2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications

A Reference Framework for Utilization of Software Operation Knowledge

Henk van der Schuur, Slinger Jansen, Sjaak Brinkkemper
Department of Information and Computing Sciences
Utrecht University
Utrecht, The Netherlands
{*h.schuur, s.jansen, s.brinkkemper*}*@cs.uu.nl*

***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.**

***Keywords*-software performance, software quality, software usage, software feedback, software process improvement**

I. 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 [1]. 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’ [2] 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[1], 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[2] 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 [3] focuses on user community-supported quality assurance of software operating in large configuration spaces. Furthermore, the GAMMA project [4] 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 II) and framework (section III). The research evaluation approach is described in section IV. Next, results of our empirical study are presented in sections V and VI. Finally, limitations of this research are discussed in section VII and research conclusions are presented in section VIII.

[1]http://www.google.com/websiteoptimizer/, verified 26/05/2010

[2]http://labs.mozilla.com/testpilot/, verified 26/05/2010

978-0-7695-4170-9/10 $26.00 © 2010 IEEE
DOI 10.1109/SEAA.2010.20

II. 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 [5]–[7] and software operation data acquisition techniques and tools have been developed [2], [8]. Little research has been initiated to incorporate all processes in one framework, however. Selby et al. [9] 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 [10] 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 III). Tautz and Althoff [11] 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[3].

A. 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. [12] 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. [13], 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 [14]. 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.

B. Quality (κ_Q)

Several software quality models with diverse sets of characteristics have been proposed, and as observed by Bøegh [15], many perspectives on what composes a software quality model's key quality characteristics exist. The ISO 9126 quality model [16] 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 [15]). 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. [17], 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 [18] propose a set of QoS attributes (e.g. *Cost*, *Reliability*, *Availability*),

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

as impact factors of service selection algorithm creation. In their research on the construction of a Web service quality model, Zeng et al. [19] 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.

C. 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 [20] 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 [21]. 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.

D. 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.

III. SOK Framework

Partially based on our observations of industry practices, the software operation knowledge framework (see figure 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 and 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.

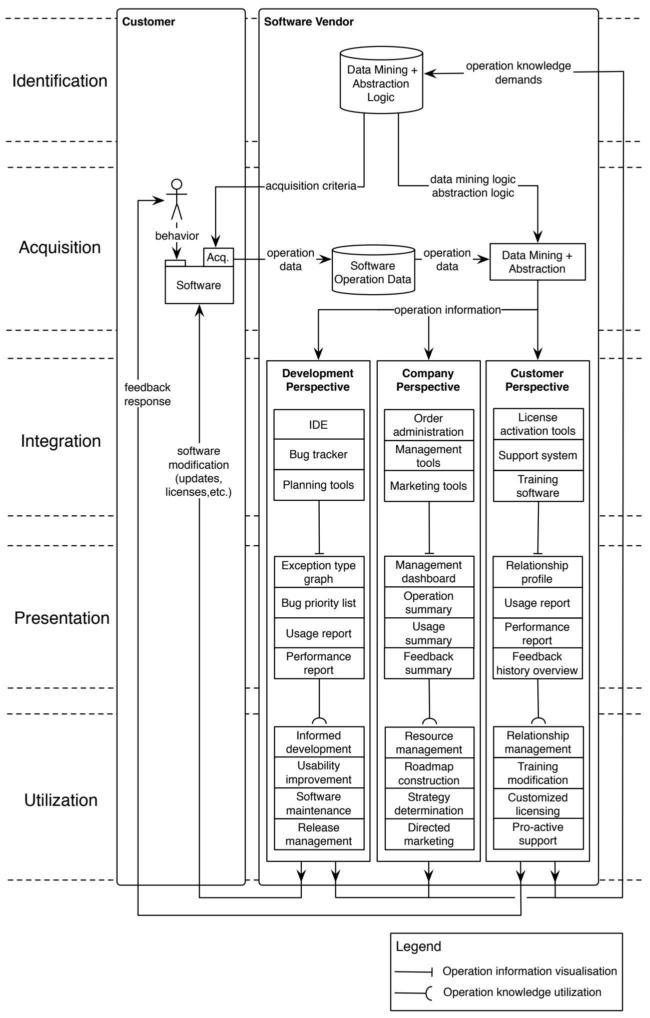

Figure 1. Software Operation Knowledge framework

A. 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. [22], are considered out the scope of the framework.

B. Processes

The SOK life cycle processes depicted in figure 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. Secondly, 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 [2], [4], [23]. 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 [8]. 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 [2] 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 III-C) 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).

C. 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 [24].

First, the *Development* perspective concerns all processes that contribute to production of software products that can readily be deployed at customers. Secondly, the *Company* perspective concerns processes indirectly related to software development, such as marketing, sales, and quality control. Thirdly, 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.

IV. 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. Secondly, 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.

A. Questionnaire Approach

The questionnaire consists of 21 questions divided over four sections[4]. 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 II, the focus group participants were asked whether they considered the knowledge type to be part of software operation knowledge. Thirdly, 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.

B. Case Study Approach

Case study techniques described by Yin [25] 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.

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

[4]See http://people.cs.uu.nl/schuurhw/soksurvey/ for a list of all questions.

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.

V. 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 II, 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 III, 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 III-B, 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 II, 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 mentioned 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.

VI. 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.

A. 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,

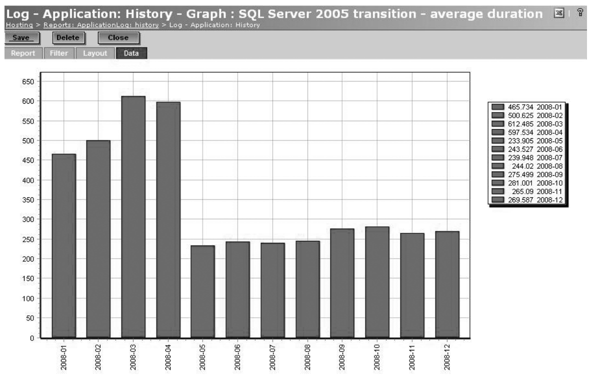

Figure 2. Performance graph of Wareex's software operation dashboard, showing the impact of a database version change on Lineex performance

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 III-B. 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 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 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 III-B, 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 software 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’.

B. 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 message, 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.

C. 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 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

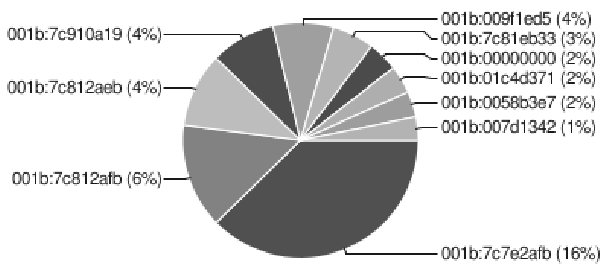

Figure 3. A graph from Ansta's SOK presentation tool, Denerr, showing the top 10 crash memory locations based on submitted error reports. 16% of the submissions report errors on one and the same memory location

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 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.

D. 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.

VII. 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 [25] 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.

VIII. 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.

IX. Acknowledgments

This research was supported by Stabiplan B.V., the Netherlands. We thank all questionnaire subjects and case study interviewees for sharing their ideas and experiences.

References

[1] H. Brelsford, M. S. Toot, K. Kiri, and R. V. Steenburgh, *Connecting to Customers*. Microsoft Press, February 2002.

[2] J. Bowring, A. Orso, and M. J. Harrold, "Monitoring Deployed Software Using Software Tomography," in *PASTE '02: Proceedings of the 2002 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*. ACM, 2002, pp. 2–9.

[3] A. Memon, A. Porter, C. Yilmaz, A. Nagarajan, D. Schmidt, and B. Natarajan, "Skoll: Distributed Continuous Quality Assurance," in *ICSE'04: Proceedings of the 26th Int. Conf. on Software Engineering*. IEEE Computer Society, 2004, pp. 459–468.

[4] A. Orso, D. Liang, M. J. Harrold, and R. Lipton, "Gamma System: Continuous Evolution of Software after Deployment," in *ISSTA'02: Proceedings of the 2002 ACM SIGSOFT International Symposium on Software Testing and Analysis*. ACM, 2002, pp. 65–69.

[5] C. Ebert and R. Dumke, *Software Measurement*. Springer, 2007.

[6] N. H. Madhavji, J. Fernandez-Ramil, Juan, and D. Perry, *Software Evolution and Feedback: Theory and Practice*. John Wiley & Sons, 2006.

[7] B. V. Rompaey, B. D. Bois, S. Demeyer, J. Pleunis, R. Putman, K. Meijfroidt, J. C. Dueas, and B. Garca, "SERIOUS: Software Evolution, Refactoring, Improvement of Operational and Usable Systems," in *CSMR'09: Proceedings of the European Conference on Software Maintenance and Reengineering*. IEEE Computer Society, 2009, pp. 277–280.

[8] H. van der Schuur, S. Jansen, and S. Brinkkemper, "Becoming Responsive to Service Usage and Performance Changes by Applying Service Feedback Metrics to Software Maintenance," in *23rd IEEE/ACM International Conference on Automated Software Engineering - Workshop Proceedings (ASE Workshops 2008)*. IEEE Computer Society, 2008, pp. 53–62.

[9] R. W. Selby, A. A. Porter, D. C. Schmidt, and J. Berney, "Metric-driven analysis and feedback systems for enabling empirically guided software development," in *ICSE'91: Proceedings of the 13th Int. Conf. on Software Engineering*. IEEE Computer Society, 1991, pp. 288–298.

[10] M. M. Lehman and J. F. Ramil, "The impact of feedback in the global software process," *Journal of Systems and Software*, vol. 46, pp. 123–134, 1999.

[11] C. Tautz and K.-D. Althoff, "Using Case-Based Reasoning for Reusing Software Knowledge," *Case-Based Reasoning Research and Development*, pp. 156–165, 1997.

[12] E. Putrycz, M. Woodside, and X. Wu, "Performance Techniques for COTS Systems," *IEEE Software*, vol. 22, no. 4, pp. 36–44, 2005.

[13] M. J. Johnson, C.-W. Ho, E. M. Maximilien, and L. Williams, "Incorporating Performance Testing in Test-Driven Development," *IEEE Software*, vol. 24, no. 3, pp. 67–73, 2007.

[14] A. Mani and A. Nagarajan, "Understanding quality of service for Web services." IBM, 2002, www.ibm.com/developerworks/java/library/ws-quality.html.

[15] J. Bøegh, "A New Standard for Quality Requirements," *IEEE Software*, vol. 25, no. 2, pp. 57–63, 2008.

[16] International Organization for Standardization, "ISO/IEC 9126-1:2001: Software engineering – Product quality – Part 1: Quality model," 2001.

[17] T. Gyimóthy, R. Ferenc, and I. Siket, "Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction," *IEEE Transactions on Software Engineering*, vol. 31, no. 10, pp. 897–910, 2005.

[18] T. Yu and K.-J. Lin, "Service selection algorithms for Web services with end-to-end QoS constraints," *Inf. Systems and E-Business Management*, vol. 3, pp. 103–126, July 2005.

[19] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng, "Quality Driven Web Services Composition," in *WWW '03: Proceedings of the 12th Int. Conf. on World Wide Web*. ACM, 2003, pp. 411–421.

[20] E. Simmons, "The Usage Model: Describing Product Usage during Design and Development," *IEEE Software*, vol. 23, no. 3, pp. 34–41, 2006.

[21] C. Kallepalli and J. Tian, "Measuring and Modeling Usage and Reliability for Statistical Web Testing," *IEEE Trans. on Software Engineering*, vol. 27, no. 11, pp. 1023–1036, 2001.

[22] S. Jansen, S. Brinkkemper, and A. Finkelstein, "A Sense of Community: A Research Agenda for Software Ecosystems," in *ICSE'09: Proceedings of the 31st ICSE Conference on Software Engineering*, 2009.

[23] J. Clause and A. Orso, "A Technique for Enabling and Supporting Debugging of Field Failures," in *ICSE '07: Proceedings of the 29th Int. Conf. on Software Engineering*. IEEE Computer Society, 2007, pp. 261–270.

[24] S. Brinkkemper and L. Xu, "Concepts of Product Software," *European Journal of Information Systems*, vol. 16, pp. 531–541, 2007.

[25] R. K. Yin, *Case Study Research: Design and Methods (Applied Social Research Methods)*. SAGE Publications, December 2002.