. *The McKinsey Quarterly: Risk and Resilience*, (2), 59–69. (Cited on page 48)

[Butler 1998] Butler, T. (1998). Towards a hermeneutic method for interpretive research in information systems. *Journal of Information Technology*, *13*(4), 285–300. (Cited on page 176)

[CEIP] Microsoft Customer Experience Improvement Program. `http://www.microsoft.com/products/ceip/`. (Cited on page 51)

[Clause and Orso 2007] Clause, J. and Orso, A. (2007). A Technique for Enabling and Supporting Debugging of Field Failures. In *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*, (pp. 261–270). IEEE Computer Society Press. (Cited on pages 31 and 73)

[Cole and Avison 2007] Cole, M. and Avison, D. (2007). The potential of hermeneutics in information systems research. *European Journal of Information Systems*, *16*(6), 820–833. (Cited on pages 175, 176, and 177)

[Conradi and Fuggetta 2002] Conradi, R. and Fuggetta, A. (2002). Improving Software Process Improvement. *IEEE Software*, *19*(4), 92–99. (Cited on pages 8 and 98)

[Cornelissen *et al.* 2008] Cornelissen, B., Zaidman, A., Holten, D., Moonen, L., Van Deursen, A., and Van Wijk, J. J. (2008). Execution trace analysis through massive sequence and circular bundle views. *Journal of Systems and Software*, *81*(12), 2252–2268. (Cited on page 74)

[Dangle *et al.* 2005] Dangle, K. C., Larsen, P., Shaw, M., and Zelkowitz, M. V. (2005). Software Process Improvement in Small Organizations: A Case Study. *IEEE Software*, *22*, 68–75. (Cited on page 99)

[Darke *et al.* 1998] Darke, P., Shanks, G. G., and Broadbent, M. (1998). Successfully completing case study research: combining rigour, relevance and pragmatism. *Information Systems Journal*, *8*(4), 273–290. (Cited on page 15)

[Davis *et al.* 1988] Davis, A. M., Bersoff, H., and Comer, E. R. (1988). A Strategy for Comparing Alternative Software Development Life Cycle Models. *IEEE Transactions on Software Engineering*, *14*(10), 1453–1461. (Cited on page 7)

[Davison *et al.* 2004] Davison, R. M., Martinsons, M. G., and Kock, N. (2004). Principles of canonical action research. *Information Systems Journal*, *14*, 65–86. (Cited on pages 15, 100, 101, and 114)

[Dawes] Dawes, J. Do Data Characteristics Change According to the number of scale points used? An experiment using 5-point, 7-point and 10-point scales. *International Journal of Market Research*, *50*(1), 61–77. (Cited on page 150)

[Denning 1997] Denning, P. J. (1997). A New Social Contract for Research. *Communications of the ACM*, *40*, 132–134. (Cited on page 12)

[Dorling 1993] Dorling, A. (1993). SPICE: Software process improvement and capability dEtermination. *Software Quality Journal*, *2*, 209–224. (Cited on page 8)

[Dul and Hak 2008] Dul, J. and Hak, T. (2008). *Case Study Methodology in Business Research*. Butterworth-Heinemann. (Cited on page 15)

[Easterbrook *et al.* 2008] Easterbrook, S., Singer, J., Storey, M.-A., and Damian, D., Selecting Empirical Methods for Software Engineering Research, In Shull, F., Singer, J., and Sjøberg, D. I. K. (Eds.), *Guide to Advanced Empirical Software Engineering*, (pp. 285–311). (Cited on pages 73 and 82)

[Ebert and Dumke 2007] Ebert, C. and Dumke, R. (2007). *Software Measurement*. Springer. (Cited on pages 9, 10, and 25)

[Eisenhardt 1989] Eisenhardt, K. M. (1989). Building Theories from Case Study Research. *The Academy of Management Review*, *14*(4), 532–550. (Cited on page 15)

[Falconer and Mackay 1999] Falconer, D. J. and Mackay, D. R. (1999). The Key to the Mixed Method Dilemma. In *ACIS '99: Proceedings of the 10th Australasian Conference on Information Systems*, (pp. 286–297). School of Communications and Information Management. (Cited on page 173)

[Farbey and Finkelstein 1999] Farbey, B. and Finkelstein, A. (1999). Exploiting software supply chain business architecture: a research agenda. In *EDSER '99: 1st Workshop on Economics-Driven Software Engineering Research*. IEEE Computer Society Press. (Cited on pages 46 and 51)

[Farrell and Kreger 2002] Farrell, J. and Kreger, H. (2002). Web services management approaches. *IBM Systems Journal*, *41*(2). (Cited on pages 144, 147, and 152)

[Filman *et al.* 2005] Filman, R. E., Elrad, T., Clarke, S., and AKŞIT, M. (2005). *Aspect-Oriented Software Development*. Addison-Wesley. (Cited on page 157)

[Fitzgerald and O'Kane 1999] Fitzgerald, B. and O'Kane, T. (1999). A Longitudinal Study of Software Process Improvement. *IEEE Software*, *16*(3), 37–45. (Cited on pages 8 and 99)

[Fraiteur 2008] Fraiteur, G. (2008). User-friendly aspects with compile-time imperative semantics in .NET. Presented at the 7th International Conference on Aspect-Oriented Software Development (AOSD 2008). (Cited on page 157)

[Fritz and Murphy 2010] Fritz, T. and Murphy, G. C. (2010). Using Information Fragments to Answer the Questions Developers Ask. In

ICSE '10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering — Volume 1, (pp. 175–184). ACM Press. (Cited on page 139)

[Gable 1994] Gable, G. G. (1994). Integrating Case Study and Survey Research Methods: An Example in Information Systems. *European Journal of Information Systems*, *3*(2), 112–126. (Cited on page 173)

[Gawer and Cusumano 2002] Gawer, A. G. and Cusumano, M. A. (2002). *Platform Leadership*. Harvard Business School Press. (Cited on page 48)

[Glerum *et al.* 2009] Glerum, K., Kinshumann, K., Greenberg, S., Aul, G., Orgovan, V., Nichols, G., Grant, D., Loihle, G., and Hunt, G. C. (2009). Debugging in the (Very) Large: Ten Years of Implementation and Experience. In *SOSP '09: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, (pp. 103–116). ACM Press. (Cited on pages 51, 72, 98, and 120)

[Google Breakpad 2010] (2010). google-breakpad – Crash reporting. `http://code.google.com/p/google-breakpad/`. (Cited on pages 120, 126, and 128)

[Gray 1986] Gray, J. (1986). Why Do Computers Stop and What Can Be Done About It? In *SRDS '86: Proceedings of the 5th Symposium on Reliability in Distributed Software and Database Systems*, (pp. 3–12). IEEE Computer Society Press. (Cited on pages 58 and 60)

[GWO] Google Website Optimizer. `http://www.google.com/websiteoptimizer/`. (Cited on page 24)

[Gyimóthy *et al.* 2005] Gyimóthy, T., Ferenc, R., and Siket, I. (2005). Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction. *IEEE Transactions on Software Engineering*, *31*(10), 897–910. (Cited on page 27)

[Häcki and Lighton 2001] Häcki, R. and Lighton, J. (2001). The future of the networked company. *The McKinsey Quarterly*, (3), 26–39. (Cited on pages 47, 48, 50, and 51)

[Hall *et al.* 2002] Hall, T., Rainer, A., and Baddoo, N. (2002). Implementing software process improvement: An empirical study. *Software Process: Improvement and Practice*, *7*(1), 3–15. (Cited on page 8)

[Hays 2003] Hays, P. A., Case study research, In DeMarrais, K. B. and Lapan, S. D. (Eds.), *Foundations for Research: Methods of Inquiry in Education and the Social Sciences*. (Cited on page 15)

[Heidegger 1927] Heidegger, M. (1927). *Sein und Zeit* (Being and Time). Max Niemeyer Verlag. (Cited on pages 176 and 177)

[Hevner *et al.* 2004] Hevner, A. R., March, S. T., Park, J., and Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly*, *28*(1), 75–105. (Cited on pages 12, 13, 14, 16, 73, 100, and 177)

[Hilbert and Redmiles 2000] Hilbert, D. M. and Redmiles, D. F. (2000). Extracting Usability Information from User Interface Events. *ACM Computing Surveys*, *32*. (Cited on page 9)

[Iansiti and Levien 2004a] Iansiti, M. and Levien, R. (2004). Strategy as Ecology. *Harvard Business Review*, *82*, 68–78. (Cited on page 48)

[Iansiti and Levien 2004b] Iansiti, M. and Levien, R. (2004). *The Keystone Advantage: What the New Dynamics of Business Ecosystems Mean for Strategy, Innovation, and Sustainability*. Harvard Business School Press. (Cited on page 48)

[Ibrahim *et al.* 2007] Ibrahim, M. H., Holley, K., Josuttis, N. M., Michelson, B., Thomas, D. A., and DeVadoss, J. (2007). The future of soa: what worked, what didn't, and where is it going from here? In *OOPSLA '07: Proceedings of the 22nd ACM SIGPLAN conference on Object-Oriented Programming Systems and Applications*, (pp. 1034–1038). ACM Press. (Cited on page 144)

[ISO/IEC 2001] (2001). ISO/IEC 9126-1:2001: Software engineering – Product quality – Part 1: Quality model. International Organization for Standardization. (Cited on page 27)

[ISO/IEC 2006] (2006). ISO/IEC 14764:2006: Software Engineering — Software Life Cycle Processes — Maintenance. (Cited on pages 118, 119, 120, and 123)

[ISO/IEC 2008] (2008). ISO/IEC 12207:2008: Systems and software engineering — Software life cycle processes. International Organization for Standardization. (Cited on page 7)

[Iversen *et al.* 2004] Iversen, J. H., Mathiassen, L., and Nielsen, P. A. (2004). Managing Risk in Software Process Improvement: An Action Research Approach. *MIS Quarterly*, *28*(3). (Cited on page 99)

[Jansen *et al.* 2009] Jansen, S., Brinkkemper, S., and Finkelstein, A. (2009). A Sense of Community: A Research Agenda for Software Ecosystems. In *ICSE'09: Proceedings of the 31st International Conference on Software Engineering*. IEEE Computer Society Press. (Cited on pages 30, 47, and 51)

[Jansen *et al.* 2008] Jansen, S., Brinkkemper, S., and Helms, R. (2008). Benchmarking the Customer Configuration Updating Practices of Product Software Vendors. In *ICCBSS '08: Proceedings of the Seventh International Conference on Composition-Based Software Systems*, (pp. 82–91). IEEE Computer Society Press. (Cited on pages 3, 72, and 144)

[Jansen and Brinkkemper 2008] Jansen, S. and Brinkkemper, S., Applied Multi-Case Research in a Mixed-Method Research Project: Customer Configuration Updating Improvement, In Cater-steel, A. and Al-Hakim, L. (Eds.), *Information Systems Research Methods, Epistemology, and Applications*, (pp. 422). (Cited on pages 15, 16, and 173)

[Jansen *et al.* 2010] Jansen, S., Buts, W., Brinkkemper, S., and Van der Hoek, A. (2010). Benchmarking the Customer Configuration Updating Process of the International Product Software Industry. In *ICSP '10: Proceedings of the 2010 International Conference on Software Process*, (pp. 369–380). Springer. (Cited on page 3)

[Jansen *et al.* 2009] Jansen, S., Finkelstein, A., and Brinkkemper, S. (2009). Business Network Management as a Survival Strategy: A Tale of Two Software Ecosystems. In *IWSECO '09: Proceedings of the First International Workshop on Software Ecosystems*, (pp. 34–48). Springer. (Cited on pages 47 and 48)

[Jansen 2007] Jansen, S. (2007). *Customer Configuration Updating in a Software Supply Network*. PhD thesis, Universiteit Utrecht. (Cited on pages 6, 7, and 171)

[Johnson *et al.* 2007] Johnson, M. J., Ho, C.-W., Maximilien, E. M., and Williams, L. (2007). Incorporating Performance Testing in Test-Driven Development. *IEEE Software*, *24*(3), 67–73. (Cited on page 26)

[Jones *et al.* 2004] Jones, J. A., Orso, A., and Harrold, M. J. (2004). GAMMATELLA: visualizing program-execution data for deployed software. *Information Visualization*, *3*(3), 173–188. (Cited on page 74)

[Kaiser *et al.* 1988] Kaiser, G. E., Feiler, P. H., and Popovich, S. S. (1988). Intelligent Assistance for Software Development and Maintenance. *IEEE Software*, *5*, 40–49. (Cited on page 9)

[Kalepu *et al.* 2003] Kalepu, S., Krishnaswamy, S., and Loke, S. W. (2003). Verity: a QoS metric for selecting Web services and providers. In *WISEW '03: Proceedings of the 4th International Conference on Web Information Systems Engineering Workshops*, (pp. 131–139). IEEE Computer Society Press. (Cited on page 146)

[Kallepalli and Tian 2001] Kallepalli, C. and Tian, J. (2001). Measuring and Modeling Usage and Reliability for Statistical Web Testing. *27*(11), 1023–1036. (Cited on pages 9 and 28)

[Kettinger and Li 2010] Kettinger, W. J. and Li, Y. (2010). The infological equation extended: towards conceptual clarity in the relationship between data, information and knowledge. *European Journal of Information Systems*, *19*(4), 409–421. (Cited on page 9)

[Kim *et al.* 2011] Kim, D., Wang, X., Kim, S., Zeller, A., Cheung, S., and Park, S. (2011). Which Crashes Should I Fix First?: Predicting Top Crashes at an Early Stage to Prioritize Debugging Efforts. *IEEE Transactions on Software Engineering*. (Cited on page 139)

[Kitchenham *et al.* 2008] Kitchenham, B., Al-Khilidar, H., Babar, M., Berry, M., Cox, K., Keung, J., Kurniawati, F., Staples, M., Zhang, H., and Zhu, L. (2008). Evaluating guidelines for reporting empirical software engineering studies. *Empirical Software Engineering*, *13*, 97–121. (Cited on pages 73 and 82)

[Kitchenham and Pfleeger 2002] Kitchenham, B. A. and Pfleeger, S. L. (2002). Principles of Survey Research Part 3: Constructing a Survey Instrument. *SIGSOFT Software Engineering Notes*, *27*, 20–24. (Cited on pages 121 and 138)

[Kittlaus and Clough 2009] Kittlaus, H.-B. and Clough, P. N. (2009). *Software Product Management and Pricing: Key Success Factors for Software Organizations*. Springer. (Cited on page 47)

[Klein and Myers 1999] Klein, H. K. and Myers, M. D. (1999). A set of principles for conducting and evaluating interpretive field studies in information systems. *MIS Quarterly*, *23*, 67–93. (Cited on pages 16 and 65)

[Kristjánsson and Van der Schuur 2009] Kristjánsson, B. and Van der Schuur, H. (2009). A Survey of Tools for Software Operation Knowledge

Acquisition. Technical Report UU-CS-2009-028, Department of Information and Computing Sciences, Utrecht University. (Cited on page 73)

[Kuilboer and Ashrafi 2000] Kuilboer, J. P. and Ashrafi, N. (2000). Software process and product improvement: an empirical assessment. *Information and Software Technology*, *42*(1), 27–34. (Cited on pages 8, 98, and 172)

[Lee and Baskerville 2003] Lee, A. S. and Baskerville, R. L. (2003). Generalizing Generalizability in Information Systems Research. *Information Systems Research*, *14*, 221–243. (Cited on page 177)

[Lehman and Ramil 1999] Lehman, M. M. and Ramil, J. F. (1999). The impact of feedback in the global software process. *Journal of Systems and Software*, *46*, 123–134. (Cited on pages 9 and 25)

[Lehman 1996] Lehman, M. M. (1996). Feedback, Evolution and Software Technology. In *ISPW '96: Proceedings of the 10th International Software Process Workshop*, (pp. 101–103). IEEE Computer Society Press. (Cited on page 9)

[Lehtola and Kauppinen 2006] Lehtola, L. and Kauppinen, M. (2006). Suitability of Requirements Prioritization Methods for Market-driven Software Product Development. *Softw. Process: Improvement and Practice*, *11*(1), 7–19. (Cited on page 118)

[Lewis 2001] Lewis, L. (2001). *Managing Business and Service Networks*. Kluwer Academic Publishers. (Cited on page 149)

[Lippoldt and Stryszowski 2009] Lippoldt, D. and Stryszowski, P. (2009). *Innovation in the Software Sector*. OECD Publishing. (Cited on pages 3 and 5)

[Liu and Xu 2007] Liu, D. and Xu, S. (2007). New Quality Metrics for Object-Oriented Programs. In *SNPD '07: Proceedings of the 8th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing — Volume 03*, (pp. 870–875). IEEE Computer Society Press. (Cited on page 9)

[Mac OS X Reference Library 2010] (2010). Mac OS X Reference Library – TN2123: CrashReporter. `http://developer.apple.com/library/mac/#technotes/tn2004/tn2123.html`. (Cited on pages 120, 126, and 128)

[Madhavji *et al.* 2006] Madhavji, N. H., Fernandez-Ramil, J., and Perry, D. (2006). *Software Evolution and Feedback: Theory and Practice*. John Wiley & Sons. (Cited on pages 9, 10, 25, and 72)

[Van Manen 1997] Van Manen, M. (1997). From Meaning to Method. *Qualitative Health Research*, *7*(3), 345–369. (Cited on page 175)

[Mani and Nagarajan 2002] Mani, A. and Nagarajan, A. (2002). Understanding quality of service for Web services. `http://www.ibm.com/developerworks/java/library/ws-quality.html`. (Cited on pages 26 and 147)

[March and Smith 1995] March, S. T. and Smith, G. F. (1995). Design and natural science research on information technology. *Decision Support Systems*, *15*, 251–266. (Cited on page 12)

[Mathews 1987] Mathews, M. L., Hypercube software performance metrics, In *Hypercube Multiprocessors 1987*, (pp. 155–161). (Cited on page 9)

[Maturana 1980] Maturana, H., Man and society, In Benseler, F., Hejl, P., and Koch, W. (Eds.), *Autopoiesis, Communication, and Society: The Theory of Autropoietic Systems in the Social Sciences*, (pp. 11–31). (Cited on page 175)

[Memon *et al.* 2004] Memon, A., Porter, A., Yilmaz, C., Nagarajan, A., Schmidt, D., and Natarajan, B. (2004). Skoll: Distributed Continuous Quality Assurance. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, (pp. 459–468). IEEE Computer Society Press. (Cited on page 24)

[Menascé *et al.* 2001] Menascé, D. A., Barbará, D., and Dodge, R. (2001). Preserving QoS of E-commerce Sites Through Self-Tuning: A Performance Model Approach. In *EC '01: Proceedings of the 3rd ACM conference on Electronic Commerce*, (pp. 224–234)., New York, NY, USA. ACM Press. (Cited on pages 144 and 147)

[Mendelson and Ziegler 1999] Mendelson, H. and Ziegler, J. (1999). *Survival of the Smartest: Managing Information for Rapid Action and World-Class Performance*. John Wiley & Sons. (Cited on page 145)

[Messerschmitt and Szyperski 2003] Messerschmitt, D. G. and Szyperski, C. (2003). *Software Ecosystem: Understanding an Indispensable Technology and Industry*. MIT Press. (Cited on page 47)

[Microsoft Error Reporting 2006] (2006). How to: Configure Microsoft Error Reporting. `http://msdn.microsoft.com/en-us/library/bb219076(v=office.12).aspx`. (Cited on pages 120, 126, and 128)

[Microsoft .NET Framework] .NET Framework Developer Center. `http://msdn.microsoft.com/netframework/`. (Cited on pages 77, 81, and 82)

[Microsoft Online Crash Analysis 2005] (2005). Microsoft Online Crash Analysis – Error Report Contents Information. `http://oca.microsoft.com/en/dcp20.asp`. (Cited on pages 120, 126, and 128)

[Miler and Górski 2004] Miler, J. and Górski, J. (2004). Risk-driven Software Process Improvement — a Case Study. In *EuroSPI'04: Proceedings of the 11th European Conference on Software Process Improvement*. Springer. (Cited on page 99)

[MLTP] Mozilla Labs Test Pilot. `http://labs.mozilla.com/testpilot/`. (Cited on page 24)

[Mockus *et al.* 2005] Mockus, A., Zhang, P., and Li, P. L. (2005). Predictors of Customer Perceived Software Quality. In *ICSE '05: Proceedings of the 27th International Conference on Software Engineering*, (pp. 225–233). ACM Press. (Cited on page 9)

[Mono Project] The Mono Project. `http://mono-project.com/`. (Cited on page 77)

[Morgan 1996] Morgan, D. L. (1996). Focus groups. *Annual Review of Sociology*, *22*, 129–152. (Cited on page 16)

[Murphy 2004] Murphy, B. (2004). Automating Software Failure Reporting. *Queue*, *2*, 42–48. (Cited on page 9)

[Musa 1993] Musa, J. D. (1993). Operational profiles in software-reliability engineering. *IEEE Software*, *10*, 14–32. (Cited on page 9)

[Nachmanson *et al.* 2008] Nachmanson, L., Robertson, G., and Lee, B. (2008). Drawing Graphs with GLEE. *Graph Drawing*, 389–394. (Cited on page 81)

[Nagappan *et al.* 2006] Nagappan, N., Ball, T., and Zeller, A. (2006). Mining metrics to predict component failures. In *ICSE '06: Proceedings of the 28th International Conference on Software Engineering*, (pp. 452–461). ACM Press. (Cited on page 9)

[Nagappan *et al.* 2010] Nagappan, N., Zeller, A., Zimmermann, T., Herzig, K., and Murphy, B. (2010). Change Bursts as Defect Predictors. In *ISSRE '10:*

Proceedings of the 21st IEEE International Symposium on Software Reliability Engineering. IEEE Computer Society Press. (Cited on page 9)

[Nap 2011] Nap, C. (2011). Universiteit onderzoekt prioriteren onopgeloste bugs (*University researches prioritization of unsolved bugs*). Automatisering Gids. `http://www.automatiseringgids.nl/technologie/software/2011/4/onderzoek-naar-softwareonderhoud.aspx`. (Cited on page 127)

[Narayanasamy *et al.* 2005] Narayanasamy, S., Pokam, G., and Calder, B. (2005). BugNet: Continuously Recording Program Execution for Deterministic Replay Debugging. In *ISCA '05: Proceedings of the 32nd International Symposium on Computer Architecture*, (pp. 284–295). IEEE Computer Society Press. (Cited on page 73)

[Naumann *et al.* 1999] Naumann, F., Leser, U., and Freytag, J. C. (1999). Quality-driven Integration of Heterogenous Information Systems. In *VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases*, (pp. 447–458). Morgan Kaufmann Publishers Inc. (Cited on pages 147 and 148)

[Newcomer and Lomow 2004] Newcomer, E. and Lomow, G. (2004). *Understanding SOA with Web Services (Independent Technology Guides)*. Addison-Wesley. (Cited on page 144)

[Nielsen 1993] Nielsen, J. (1993). *Usability Engineering*. Academic Press. (Cited on page 148)

[Nuntia 2011] (2011). Nuntia — A Tool for Generic Binary Instrumentation. `http://nuntia.nl/`. (Cited on page 18)

[Nusayr and Cook 2009] Nusayr, A. and Cook, J. (2009). AOP for the Domain of Runtime Monitoring: Breaking Out of the Code-Based Model. In *DSAL '09: Proceedings of the 4th workshop on Domain-specific aspect languages*, (pp. 7–10). ACM Press. (Cited on pages 73 and 75)

[OECD 2010] (2010). OECD Information Technology Outlook 2010. OECD Publishing. (Cited on pages 3 and 5)

[OOPEC 2005] (2005). *The new SME definition — User guide and model declaration*. Enterprise and Industry Publications. Office for Official Publications of the European Communities. (Cited on page 127)

[Orso *et al.* 2002] Orso, A., Liang, D., Harrold, M. J., and Lipton, R. (2002). Gamma System: Continuous Evolution of Software after Deployment. In *ISSTA '02: Proceedings of the 2002 ACM SIGSOFT International Symposium on Software Testing and Analysis*, (pp. 65–69). ACM Press. (Cited on pages 24 and 31)

[Paint.NET Roadmap and Change Log] Paint.NET Roadmap and Change Log. `http://www.getpaint.net/roadmap.html`. (Cited on page 84)

[Papapetrou and Papadopoulos 2004] Papapetrou, O. and Papadopoulos, G. A. (2004). Aspect oriented programming for a component-based real life application: a case study. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, (pp. 1554–1558). ACM Press. (Cited on pages 153 and 157)

[Pettersson *et al.* 2008] Pettersson, F., Ivarsson, M., Gorschek, T., and Öhman, P. (2008). A practitioner's guide to light weight software process assessment and improvement planning. *Journal of Systems and Software*, *81*(6), 972–995. (Cited on pages 98 and 99)

[Pfleeger and Atlee 2009] Pfleeger, S. and Atlee, J. (2009). *Software Engineering: Theory and Practice*. Prentice Hall. (Cited on page 4)

[Pfleeger and Kitchenham 2001] Pfleeger, S. L. and Kitchenham, B. A. (2001). Principles of survey research — part 1: Turning lemons into lemonade. *SIGSOFT Software Engineering Notes*, *26*, 16–18. (Cited on page 16)

[Pigoski 1997] Pigoski, T. M. (1997). *Practical Software Maintenance: Best Practices for Managing Your Software Investment*. John Wiley & Sons. (Cited on pages 119 and 120)

[Porter 1993] Porter, A. A. (1993). Using measurement-driven modeling to provide empirical feedback to software developers. *Journal of Systems and Software*, *20*(3), 237–243. (Cited on page 9)

[Pressman 2010] Pressman, R. (2010). *Software Engineering: A Practitioner's Approach*. McGraw-Hill Higher Education. (Cited on page 4)

[Putrycz *et al.* 2005] Putrycz, E., Woodside, M., and Wu, X. (2005). Performance Techniques for COTS Systems. *IEEE Software*, *22*(4), 36–44. (Cited on pages 9 and 26)

[Ran 2003] Ran, S. (2003). A Model for Web Services Discovery With QoS. *ACM SIGecom Exchanges*, *4*(1), 1–10. (Cited on page 147)

[Rompaey *et al.* 2009] Rompaey, B. V., Bois, B. D., Demeyer, S., Pleunis, J., Putman, R., Meijfroidt, K., Dueñas, J. C., and García, B. (2009). SERIOUS: Software Evolution, Refactoring, Improvement of Operational and Usable Systems. In *CSMR '09: Proceedings of the European Conference on Software Maintenance and Reengineering*, (pp. 277–280). IEEE Computer Society Press. (Cited on page 25)

[Rout *et al.* 2007] Rout, T. P., El Emam, K., Fusani, M., Goldenson, D., and Jung, H.-W. (2007). SPICE in retrospect: Developing a standard for process assessment. *Journal of Systems and Software*, *80*, 1483–1493. (Cited on page 8)

[Sahai *et al.* 2001] Sahai, A., Ouyang, J., Machiraju, V., and Wurster, K. (2001). HP Laboratories — Specifying and Guaranteeing Quality of Service for Web Services through Real Time Measurement and Adaptive Control. `http://www.hpl.hp.com/techreports/2001/HPL-2001-134.html`. (Cited on page 147)

[Sandelowski 2000] Sandelowski, M. (2000). Combining Qualitative and Quantitative Sampling, Data Collection, and Analysis Techniques in Mixed-Method Studies. *Research in Nursing & Health*, *23*(3), 246–255. (Cited on pages 16 and 173)

[Van der Schuur *et al.* 2008] Van der Schuur, H., Jansen, S., and Brinkkemper, S. (2008). Becoming Responsive to Service Usage and Performance Changes by Applying Service Feedback Metrics to Software Maintenance. In *Evol '08: Proceedings of the 4th International ECRIM Workshop on Software Evolution and Evolvability*, (pp. 53–62). IEEE Computer Society Press. (Cited on pages 19, 25, 31, 48, 75, and 143)

[Van der Schuur *et al.* 2010] Van der Schuur, H., Jansen, S., and Brinkkemper, S. (2010). A Reference Framework for Utilization of Software Operation Knowledge. In *SEAA '10: Proceedings of the 36th EUROMICRO Conference on Software Engineering and Advanced Applications*, (pp. 245–254). IEEE Computer Society Press. (Cited on pages 18, 23, 46, 48, 72, 74, 98, 102, 107, and 123)

[Van der Schuur *et al.* 2011a] Van der Schuur, H., Jansen, S., and Brinkkemper, S. (2011). If the SOK Fits, Wear It: Pragmatic Process Improvement through Software Operation Knowledge. In *PROFES '11: Proceedings of the 12th International Conference on Product Focused Software Development and Process Improvement*, (pp. 306–321). Springer. (Cited on pages 19, 97, 121, and 141)

[Van der Schuur *et al.* 2011b] Van der Schuur, H., Jansen, S., and Brinkkemper, S. (2011). Reducing Maintenance Effort through Software Operation Knowledge: An Eclectic Empirical Evaluation. In *CSMR '11: Proceedings of the 15th European Conference on Software Maintenance and Reengineering*, (pp. 201–210). IEEE Computer Society Press. (Cited on pages 18, 48, 71, 98, 102, and 119)

[Van der Schuur *et al.* 2011c] Van der Schuur, H., Jansen, S., and Brinkkemper, S. (2011). Sending Out a Software Operation Summary: Leveraging Software Operation Knowledge for Prioritization of Maintenance Tasks. In *MENSURA '11: Proceedings of the 6th International Conference on Software Process and Product Measurement*. IEEE Computer Society Press. (Cited on pages 3, 19, 118, 119, and 120)

[Van der Schuur *et al.* 2011d] Van der Schuur, H., Jansen, S., and Brinkkemper, S. (2011). The Power of Propagation: On the Role of Software Operation Knowledge within Software Ecosystems. In *MEDES '11: Proceedings of the International Conference on Management of Emergent Digital EcoSystems*. ACM Press. (Cited on pages 18 and 45)

[Seffah *et al.* 2006] Seffah, A., Donyaee, M., Kline, R. B., and Padda, H. K. (2006). Usability measurement and metrics: A consolidated model. *Software Quality Journal*, *14*(2), 159–178. (Cited on page 148)

[SEI 2010] SEI (2010). CMMI® for Development — SCAMPI Class A Appraisal Results — 2010 End-Year Update. Software Engineering Institute. (Cited on page 8)

[Selby *et al.* 1991] Selby, R. W., Porter, A. A., Schmidt, D. C., and Berney, J. (1991). Metric-driven analysis and feedback systems for enabling empirically guided software development. In *ICSE '91: Proceedings of the 13th International Conference on Software Engineering*, (pp. 288–298). IEEE Computer Society Press. (Cited on pages 9 and 25)

[Shackel 1991] Shackel, B., Usability — context, framework, definition, design and evaluation, In *Human factors for informatics usability*, (pp. 21–37). (Cited on page 148)

[Silver *et al.* 1995] Silver, M. S., Markus, M. L., and Beath, C. M. (1995). The Information Technology Interaction Model: A Foundation for the MBA Core Course. *MIS Quarterly*, *19*, 361–390. (Cited on page 12)

[Simmons 2006] Simmons, E. (2006). The Usage Model: Describing Product Usage during Design and Development. *IEEE Software*, *23*(3), 34–41. (Cited on page 28)

[Smite and Gencel 2009] Smite, D. and Gencel, C., Why a CMMI Level 5 Company Fails to Meet the Deadlines?, In *Product-Focused Software Process Improvement*, volume 32 of *Lecture Notes in Business Information Processing*, (pp. 87–95). (Cited on pages 8, 98, and 172)

[Sommerville 2007] Sommerville, I. (2007). *Software Engineering*. International Computer Science. Addison-Wesley. (Cited on page 4)

[Stake 1995] Stake, R. E. (1995). *The Art of Case Study Research*. Sage. (Cited on page 15)

[Staples *et al.* 2007] Staples, M., Niazi, M., Jeffery, R., Abrahams, A., Byatt, P., and Murphy, R. (2007). An exploratory study of why organizations do not adopt CMMI. *Journal of Systems and Software*, *80*(6), 883–895. (Cited on pages 8 and 172)

[Stavely 1978] Stavely, A. M. (1978). Design feedback and its use in software design aid systems. *SIGSOFT Software Engineering Notes*, *3*, 72–78. (Cited on page 8)

[Van Steenbergen *et al.* 2010] Van Steenbergen, M., Bos, R., Brinkkemper, S., Van de Weerd, I., and Bekkers, W. (2010). The Design of Focus Area Maturity Models. In *DESRIST '10: Proceedings of the 5th International Conference on Design Science Research in Information Systems and Technology*, (pp. 317–332). Springer. (Cited on pages 4 and 8)

[Susman and Evered 1978] Susman, G. I. and Evered, R. D. (1978). An Assessment of the Scientific Merits of Action Research. *Administrative Science Quarterly*, *23*(4), 582–603. (Cited on page 101)

[Tautz and Althoff 1997] Tautz, C. and Althoff, K.-D. (1997). Using Case-Based Reasoning for Reusing Software Knowledge. *Case-Based Reasoning Research and Development*, 156–165. (Cited on page 25)

[The PostSharp Platform] The PostSharp Platform. `http://www.postsharp.org/`. (Cited on page 79)

[Tsichritzis 1998] Tsichritzis, D., The Dynamics of Innovation, In *Beyond Calculation: The Next Fifty Years of Computing*, (pp. 259–265). (Cited on page 12)

[Ubuntu Wiki 2010] (2010). Ubuntu Wiki – Apport. `https://wiki.ubuntu.com/Apport/`. (Cited on pages 120, 126, and 128)

[Vaishnavi and Kuechler 2009] Vaishnavi, V. and Kuechler, W. (2009). Design Research in Information Systems. `http://desrist.org/design-research-in-information-systems/`. (Cited on page 5)

[Verma 1999] Verma, D. (1999). *Supporting Service Level Agreements on IP Networks*. Macmillan Technical Publishing. (Cited on page 149)

[VS2008] Microsoft — .NET Framework Programming in Visual Studio. `http://msdn.microsoft.com/en-us/library/k1s94fta.aspx`. (Cited on page 157)

[Van de Weerd 2009] Van de Weerd, G. (2009). *Advancing in Software Product Management: An Incremental Method Engineering Approach*. PhD thesis, Utrecht University. (Cited on page 4)

[Van de Weerd *et al.* 2010] Van de Weerd, I., Bekkers, W., and Brinkkemper, S. (2010). Developing a Maturity Matrix for Software Product Management. In *ICSOB '10: Proceedings of the 1st International Conference on Software Business*, (pp. 76–89). (Cited on page 4)

[Van de Weerd *et al.* 2006] Van de Weerd, I., Brinkkemper, S., Nieuwenhuis, R., Versendaal, J., and Bijlsma, L. (2006). Towards a Reference Framework for Software Product Management. In *RE '06: Proceedings of the 14th IEEE International Requirements Engineering Conference*, (pp. 319–322). IEEE Computer Society Press. (Cited on page 4)

[Van de Weerd and Brinkkemper 2008] Van de Weerd, I. and Brinkkemper, S., Meta-Modeling for Situational Analysis and Design Methods, In Syed, M. R. and Syed, S. N. (Eds.), *Handbook of Research on Modern Systems Analysis and Design Technologies and Applications*, (pp. 38–58). (Cited on pages 102, 103, and 121)

[Wiegner and Nof 1993] Wiegner, R. and Nof, S. (1993). The software product feedback flow model for development planning. *Information and Software Technology*, *35*(8), 427–438. (Cited on page 9)

[WinQual] Windows Quality Online Services. `https://winqual.microsoft.com/`. (Cited on page 51)

[Xu and Brinkkemper 2007] Xu, L. and Brinkkemper, S. (2007). Concepts of Product Software. *European Journal of Information Systems*, *16*, 531–541. (Cited on pages 4, 5, 6, 7, 33, and 171)

[Yin 2009] Yin, R. K. (2009). *Case Study Research: Design and Methods (Applied Social Research Methods)* (Fourth Edition. ed.). Sage Publications. (Cited on pages 15, 34, 43, 136, and 153)

[Yu and Lin 2005] Yu, T. and Lin, K.-J. (2005). Service selection algorithms for Web services with end-to-end QoS constraints. *Inf. Systems and E-Business Management*, *3*, 103–126. (Cited on pages 27, 144, 147, and 152)

[Zeng *et al.* 2003] Zeng, L., Benatallah, B., Dumas, M., Kalagnanam, J., and Sheng, Q. Z. (2003). Quality Driven Web Services Composition. In *WWW '03: Proceedings of the 12th International Conference on World Wide Web*, (pp. 411–421). ACM Press. (Cited on pages 27, 147, and 148)

[Zimmermann *et al.* 2008] Zimmermann, T., Nagappan, N., and Zeller, A., Predicting Bugs from History, In Mens, T. and Demeyer, S. (Eds.), *Software Evolution*, (pp. 69–88). (Cited on page 9)

[Zimmermann *et al.* 2010] Zimmermann, T., Premraj, R., Bettenburg, N., Just, S., Schröter, A., and Weiss, C. (2010). What Makes a Good Bug Report? *IEEE Transactions on Software Engineering*, *36*(5), 618–643. (Cited on page 139)

A

Software Operation Knowledge Survey Questions

CONTEXTUAL INFORMATION

1. Please describe your function within the organization you're employed in.

2. Please give an indication of the size of the organization you're employed in.

3. Please characterize the organization you're employed in (type of products, services, branch, etc.).

4. Please indicate for how long your organization's main software product (or service) is available on the market now.

5. How many major releases have been released since the first commercial version of your organization's main software product (or service)?

IDENTIFICATION AND DEFINITION

6. Please indicate if you consider knowledge about performance of software in the field (e.g. latency, throughput, response times) as a part of the concept of 'Software Operation Knowledge'.

7. Please indicate if you consider knowledge about quality of software in the field (e.g. #exceptions, effectiveness, productivity, reliability, availability) as a part of the concept of 'Software Operation Knowledge'.

8. Please indicate if you consider knowledge about usage of software in the field (user interface paths, method calls and crash report frequency statistics) as a part of the concept of 'Software Operation Knowledge'.

9. Please indicate if you consider knowledge about feedback on software in the field by end-users (end-user feedback, customer satisfaction and feedback ratings) as a part of the concept of 'Software Operation Knowledge'.

10. Do you miss any types or forms of knowledge in the definition of the software operation knowledge concept given above? If so, detail the type(s) of knowledge you miss. Otherwise, explain why you think the concept definition is complete.

FRAMEWORK COMPONENTS

11. Do you miss a particular stakeholder besides the Software Vendor and the End-user(s)? If so, detail which stakeholders or stakeholders you miss and detail the role you think this stakeholder should have with respect to the concept of software operation knowledge. Otherwise, explain why you think the current two stakeholders are sufficient.

12. Do you miss a particular software operation knowledge phase? If so, detail which phase(s) you miss and why this phase should be positioned in the framework. Otherwise, explain why you think the current five phases are sufficient.

13. Do you miss a particular perspective from which software operation knowledge can be observed? If so, detail the perspective(s) you miss. Otherwise, explain why you think the current three perspectives are sufficient.

14. Do you miss any flows (possibly software operation knowledge flows, but not in particular) in the framework? If so, detail which (knowledge) flow(s) you miss and which components are sending and receiving which knowledge. Otherwise, explain why you think the current flows are sufficient.

15. Please describe any elements you would expect being a part of the framework, that are currently missing. If you don't miss any elements, please criticize the framework or write down general remarks.

IMPROVING ACTIVITIES THROUGH SOK SUPPORT

16. Please indicate which of the activities within your organization would be most supported by software performance knowledge. Motivate why.
17. Please indicate which of the activities within your organization would be most supported by software quality knowledge. Motivate why.
18. Please indicate which of the activities within your organization would be most supported by software usage knowledge. Motivate why.
19. Please indicate which of the activities within your organization would be most supported by end-user feedback knowledge. Motivate why.
20. Which (positive) effects are not mentioned but are, according to you, also a consequence of software operation knowledge utilization in your organization?
21. Please explain which factors (for instance time, knowledge, tools, customer demands, etc.) contributed positively or negatively to the current state of software operation knowledge utilization in your organization.

B

Software Operation Knowledge Integration Interview Questions

PROCESS IMPROVEMENT

1. Which processes have actually been improved by integration of acquired operation information?

2. How did these processes change after integration of operation information?

3. What were unforeseen positive side effects of this integration?

4. What were unforeseen negative side effects of this integration?

INTEGRATION OBJECTIVES

5. What were the objectives of integrating acquired operation information in <process> before doing so?

6. Have these objectives changed during the integration of acquired operation information?

7. What were the causes and effects (consequences) of this change?

8. Was it a welcome change according to you? Why?

INTEGRATION CHALLENGES

9. Which (technical) challenges did you encounter while integrating operation information in <process> and corresponding decisions?
10. Why did they encounter?
11. Could they have been prevented?
12. How were they addressed?
13. Do you think they can be prevented in the future? If so, how?

RETURN ON INVESTMENT

14. Considering the integration process in retrospect, would you consider the process as advantageous?
15. What would be the main Return On Investment? (time-, money-, knowledge investment, etc.)
16. How would you quantify this RoI? (percentage gain)
17. In retrospect, are the results of the integration process worth the effort you put into it? Why?
18. Did you notice any change in customer satisfaction during or after application of the method?
19. What was the cause of this change?

LESSONS LEARNED

20. What are the general lessons that you have learned from integrating acquired operation information in <process>?
21. Suppose that you have to integrate acquired operation information at another, equivalent software vendor. Given the knowledge you gained during this integration process, what would you do differently?
22. What would you advice to other, equivalent software vendors that are about to initiate integration of operation information in <process>?
23. What would you dissuade such vendors from doing?

24. How could such vendors optimize the process of SOK integration?
25. What are key conditions for successful SOK integration?

FUTURE

26. How does the future look like in terms of SOK integration?
27. Will there still be a need for SOK integration? Why?
28. What will be the next steps regarding SOK integration within your organization?
29. Who should be responsible for future integration of SOK in software processes?
30. Development manager or business manager? Internal or external?
31. What will be the major (technical) challenge for software vendors regarding the future of SOK integration?

FINAL REMARKS

32. Do you have any other remarks?
33. Do you want to express anything other regarding the performed SOK integration process?
34. Do you want to express anything other regarding the lessons learned?

C

Software Operation Knowledge Propagation Interview Questions

VENDOR IDENTIFICATION

1. Number of employees
2. Number of developers
3. Number of products / services
4. Name of main product / service (most licenses / users)
5. Number of releases main product / service
6. Number of development locations

ECOSYSTEM IDENTIFICATION

7. Software market
8. Main technology
9. What are your role(s) in the software ecosystem(s) you are participating in?
10. Which platforms are you making use of / providing software for?
11. Which platforms are you providing?
12. Nature of keystone-niche player relationship

13. Number of keystones
14. Number of niche players
15. Are there preferred relations with one or more keystones?
16. Are there preferred relations with certain niche players?
17. How are niche players allowed to participate in your software ecosystem?
18. How were you allowed to participate in your keystones' software ecosystems?

SOK PROPAGATION

19. How are software operation data / information extracted from in-the-field software operation / end-user behavior?
20. Which software operation data / information are considered relevant and valuable? Specific for niche players?
21. How is resulting software operation knowledge exerted? To which practices, processes or products?
22. What kind of SOK is propagated? To which actors?
23. What kind of SOK is received? From which actors?
24. Is there a strategy / protocol defined for SOK propagation?
25. Is there a strategy / protocol defined for processing received SOK?
26. Who are responsible for / involved with SOK propagation?
27. Who are responsible for / involved with SOK processing received SOK?

CHALLENGES

28. Which challenges are addressed by propagating SOK through your software ecosystem?
29. Are SECO control strategies and policies based on SOK? (perfect thrive strategy, make ecosystem more attractive)
30. What is the role of SOK propagation on relationships with other actors in your software ecosystem? Are relations intensified, prolonged or diminished? Are new relations established?

31. What is the role of SOK propagation on your insight of (the composition of) your software ecosystem?
32. To which ecosystem actor has the most SOK been propagated?
33. From which ecosystem actor has the most SOK been received?
34. What is the role of SOK propagation on the quality of your software?
35. What is the role of SOK propagation on portfolio and product line planning?
36. What is, in general, the most beneficial effect of SOK propagation for your organization?
37. What was the most valuable / effective SOK received until now? Why?
38. What was the most valuable / effective SOK propagated until now? Why?

FUTURE

39. How do you see the future in terms of SOK propagation?
40. How does the ideal future in terms of SOK propagation look like?
41. Which challenges, unveiled by SOK propagation, still have to be addressed?

List of Acronyms

AOP Aspect-Oriented Programming

API Application Programming Interface

CAD Computer-Aided Design

C-CCU Continuous Customer Configuration Updating

CMMI Capability Maturity Model Integration

COTS Commercial Off-The-Shelf

CRM Customer Relationship Management

ERP Enterprise Resource Planning

HRM Human Resource Management

SDK Software Development Kit

SECO Software Ecosystem

SKU Service Knowledge Utilization

SLA Service Level Agreement

SOK Software Operation Knowledge

SOS Software Operation Summary

SPM Software Product Management

SSN Software Supply Network

SWEBOK Software Engineering Body of Knowledge

UDDI Universal Description, Discovery and Integration

XML Extensible Markup Language

Publication List

In order of publication date

- Van der Schuur, H., Jansen, S., and Brinkkemper, S. (2008). Becoming Responsive to Service Usage and Performance Changes by Applying Service Feedback Metrics to Software Maintenance. In *Evol '08: Proceedings of the 4th International ECRIM Workshop on Software Evolution and Evolvability*, (pp. 53–62). IEEE Computer Society Press.

- Kristjánsson, B., and Van der Schuur, H. (2009). A Survey of Tools for Software Operation Knowledge Acquisition. Technical Report UU-CS-2009-028, Department of Information and Computing Sciences, Utrecht University.

- Van der Schuur, H., Jansen, S., and Brinkkemper, S. (2010). A Reference Framework for Utilization of Software Operation Knowledge. In *SEAA '10: Proceedings of the 36th EUROMICRO Conference on Software Engineering and Advanced Applications*, (pp. 245–254). IEEE Computer Society Press.

- Van der Schuur, H., Jansen, S., and Brinkkemper, S. (2011). Reducing Maintenance Effort through Software Operation Knowledge: An Eclectic Empirical Evaluation. In *CSMR '11: Proceedings of the 15th European Conference on Software Maintenance and Reengineering*, (pp. 201–210). IEEE Computer Society Press.

- Van der Schuur, H., Jansen, S., and Brinkkemper, S. (2011). If the SOK Fits, Wear It: Pragmatic Process Improvement through Software Operation Knowledge. In *PROFES '11: Proceedings of the 12th International Conference on Product Focused Software Development and Process Improvement*, (pp. 306–321). Springer.

- Van der Schuur, H., Jansen, S., and Brinkkemper, S. (2011). Sending Out a Software Operation Summary: Leveraging Software Operation Knowledge for Prioritization of Maintenance Tasks. In *MENSURA '11: Proceedings of the 6th International Conference on Software Process and Product Measurement*. IEEE Computer Society Press.

- Van der Schuur, H., Jansen, S., and Brinkkemper, S. (2011). The Power of Propagation: On the Role of Software Operation Knowledge within Software Ecosystems. In *MEDES '11: Proceedings of the International Conference on Management of Emergent Digital EcoSystems*. ACM Press.

Summary

Although the software industry is flourishing, and software-producing organizations strive for high levels of end-user satisfaction, these organizations do only limitedly recognize and use knowledge of the in-the-field operation of their software (e.g. software operation knowledge or SOK). Less than one-third of these organizations makes use of crash and usage feedback reports to acquire knowledge of the in-the-field behavior of their software and end-users. Furthermore, the concept of software operation knowledge has only been vaguely described, and is underexposed till date. For example, virtually no techniques or methods exist for structurally *using* such knowledge in processes implemented at software-producing organizations.

Consequently, an emerging need is observed for a framework that guides software-producing organizations in improving their software processes through knowledge of the in-the-field behavior of their software and end-users. In this dissertation research, a framework is presented that provides and structures directions for identification, acquisition, integration, presentation and utilization of software operation knowledge. The framework, as well as the tools, techniques and methods that are presented in this dissertation, aid software-producing organizations in increasing the efficiency of their software processes.

This dissertation is divided in four parts. After the introductory part, which describes research triggers, questions and methods, the SOK concept is defined. Also, the SOK framework is presented: a structure that describes parties, perspectives and life cycle processes (see figure) related to knowledge of in-the-field software operation. The SOK concept is then further established by identification and classification of operational software operation knowledge practices of software-producing organizations in software ecosystems. The third part of this dissertation is focused on process improvement through knowledge of in-the-field software operation. A novel technique for generic recording and visualization of in-the-field software operation is presented. We show that this technique enables software vendors to obtain a uniform insight in the operation of their software in the field, and contributes to reduction of software maintenance efforts. Also, we present a template method for situational integration of software operation information in software processes. We show that by using the template method, software-producing organizations can improve their particular software processes with acquired software operation information.

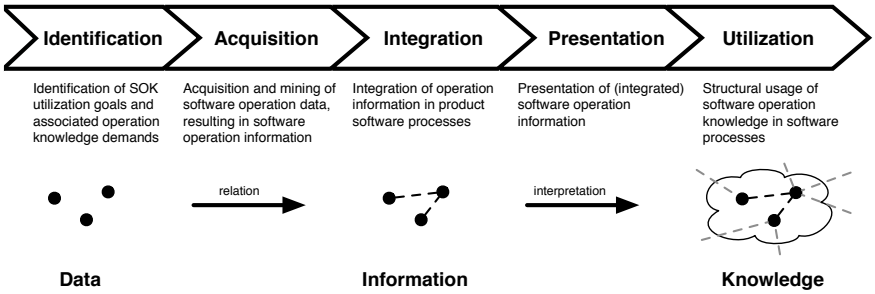


Figure Software operation knowledge life cycle

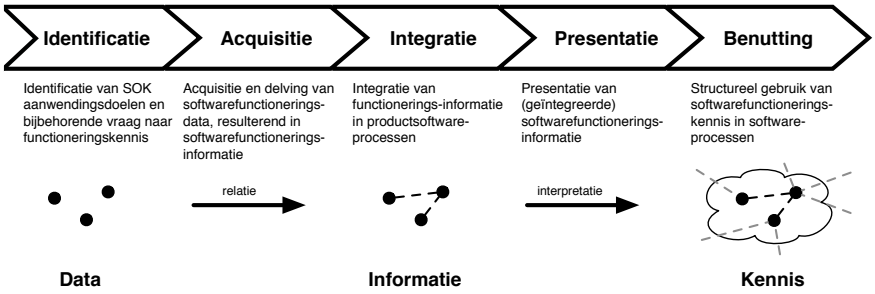
Another artifact that is presented in this dissertation is the software operation summary (SOS), a medium for presentation of software operation information. We show that the summary increases awareness of in-the-field software operation throughout software-producing organizations, and demonstrate that it contributes to reaching consensus on software maintenance task prioritization. Finally, service knowledge utilization (SKU) is introduced as an approach to increase a software vendor's flexibility, as well as its responsiveness to changes in performance and usage of its service-based, online software. The last part of this dissertation provides answers to the formulated research questions, as well as a reflection on the hermeneutic process we experienced throughout the construction of the artifacts that are presented in this dissertation.

Nederlandse samenvatting

Hoewel de softwareindustrie floreert, en softwarebedrijven streven naar hoge klanttevredenheid, zijn deze bedrijven slechts beperkt in staat om kennis van het in-het-veld functioneren van hun software (ofwel: softwarefunctioneringskennis of SOK) te herkennen en te gebruiken. Minder dan één derde van de softwarebedrijven maakt bijvoorbeeld gebruik van crash- en gebruiksfeedbackrapporten om kennis van het in-het-veld gedrag van hun software en eindgebruikers te verwerven. Daarnaast is het concept van softwarefunctioneringskennis tot nu toe slechts vaag beschreven en onderbelicht. Er bestaan bijvoorbeeld praktisch geen technieken of methoden voor het integreren van dergelijke kennis in de bedrijfsprocessen van softwarebedrijven.

Als gevolg hiervan wordt een opkomende vraag waargenomen naar een raamwerk dat softwarebedrijven begeleidt in het verbeteren van hun processen door middel van kennis van het in-het-veld gedrag van hun software en eindgebruikers. In dit proefschrift wordt een raamwerk gepresenteerd dat voorziet, en structuur brengt in richtlijnen voor identificatie, acquisitie, integratie, presentatie en benutting van softwarefunctioneringskennis. Het raamwerk, alsook de tools, technieken en methoden die in dit proefschrift worden gepresenteerd, helpen softwarebedrijven in het verhogen van de efficiëntie van hun bedrijfsprocessen.

Dit proefschrift is onderverdeeld in vier delen. Na het inleidende deel, dat onderzoeksaanleidingen, -vragen en -methoden beschrijft, wordt het SOK concept gedefinieerd. Ook wordt het SOK raamwerk gepresenteerd: een structuur die partijen, perspectieven en levenscyclusprocessen beschrijft die gerelateerd zijn aan kennis van in-het-veld softwarefunctioneren. Het concept van softwarefunctioneringskennis wordt vervolgens verder tot stand gebracht door operationele softwarefunctioneringskennispraktijken van softwarebedrijven in software-ecosystemen te identificeren en te classificeren. Het derde deel van dit proefschrift is gericht op procesverbetering door middel van kennis van in-het-veld softwarefunctioneren. Een nieuwe techniek voor generieke registratie en visualisatie van in-het-veld softwarefunctioneren wordt gepresenteerd. We laten zien dat deze techniek softwarebedrijven in staat stelt om een uniform inzicht in het functioneren van hun software in het veld te verkrijgen, en bijdraagt aan vermindering van softwareonderhoudsinspanningen. Ook presenteren we een sjabloonmethode voor situationele integratie van softwarefunctioneringskennis



Figuur Levenscyclus van softwarefunctioneringskennis

informatie in bedrijfsprocessen. We laten zien dat door de sjabloonmethode te gebruiken, softwarebedrijven hun specifieke bedrijfsprocessen kunnen verbeteren met verkregen softwarefunctioneringsinformatie. Een ander artefact dat in dit proefschrift wordt gepresenteerd is het softwarefunctioneringsoverzicht (SOS), een medium voor presentatie van softwarefunctioneringsinformatie. We laten zien dat dit overzicht het besef van in-het-veld softwarefunctioneren binnen softwarebedrijven verhoogt, en tonen aan dat het overzicht bijdraagt aan het bereiken van consensus over prioritering van softwareonderhoudstaken. Tenslotte wordt servicekennisbenutting (SKU) geïntroduceerd als een manier om de flexibiliteit van een softwarebedrijf, alsook haar responsiviteit op veranderingen in prestatie en gebruik van haar servicegebaseerde, online software, te verhogen. Het laatste deel van dit proefschrift biedt antwoorden op de geformuleerde onderzoeksvragen, alsook een reflectie op het hermeneutisch proces dat we ervaren hebben gedurende de constructie van de artefacten die in dit proefschrift worden gepresenteerd.

Curriculum Vitæ

Henk van der Schuur was born on March 29, 1986 in the Netherlands. From 2003 to 2006, he studied Computer Science at Utrecht University. He obtained his Master's degree in Business Informatics in September, 2008. His Master's thesis, entitled 'SKU: Software Vendor Meets End-user' was nominated for the Utrecht University Thesis Award. He started his PhD research as an external PhD researcher at Stabiplan B.V. In July, 2011, he became employed by AFAS ERP Software B.V. in the role of software product manager. He successfully finished his PhD research in November, 2011.

The research and educational activities of Henk van der Schuur focus on the areas of software feedback, software process improvement and software product management. In addition, he provides lectures, training and coaching in these areas.

SIKS PhD Theses

1998-1 Johan van den Akker (CWI)
DEGAS — An Active, Temporal Database of Autonomous Objects
1998-2 Floris Wiesman (UM)
Information Retrieval by Graphically Browsing Meta-Information
1998-3 Ans Steuten (TUD)
A Contribution to the Linguistic Analysis of
Business Conversations within the Language/Action Perspective
1998-4 Dennis Breuker (UM)
Memory versus Search in Games
1998-5 E.W.Oskamp (RUL)
Computerondersteuning bij Straftoemeting
1999-1 Mark Sloof (VU)
Physiology of Quality Change Modelling;
Automated modelling of Quality Change of Agricultural Products
1999-2 Rob Potharst (EUR)
Classification using decision trees and neural nets
1999-3 Don Beal (UM)
The Nature of Minimax Search
1999-4 Jacques Penders (UM)
The practical Art of Moving Physical Objects
1999-5 Aldo de Moor (KUB)
Empowering Communities: A Method for the Legitimate User-Driven
Specification of Network Information Systems
1999-6 Niek J.E. Wijngaards (VU)
Re-design of compositional systems
1999-7 David Spelt (UT)
Verification support for object database design
1999-8 Jacques H.J. Lenting (UM)
Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation
2000-1 Frank Niessink (VU)
Perspectives on Improving Software Maintenance
2000-2 Koen Holtman (TUE)
Prototyping of CMS Storage Management
2000-3 Carolien M.T. Metselaar (UvA)
Sociaal-organisatorische gevolgen van kennistechnologie; een procesbenadering en actorperspectief
2000-4 Geert de Haan (VU)
ETAG, A Formal Model of Competence Knowledge for User Interface Design
2000-5 Ruud van der Pol (UM)
Knowledge-based Query Formulation in Information Retrieval
2000-6 Rogier van Eijk (UU)
Programming Languages for Agent Communication
2000-7 Niels Peek (UU)
Decision-theoretic Planning of Clinical Patient Management
2000-8 Veerle Coup (EUR)
Sensitivity Analyis of Decision-Theoretic Networks
2000-9 Florian Waas (CWI)
Principles of Probabilistic Query Optimization
2000-10 Niels Nes (CWI)
Image Database Management System Design Considerations, Algorithms and Architecture
2000-11 Jonas Karlsson (CWI)
Scalable Distributed Data Structures for Database Management

2001-1 Silja Renooij (UU)
Qualitative Approaches to Quantifying Probabilistic Networks
2001-2 Koen Hindriks (UU)
Agent Programming Languages: Programming with Mental Models
2001-3 Maarten van Someren (UvA)
Learning as problem solving
2001-4 Evgueni Smirnov (UM)
Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets
2001-5 Jacco van Ossenbruggen (VU)
Processing Structured Hypermedia: A Matter of Style
2001-6 Martijn van Welie (VU)
Task-based User Interface Design
2001-7 Bastiaan Schonhage (VU)
Diva: Architectural Perspectives on Information Visualization
2001-8 Pascal van Eck (VU)
A Compositional Semantic Structure for Multi-Agent Systems Dynamics
2001-9 Pieter Jan 't Hoen (RUL)
Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes
2001-10 Maarten Sierhuis (UvA)
Modeling and Simulating Work Practice —
BRAHMS: a multiagent modeling and simulation language for work practice analysis and design
2001-11 Tom M. van Engers (VU)
Knowledge Management: The Role of Mental Models in Business Systems Design
2002-01 Nico Lassing (VU)
Architecture-Level Modifiability Analysis
2002-02 Roelof van Zwol (UT)
Modelling and searching web-based document collections
2002-03 Henk Ernst Blok (UT)
Database Optimization Aspects for Information Retrieval
2002-04 Juan Roberto Castelo Valdueza (UU)
The Discrete Acyclic Digraph Markov Model in Data Mining
2002-05 Radu Serban (VU)
The Private Cyberspace Modeling Electronic Environments inhabited by Privacy-concerned Agents
2002-06 Laurens Mommers (UL)
Applied legal epistemology; Building a knowledge-based ontology of the legal domain
2002-07 Peter Boncz (CWI)
Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications
2002-08 Jaap Gordijn (VU)
Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas
2002-09 Willem-Jan van den Heuvel (KUB)
Integrating Modern Business Applications with Objectified Legacy Systems
2002-10 Brian Sheppard (UM)
Towards Perfect Play of Scrabble
2002-11 Wouter C.A. Wijngaards (VU)
Agent Based Modelling of Dynamics: Biological and Organisational Applications
2002-12 Albrecht Schmidt (UvA)
Processing XML in Database Systems
2002-13 Hongjing Wu (TUE)
A Reference Architecture for Adaptive Hypermedia Applications
2002-14 Wieke de Vries (UU)
Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems
2002-15 Rik Eshuis (UT)
Semantics and Verification of UML Activity Diagrams for Workflow Modelling
2002-16 Pieter van Langen (VU)
The Anatomy of Design: Foundations, Models and Applications
2002-17 Stefan Manegold (UvA)
Understanding, Modeling, and Improving Main-Memory Database Performance

2003-01 Heiner Stuckenschmidt (VU)
Ontology-Based Information Sharing in Weakly Structured Environments
2003-02 Jan Broersen (VU)
Modal Action Logics for Reasoning About Reactive Systems
2003-03 Martijn Schuemie (TUD)
Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy
2003-04 Milan Petkovic (UT)
Content-Based Video Retrieval Supported by Database Technology
2003-05 Jos Lehmann (UvA)
Causation in Artificial Intelligence and Law — A modelling approach
2003-06 Boris van Schooten (UT)
Development and specification of virtual environments
2003-07 Machiel Jansen (UvA)
Formal Explorations of Knowledge Intensive Tasks
2003-08 Yongping Ran (UM)
Repair Based Scheduling
2003-09 Rens Kortmann (UM)
The resolution of visually guided behaviour
2003-10 Andreas Lincke (UvT)
Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture
2003-11 Simon Keizer (UT)
Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks
2003-12 Roeland Ordelman (UT)
Dutch speech recognition in multimedia information retrieval
2003-13 Jeroen Donkers (UM)
Nosce Hostem — Searching with Opponent Models
2003-14 Stijn Hoppenbrouwers (KUN)
Freezing Language: Conceptualisation Processes across ICT-Supported Organisations
2003-15 Mathijs de Weerdt (TUD)
Plan Merging in Multi-Agent Systems
2003-16 Menzo Windhouwer (CWI)
Feature Grammar Systems — Incremental Maintenance of Indexes to Digital Media Warehouses
2003-17 David Jansen (UT)
Extensions of Statecharts with Probability, Time, and Stochastic Timing
2003-18 Levente Kocsis (UM)
Learning Search Decisions
2004-01 Virginia Dignum (UU)
A Model for Organizational Interaction: Based on Agents, Founded in Logic
2004-02 Lai Xu (UvT)
Monitoring Multi-party Contracts for E-business
2004-03 Perry Groot (VU)
A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving
2004-04 Chris van Aart (UvA)
Organizational Principles for Multi-Agent Architectures
2004-05 Viara Popova (EUR)
Knowledge discovery and monotonicity
2004-06 Bart-Jan Hommes (TUD)
The Evaluation of Business Process Modeling Techniques
2004-07 Elise Boltjes (UM)
Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes
2004-08 Joop Verbeek (UM)
Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politiële gegevensuitwisseling en digitale expertise
2004-09 Martin Caminada (VU)
For the Sake of the Argument; explorations into argument-based reasoning

2004-10 Suzanne Kabel (UvA)
Knowledge-rich indexing of learning-objects
2004-11 Michel Klein (VU)
Change Management for Distributed Ontologies
2004-12 The Duy Bui (UT)
Creating emotions and facial expressions for embodied agents
2004-13 Wojciech Jamroga (UT)
Using Multiple Models of Reality: On Agents who Know how to Play
2004-14 Paul Harrenstein (UU)
Logic in Conflict. Logical Explorations in Strategic Equilibrium
2004-15 Arno Knobbe (UU)
Multi-Relational Data Mining
2004-16 Federico Divina (VU)
Hybrid Genetic Relational Search for Inductive Learning
2004-17 Mark Winands (UM)
Informed Search in Complex Games
2004-18 Vania Bessa Machado (UvA)
Supporting the Construction of Qualitative Knowledge Models
2004-19 Thijs Westerveld (UT)
Using generative probabilistic models for multimedia retrieval
2004-20 Madelon Evers (Nyenrode)
Learning from Design: facilitating multidisciplinary design teams
2005-01 Floor Verdenius (UvA)
Methodological Aspects of Designing Induction-Based Applications
2005-02 Erik van der Werf (UM))
AI techniques for the game of Go
2005-03 Franc Grootjen (RUN)
A Pragmatic Approach to the Conceptualisation of Language
2005-04 Nirvana Meratnia (UT)
Towards Database Support for Moving Object data
2005-05 Gabriel Infante-Lopez (UvA)
Two-Level Probabilistic Grammars for Natural Language Parsing
2005-06 Pieter Spronck (UM)
Adaptive Game AI
2005-07 Flavius Frasincar (TUE)
Hypermedia Presentation Generation for Semantic Web Information Systems
2005-08 Richard Vdovjak (TUE)
A Model-driven Approach for Building Distributed Ontology-based Web Applications
2005-09 Jeen Broekstra (VU)
Storage, Querying and Inferencing for Semantic Web Languages
2005-10 Anders Bouwer (UvA)
Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments
2005-11 Elth Ogston (VU)
Agent Based Matchmaking and Clustering — A Decentralized Approach to Search
2005-12 Csaba Boer (EUR)
Distributed Simulation in Industry
2005-13 Fred Hamburg (UL)
Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen
2005-14 Borys Omelayenko (VU)
Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics
2005-15 Tibor Bosse (VU)
Analysis of the Dynamics of Cognitive Processes
2005-16 Joris Graaumans (UU)
Usability of XML Query Languages
2005-17 Boris Shishkov (TUD)
Software Specification Based on Re-usable Business Components
2005-18 Danielle Sent (UU)
Test-selection strategies for probabilistic networks

2005-19 Michel van Dartel (UM)
Situated Representation
2005-20 Cristina Coteanu (UL)
Cyber Consumer Law, State of the Art and Perspectives
2005-21 Wijnand Derks (UT)
Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics
2006-01 Samuil Angelov (TUE)
Foundations of B2B Electronic Contracting
2006-02 Cristina Chisalita (VU)
Contextual issues in the design and use of information technology in organizations
2006-03 Noor Christoph (UvA)
The role of metacognitive skills in learning to solve problems
2006-04 Marta Sabou (VU)
Building Web Service Ontologies
2006-05 Cees Pierik (UU)
Validation Techniques for Object-Oriented Proof Outlines
2006-06 Ziv Baida (VU)
Software-aided Service Bundling — Intelligent Methods & Tools
for Graphical Service Modeling
2006-07 Marko Smiljanic (UT)
XML schema matching — balancing efficiency and effectiveness by means of clustering
2006-08 Eelco Herder (UT)
Forward, Back and Home Again — Analyzing User Behavior on the Web
2006-09 Mohamed Wahdan (UM)
Automatic Formulation of the Auditor's Opinion
2006-10 Ronny Siebes (VU)
Semantic Routing in Peer-to-Peer Systems
2006-11 Joeri van Ruth (UT)
Flattening Queries over Nested Data Types
2006-12 Bert Bongers (VU)
Interactivation — Towards an e-cology of people, our technological environment, and the arts
2006-13 Henk-Jan Lebbink (UU)
Dialogue and Decision Games for Information Exchanging Agents
2006-14 Johan Hoorn (VU)
Software Requirements: Update, Upgrade, Redesign — towards a Theory of Requirements Change
2006-15 Rainer Malik (UU)
CONAN: Text Mining in the Biomedical Domain
2006-16 Carsten Riggelsen (UU)
Approximation Methods for Efficient Learning of Bayesian Networks
2006-17 Stacey Nagata (UU)
User Assistance for Multitasking with Interruptions on a Mobile Device
2006-18 Valentin Zhizhkun (UvA)
Graph transformation for Natural Language Processing
2006-19 Birna van Riemsdijk (UU)
Cognitive Agent Programming: A Semantic Approach
2006-20 Marina Velikova (UvT)
Monotone models for prediction in data mining
2006-21 Bas van Gils (RUN)
Aptness on the Web
2006-22 Paul de Vrieze (RUN)
Fundaments of Adaptive Personalisation
2006-23 Ion Juvina (UU)
Development of Cognitive Model for Navigating on the Web
2006-24 Laura Hollink (VU)
Semantic Annotation for Retrieval of Visual Resources
2006-25 Madalina Drugan (UU)
Conditional log-likelihood MDL and Evolutionary MCMC

2006-26 Vojkan Mihajloviç (UT)
Score Region Algebra: A Flexible Framework for Structured Information Retrieval
2006-27 Stefano Bocconi (CWI)
Vox Populi: generating video documentaries from semantically annotated media repositories
2006-28 Borkur Sigurbjornsson (UvA)
Focused Information Access using XML Element Retrieval
2007-01 Kees Leune (UvT)
Access Control and Service-Oriented Architectures
2007-02 Wouter Teepe (RUG)
Reconciling Information Exchange and Confidentiality: A Formal Approach
2007-03 Peter Mika (VU)
Social Networks and the Semantic Web
2007-04 Jurriaan van Diggelen (UU)
Achieving Semantic Interoperability in Multi-agent Systems: a dialogue-based approach
2007-05 Bart Schermer (UL)
Software Agents, Surveillance, and the Right to Privacy:
a Legislative Framework for Agent-enabled Surveillance
2007-06 Gilad Mishne (UvA)
Applied Text Analytics for Blogs
2007-07 Natasa Jovanoviç (UT)
To Whom It May Concern — Addressee Identification in Face-to-Face Meetings
2007-08 Mark Hoogendoorn (VU)
Modeling of Change in Multi-Agent Organizations
2007-09 David Mobach (VU)
Agent-Based Mediated Service Negotiation
2007-10 Huib Aldewereld (UU)
Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols
2007-11 Natalia Stash (TUE)
Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System
2007-12 Marcel van Gerven (RUN)
Bayesian Networks for Clinical Decision Support:
A Rational Approach to Dynamic Decision-Making under Uncertainty
2007-13 Rutger Rienks (UT)
Meetings in Smart Environments; Implications of Progressing Technology
2007-14 Niek Bergboer (UM)
Context-Based Image Analysis
2007-15 Joyca Lacroix (UM)
NIM: a Situated Computational Memory Model
2007-16 Davide Grossi (UU)
Designing Invisible Handcuffs.
Formal Investigations in Institutions and Organizations for Multi-agent Systems
2007-17 Theodore Charitos (UU)
Reasoning with Dynamic Networks in Practice
2007-18 Bart Orriens (UvT)
On the development an management of adaptive business collaborations
2007-19 David Levy (UM)
Intimate relationships with artificial partners
2007-20 Slinger Jansen (UU)
Customer Configuration Updating in a Software Supply Network
2007-21 Karianne Vermaas (UU)
Fast diffusion and broadening use: A research on residential adoption and usage of
broadband internet in the Netherlands between 2001 and 2005
2007-22 Zlatko Zlatev (UT)
Goal-oriented design of value and process models from patterns
2007-23 Peter Barna (TUE)
Specification of Application Logic in Web Information Systems
2007-24 Georgina RamÃŋrez Camps (CWI)
Structural Features in XML Retrieval

2007-25 Joost Schalken (VU)
Empirical Investigations in Software Process Improvement
2008-01 Katalin Boer-SorbÃąn (EUR)
Agent-Based Simulation of Financial Markets: A modular, continuous-time approach
2008-02 Alexei Sharpanskykh (VU)
On Computer-Aided Methods for Modeling and Analysis of Organizations
2008-03 Vera Hollink (UvA)
Optimizing hierarchical menus: a usage-based approach
2008-04 Ander de Keijzer (UT)
Management of Uncertain Data — towards unattended integration
2008-05 Bela Mutschler (UT)
Modeling and simulating causal dependencies on process-aware information systems from a cost perspective
2008-06 Arjen Hommersom (RUN)
On the Application of Formal Methods to Clinical Guidelines, an Artificial Intelligence Perspective
2008-07 Peter van Rosmalen (OU)
Supporting the tutor in the design and support of adaptive e-learning
2008-08 Janneke Bolt (UU)
Bayesian Networks: Aspects of Approximate Inference
2008-09 Christof van Nimwegen (UU)
The paradox of the guided user: assistance can be counter-effective
2008-10 Wauter Bosma (UT)
Discourse oriented summarization
2008-11 Vera Kartseva (VU)
Designing Controls for Network Organizations: A Value-Based Approach
2008-12 Jozsef Farkas (RUN)
A Semiotically Oriented Cognitive Model of Knowledge Representation
2008-13 Caterina Carraciolo (UvA)
Topic Driven Access to Scientific Handbooks
2008-14 Arthur van Bunningen (UT)
Context-Aware Querying; Better Answers with Less Effort
2008-15 Martijn van Otterlo (UT)
The Logic of Adaptive Behavior: Knowledge Representation and Algorithms
for the Markov Decision Process Framework in First-Order Domains
2008-16 Henriette van Vugt (VU)
Embodied agents from a user's perspective
2008-17 Martin Op 't Land (TUD)
Applying Architecture and Ontology to the Splitting and Allying of Enterprises
2008-18 Guido de Croon (UM)
Adaptive Active Vision
2008-19 Henning Rode (UT)
From Document to Entity Retrieval: Improving Precision and Performance of Focused Text Search
2008-20 Rex Arendsen (UvA)
Geen bericht, goed bericht. Een onderzoek naar de effecten van de introductie van
elektronisch berichtenverkeer met de overheid op de administratieve lasten van bedrijven
2008-21 Krisztian Balog (UvA)
People Search in the Enterprise
2008-22 Henk Koning (UU)
Communication of IT-Architecture
2008-23 Stefan Visscher (UU)
Bayesian network models for the management of ventilator-associated pneumonia
2008-24 Zharko Aleksovski (VU)
Using background knowledge in ontology matching
2008-25 Geert Jonker (UU)
Efficient and Equitable Exchange in Air Traffic Management Plan Repair using Spender-signed Currency
2008-26 Marijn Huijbregts (UT)
Segmentation, Diarization and Speech Transcription: Surprise Data Unraveled
2008-27 Hubert Vogten (OU)
Design and Implementation Strategies for IMS Learning Design

2008-28 Ildiko Flesch (RUN)
On the Use of Independence Relations in Bayesian Networks
2008-29 Dennis Reidsma (UT)
Annotations and Subjective Machines — Of Annotators, Embodied Agents, Users, and Other Humans
2008-30 Wouter van Atteveldt (VU)
Semantic Network Analysis: Techniques for Extracting, Representing and Querying Media Content
2008-31 Loes Braun (UM)
Pro-Active Medical Information Retrieval
2008-32 Trung H. Bui (UT)
Toward Affective Dialogue Management using Partially Observable Markov Decision Processes
2008-33 Frank Terpstra (UvA)
Scientific Workflow Design; theoretical and practical issues
2008-34 Jeroen de Knijf (UU)
Studies in Frequent Tree Mining
2008-35 Ben Torben Nielsen (UvT)
Dendritic morphologies: function shapes structure
2009-01 Rasa Jurgelenaite (RUN)
Symmetric Causal Independence Models
2009-02 Willem Robert van Hage (VU)
Evaluating Ontology-Alignment Techniques
2009-03 Hans Stol (UvT)
A Framework for Evidence-based Policy Making Using IT
2009-04 Josephine Nabukenya (RUN)
Improving the Quality of Organisational Policy Making using Collaboration Engineering
2009-05 Sietse Overbeek (RUN)
Bridging Supply and Demand for Knowledge Intensive Tasks —
Based on Knowledge, Cognition, and Quality
2009-06 Muhammad Subianto (UU)
Understanding Classification
2009-07 Ronald Poppe (UT)
Discriminative Vision-Based Recovery and Recognition of Human Motion
2009-08 Volker Nannen (VU)
Evolutionary Agent-Based Policy Analysis in Dynamic Environments
2009-09 Benjamin Kanagwa (RUN)
Design, Discovery and Construction of Service-oriented Systems
2009-10 Jan Wielemaker (UvA)
Logic programming for knowledge-intensive interactive applications
2009-11 Alexander Boer (UvA)
Legal Theory, Sources of Law & the Semantic Web
2009-12 Peter Massuthe (TUE, Humboldt-Universitaet zu Berlin)
perating Guidelines for Services
2009-13 Steven de Jong (UM)
Fairness in Multi-Agent Systems
2009-14 Maksym Korotkiy (VU)
From ontology-enabled services to service-enabled ontologies
(making ontologies work in e-science with ONTO-SOA)
2009-15 Rinke Hoekstra (UvA)
Ontology Representation — Design Patterns and Ontologies that Make Sense
2009-16 Fritz Reul (UvT)
New Architectures in Computer Chess
2009-17 Laurens van der Maaten (UvT)
Feature Extraction from Visual Data
2009-18 Fabian Groffen (CWI)
Armada, An Evolving Database System
2009-19 Valentin Robu (CWI)
Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets
2009-20 Bob van der Vecht (UU)
Adjustable Autonomy: Controling Influences on Decision Making

2009-21 Stijn Vanderlooy (UM)
Ranking and Reliable Classification
2009-22 Pavel Serdyukov (UT)
Search For Expertise: Going beyond direct evidence
2009-23 Peter Hofgesang (VU)
Modelling Web Usage in a Changing Environment
2009-24 Annerieke Heuvelink (VU)
Cognitive Models for Training Simulations
2009-25 Alex van Ballegooij (CWI)
RAM: Array Database Management through Relational Mapping
2009-26 Fernando Koch (UU)
An Agent-Based Model for the Development of Intelligent Mobile Services
2009-27 Christian Glahn (OU)
Contextual Support of social Engagement and Reflection on the Web
2009-28 Sander Evers (UT)
Sensor Data Management with Probabilistic Models
2009-29 Stanislav Pokraev (UT)
Model-Driven Semantic Integration of Service-Oriented Applications
2009-30 Marcin Zukowski (CWI)
Balancing vectorized query execution with bandwidth-optimized storage
2009-31 Sofiya Katrenko (UvA)
A Closer Look at Learning Relations from Text
2009-32 Rik Farenhorst (VU) and Remco de Boer (VU)
Architectural Knowledge Management: Supporting Architects and Auditors
2009-33 Khiet Truong (UT)
How Does Real Affect Affect Affect Recognition In Speech?
2009-34 Inge van de Weerd (UU)
Advancing in Software Product Management: An Incremental Method Engineering Approach
2009-35 Wouter Koelewijn (UL)
Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling
2009-36 Marco Kalz (OUN)
Placement Support for Learners in Learning Networks
2009-37 Hendrik Drachsler (OUN)
Navigation Support for Learners in Informal Learning Networks
2009-38 Riina Vuorikari (OU)
Tags and self-organisation: a metadata ecology for learning resources in a multilingual context
2009-39 Christian Stahl (TUE, Humboldt-Universitaet zu Berlin)
Service Substitution — A Behavioral Approach Based on Petri Nets
2009-40 Stephan Raaijmakers (UvT)
Multinomial Language Learning: Investigations into the Geometry of Language
2009-41 Igor Berezhnyy (UvT)
Digital Analysis of Paintings
2009-42 Toine Bogers
Recommender Systems for Social Bookmarking
2009-43 Virginia Nunes Leal Franqueira (UT)
Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients
2009-44 Roberto Santana Tapia (UT)
Assessing Business-IT Alignment in Networked Organizations
2009-45 Jilles Vreeken (UU)
Making Pattern Mining Useful
2009-46 Loredana Afanasiev (UvA)
Querying XML: Benchmarks and Recursion
2010-01 Matthijs van Leeuwen (UU)
Patterns that Matter
2010-02 Ingo Wassink (UT)
Work flows in Life Science
2010-03 Joost Geurts (CWI)
A Document Engineering Model and Processing Framework for Multimedia documents

2010-04 Olga Kulyk (UT)
Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments
2010-05 Claudia Hauff (UT)
Predicting the Effectiveness of Queries and Retrieval Systems
2010-06 Sander Bakkes (UvT)
Rapid Adaptation of Video Game AI
2010-07 Wim Fikkert (UT)
Gesture interaction at a Distance
2010-08 Krzysztof Siewicz (UL)
Towards an Improved Regulatory Framework of Free Software.
Protecting user freedoms in a world of software communities and eGovernments
2010-09 Hugo Kielman (UL)
A Politiele gegevensverwerking en Privacy, Naar een effectieve waarborging
2010-10 Rebecca Ong (UL)
Mobile Communication and Protection of Children
2010-11 Adriaan Ter Mors (TUD)
The world according to MARP: Multi-Agent Route Planning
2010-12 Susan van den Braak (UU)
Sensemaking software for crime analysis
2010-13 Gianluigi Folino (RUN)
High Performance Data Mining using Bio-inspired techniques
2010-14 Sander van Splunter (VU)
Automated Web Service Reconfiguration
2010-15 Lianne Bodenstaff (UT)
Managing Dependency Relations in Inter-Organizational Models
2010-16 Sicco Verwer (TUD)
Efficient Identification of Timed Automata, theory and practice
2010-17 Spyros Kotoulas (VU)
Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications
2010-18 Charlotte Gerritsen (VU)
Caught in the Act: Investigating Crime by Agent-Based Simulation
2010-19 Henriette Cramer (UvA)
People's Responses to Autonomous and Adaptive Systems
2010-20 Ivo Swartjes (UT)
Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative
2010-21 Harold van Heerde (UT)
Privacy-aware data management by means of data degradation
2010-22 Michiel Hildebrand (CWI)
End-user Support for Access to Heterogeneous Linked Data
2010-23 Bas Steunebrink (UU)
The Logical Structure of Emotions
2010-24 Dmytro Tykhonov
Designing Generic and Efficient Negotiation Strategies
2010-25 Zulfiqar Ali Memon (VU)
Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective
2010-26 Ying Zhang (CWI)
XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines
2010-27 Marten Voulon (UL)
Automatisch contracteren
2010-28 Arne Koopman (UU)
Characteristic Relational Patterns
2010-29 Stratos Idreos (CWI)
Database Cracking: Towards Auto-tuning Database Kernels
2010-30 Marieke van Erp (UvT)
Accessing Natural History — Discoveries in data cleaning, structuring, and retrieval
2010-31 Victor de Boer (UvA)
Ontology Enrichment from Heterogeneous Sources on the Web

2010-32 Marcel Hiel (UvT)
An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems
2010-33 Robin Aly (UT)
Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval
2010-34 Teduh Dirgahayu (UT)
Interaction Design in Service Compositions
2010-35 Dolf Trieschnigg (UT)
Proof of Concept: Concept-based Biomedical Information Retrieval
2010-36 Jose Janssen (OU)
Paving the Way for Lifelong Learning;
Facilitating competence development through a learning path specification
2010-37 Niels Lohmann (TUE)
Correctness of services and their composition
2010-38 Dirk Fahland (TUE)
From Scenarios to components
2010-39 Ghazanfar Farooq Siddiqui (VU)
Integrative modeling of emotions in virtual agents
2010-40 Mark van Assem (VU)
Converting and Integrating Vocabularies for the Semantic Web
2010-41 Guillaume Chaslot (UM)
Monte-Carlo Tree Search
2010-42 Sybren de Kinderen (VU)
Needs-driven service bundling in a multi-supplier setting — the computational e3-service approach
2010-43 Peter van Kranenburg (UU)
A Computational Approach to Content-Based Retrieval of Folk Song Melodies
2010-44 Pieter Bellekens (TUE)
An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources,
Illustrated in the Television Domain
2010-45 Vasilios Andrikopoulos (UvT)
A theory and model for the evolution of software services
2010-46 Vincent Pijpers (VU)
e3alignment: Exploring Inter-Organizational Business-ICT Alignment
2010-47 Chen Li (UT)
Mining Process Model Variants: Challenges, Techniques, Examples
2010-48 Milan Lovric (EUR)
Behavioral Finance and Agent-Based Artificial Markets
2010-49 Jahn-Takeshi Saito (UM)
Solving difficult game positions
2010-50 Bouke Huurnink (UvA)
Search in Audiovisual Broadcast Archives
2010-51 Alia Khairia Amin (CWI)
Understanding and supporting information seeking tasks in multiple sources
2010-52 Peter-Paul van Maanen (VU)
Adaptive Support for Human-Computer Teams
Exploring the Use of Cognitive Models of Trust and Attention
2010-53 Edgar Meij (UvA)
Combining Concepts and Language Models for Information Access
2011-01 Botond Cseke (RUN)
Variational Algorithms for Bayesian Inference in Latent Gaussian Models
2011-02 Nick Tinnemeier (UU)
Work flows in Life Science
2011-03 Jan Martijn van der Werf (TUE)
Compositional Design and Verification of Component-Based Information Systems
2011-04 Hado van Hasselt (UU)
Insights in Reinforcement Learning;
Formal analysis and empirical evaluation of temporal-difference learning algorithms
2011-05 Base van der Raadt (VU)
Enterprise Architecture Coming of Age — Increasing the Performance of an Emerging Discipline

2011-06 Yiwen Wang (TUE)
Semantically-Enhanced Recommendations in Cultural Heritage
2011-07 Yujia Cao (UT)
Multimodal Information Presentation for High Load Human Computer Interaction
2011-08 Nieske Vergunst (UU)
BDI-based Generation of Robust Task-Oriented Dialogues **2011-09** Tim de Jong (OU)
Contextualised Mobile Media for Learning
2011-10 Bart Bogaert (UvT)
Cloud Content Contention
2011-11 Dhaval Vyas (UT)
Designing for Awareness: An Experience-focused HCI Perspective
2011-12 Carmen Bratosin (TUE)
Grid Architecture for Distributed Process Mining
2011-13 Xiaoyu Mao (UvT)
Airport under Control. Multiagent Scheduling for Airport Ground Handling
2011-14 Milan Lovric (EUR)
Behavioral Finance and Agent-Based Artificial Markets
2011-15 Marijn Koolen (UvA)
The Meaning of Structure: the Value of Link Evidence for Information Retrieval
2011-16 Maarten Schadd (UM)
Selective Search in Games of Different Complexity
2011-17 Jiyin He (UvA)
Exploring Topic Structure: Coherence, Diversity and Relatedness
2011-18 Mark Ponsen (UM)
Strategic Decision-Making in Complex Games
2011-19 Ellen Rusman (OU)
The Mind's Eye on Personal Profiles
2011-20 Qing Gu (VU)
Guiding service-oriented software engineering — A view-based approach
2011-21 Linda Terlouw (TUD)
Modularization and Specification of Service-Oriented Systems
2011-22 Junte Zhang (UvA)
System Evaluation of Archival Description and Access
2011-23 Wouter Weerkamp (UvA)
Finding People and their Utterances in Social Media
2011-24 Herwin van Welbergen (UT)
Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying,
Scheduling and Realizing Multimodal Virtual Human Behavior
2011-25 Syed Waqar ul Qounain Jaffry (VU)
Analysis and Validation of Models for Trust Dynamics
2011-26 Matthijs Aart Pontier (VU)
Virtual Agents for Human Communication —
Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots
2011-27 Aniel Bhulai (VU)
Dynamic website optimization through autonomous management of design patterns
2011-28 Rianne Kaptein (UvA)
Effective Focused Retrieval by Exploiting Query Context and Document Structure
2011-29 Faisal Kamiran (TUE)
Discrimination-aware Classification
2011-30 Egon van den Broek (UT)
Affective Signal Processing (ASP): Unraveling the mystery of emotions
2011-31 Ludo Waltman (EUR)
Computational and Game-Theoretic Approaches for Modeling Bounded Rationality
2011-32 Nees-Jan van Eck (EUR)
Methodological Advances in Bibliometric Mapping of Science
2011-33 Tom van der Weide (UU)
Arguing to Motivate Decisions

2011-34 Paolo Turrini (UU)
Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations
2011-35 Maaike Harbers (UU)
Explaining Agent Behavior in Virtual Training
2011-36 Erik van der Spek (UU)
Experiments in serious game design: a cognitive approach
2011-37 Adriana Birlutiu (RUN)
Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference
2011-38 Nyree Lemmens (UM)
Bee-inspired Distributed Optimization
2011-39 Joost Westra (UU)
Organizing Adaptation using Agents in Serious Games
2011-40 Viktor Clerc (VU)
Architectural Knowledge Management in Global Software Development
2011-41 Luan Ibraimi (UT)
Cryptographically Enforced Distributed Data Access Control
2011-42 Michal Sindlar (UU)
In the Eye of the Beholder: Explaining Behavior through Mental State Attribution
2011-43 Henk van der Schuur (UU)
Process Improvement through Software Operation Knowledge