Reducing Maintenance Effort through Software Operation Knowledge: An Eclectic Empirical Evaluation

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 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.**

***Keywords*-software maintenance, bug localization, program comprehension, software process improvement, binary instrumentation, software feedback, empirical study**

I. 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 [1]. 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 [2]. 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.

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

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 [3], software vendors still acquire SOK in an unsophisticated manner [4]. 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 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 [5], [6]: an experiment and two case studies (i.e., field study [7]) 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 II. Next, the software operation knowledge acquisition and presentation technique is proposed (section III). Section IV introduces the prototype tool; the empirical evaluation approach and results are described in section V. Finally, research limitations (section VI), conclusions and future work (section VII) are presented.

II. 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 [8], [9], [10]. 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.

Secondly, SOK is also acquired to debug software. Clause and Orso [11] 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. [12] 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 ability 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.

Thirdly, several techniques for presenting software operation recordings exist [13], [14]. 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.

III. 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 II). Secondly, most approaches only work for a specific system or software application and thus are not generically applicable. Thirdly, 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 (A), recording (B) and a visualization (C) process and can be mapped onto the SOK framework [3]; a usage scenario[2] is provided in figure 1. Recording of software operation is preceded by a weaving process 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.

A. Weaving

Aspect-oriented programming (AOP) is effectively deployed in the domains of software monitoring and tracing [9], [15]. 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

[2]Note that the SOK framework integration process is optional [3] and omitted in this figure.

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.

B. Recording

When a SOK assembly is executed, three types of events $\varepsilon(\mu)$ are recorded for each method $\mu \in \mathcal{S}$: method entries, $\varepsilon_{entry}(\mu)$, method exits, $\varepsilon_{exit}(\mu)$ and unhandled exceptions, $\varepsilon_{exception}(\mu)$. 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 $\varepsilon(\mu)$, the SOK print is updated with additional data. Every event $\varepsilon(\mu)$ 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 $\varepsilon(\mu)$ 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 $\varepsilon_{entry}(\mu)$, 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 $\varepsilon_{exit}(\mu)$, 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 $\varepsilon_{exception}(\mu)$, 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.

C. 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.

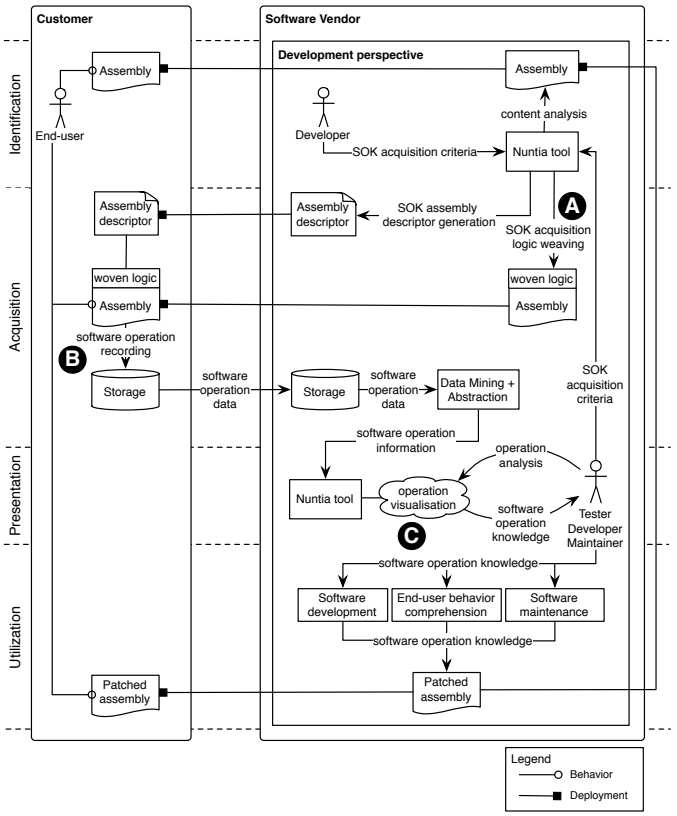

Fig. 1. SOK acquisition and presentation technique usage scenario

IV. 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 III, 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 1.

A. 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] [17]. Although Nuntia requires .NET, our technique is also applicable to other languages that allow binary instrumentation 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 [18] 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 ($\mathcal{S}$), 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.

B. Recording

Analogous to the recording mechanism description in section III, 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.

C. 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 2 shows a screenshot[4] of 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 have an elliptical or rhombic shape. Elliptical nodes represent

[3]Note that Nuntia may operate platform independently by weaving into assemblies created with the C# compiler that is part of the Mono project [16].

[4]Most figures in this paper are best viewed in color and are also available in high resolution at http://people.cs.uu.nl/schuurhw/nuntia/.

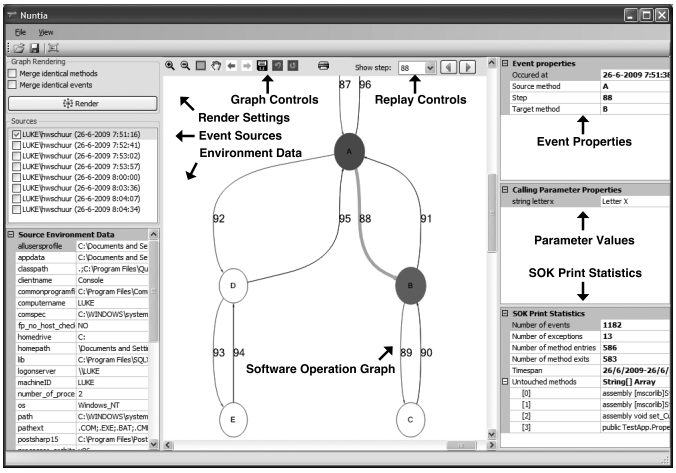

Fig. 2. Nuntia SOK print visualization and replay

methods; rhombic nodes represent exceptions. In figure 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 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 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 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 [19]. Nuntia is written in C# and requires version 3.5 SP1 of the .NET framework [17].

V. 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.

RQ1 *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.

RQ2 *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.

RQ3 *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. [5] and Kitchenham et al. [6], 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 I 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 II.

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

TABLE I
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 [17]: 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 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).

A. 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' [20]. 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 III. 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 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 IV, 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, RectangleEdge edge, List'1 v), PaintDotNet.Utility.SutherlandHodgman(RectangleF bounds, List'1 v) and PaintDotNet.Tools.SelectionTool.CreateSelectionPolygon() 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 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

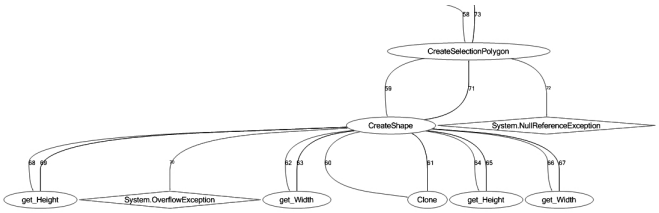

Fig. 3. Visualization of 'fixed ratio' bug of Paint.NET 3.35

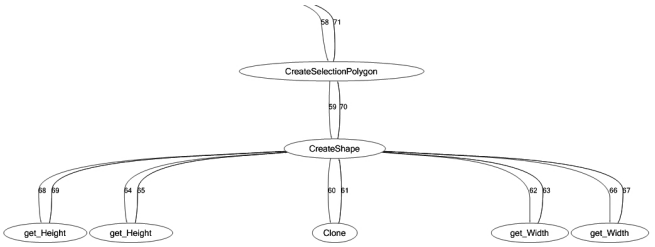

Fig. 4. No exceptions occur using Paint.NET 3.36: fixed ratio bug fixed

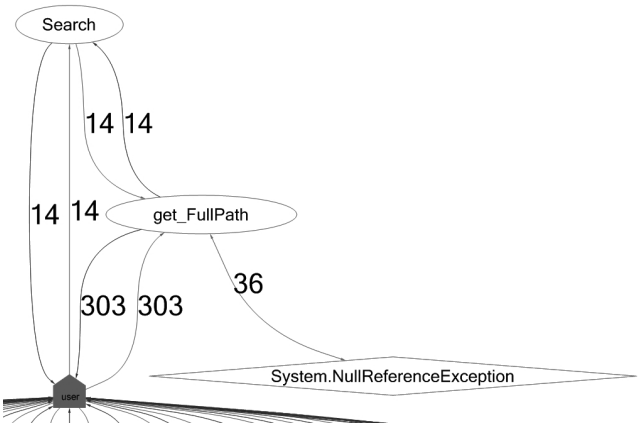

Fig. 5. Visualizing all acquired CADProd SOK prints simultaneously

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.

B. Industrial Case Studies

Approach

1) 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

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 II
QUESTIONNAIRE STATEMENTS AND RESULTS

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 II). Participating employees had an average of 11.6 years experience in information technology ($\sigma = 5.36$ years).

2) 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

1) CADProd: Figure 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 II and figure 6), the participants tended to agree with the statements, on average answering the statements with 3.9 ($\sigma = 0.83$). Participants agreed the most with statements S1 and S2 (these statements were answered with 4.4 ($\sigma = 0.61$) and 4.3 ($\sigma = 0.85$) 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, $\sigma = 0.56$) and S1 (4.4, $\sigma = 0.61$). Least consensus was reached on S10 ($\sigma = 1.11$). Participants agreed the least on statements S9, S10 and S6; these statements were answered with 3.56 ($\sigma = 0.93$), 3.63 ($\sigma = 1.11$) and 3.63 ($\sigma = 0.93$) on average, respectively. In the context of S10, product management, training and customer relationship management were mentioned as other activities supported by Nuntia.

2) 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.

C. 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 ($\sigma = 4.35$ 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 II). 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 II and figure 7), on average answering the statements with 3.83 ($\sigma = 0.80$). Participants agreed the most with statements S3 and S7 (these statements were answered with 4.4 ($\sigma = 0.50$) and 4.3 ($\sigma = 0.67$) 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 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.

The SOK acquisition and presentation technique in particular was valued especially because of its post-compilation and SOK

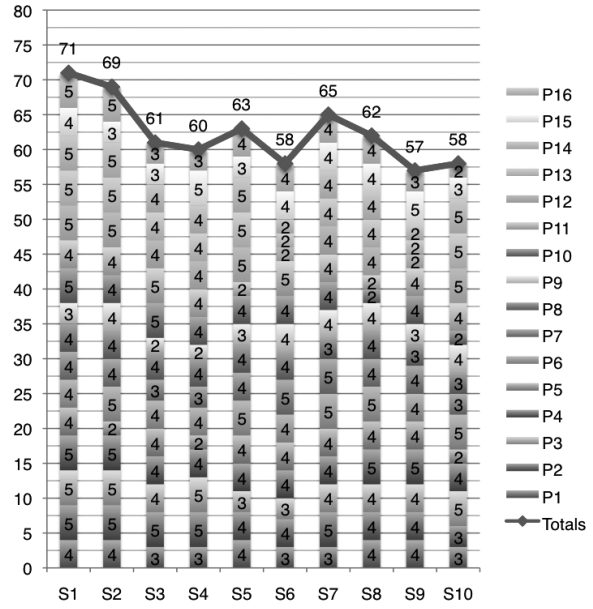

Fig. 6. Answers to questionnaire statements by CADComp employees

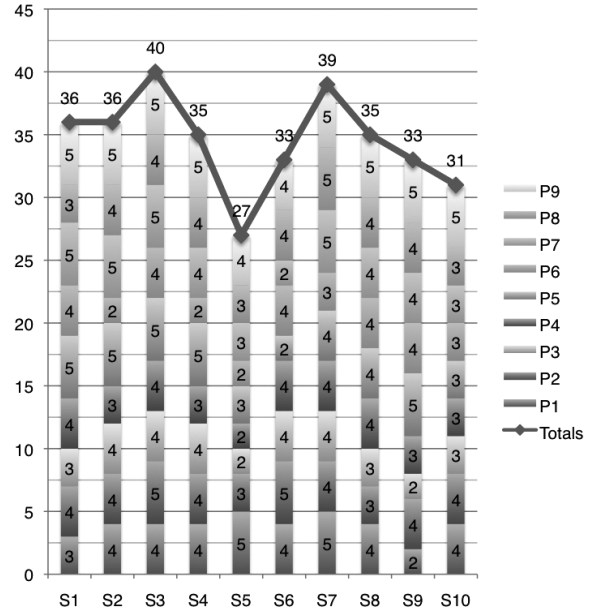

Fig. 7. Answers to questionnaire statements by focus group experts

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.

VI. 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.

VII. 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.

RQ1 *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.

RQ2 *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.

RQ3 *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 ($\sigma = 0.7$), 4.2 ($\sigma = 0.9$), 4.0 ($\sigma = 0.8$) and 3.8 ($\sigma = 0.9$) 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 ($\sigma = 0.6$) 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 ($\sigma = 1.0$) 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.

VIII. Acknowledgments

We thank all software vendors and participants, particularly G.W. Sloof, R. Dähne and A. van der Hoeven, for their cooperation and contributions. We thank A. Zaidman and J. Hage for their suggestions and comments.

References

[1] K. Glerum, K. Kinshumann, S. Greenberg, G. Aul, V. Orgovan, G. Nichols, D. Grant, G. Loihle, and G. C. Hunt, "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*. ACM, 2009, pp. 103–116.

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

[3] H. van der Schuur, S. Jansen, and S. Brinkkemper, "A Reference Framework for Utilization of Software Operation Knowledge," in *SEAA'10: 36th EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE Computer Society, 2010, pp. 245–254.

[4] S. Jansen, S. Brinkkemper, and R. Helms, "Benchmarking the Customer Configuration Updating Practices of Product Software Vendors," in *ICCBSS '08: Proceedings of the Seventh International Conference on Composition-Based Software Systems*. IEEE Computer Society, 2008, pp. 82–91.

[5] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, "Selecting Empirical Methods for Software Engineering Research," in *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer, and D. I. K. Sjberg, Eds. Springer London, 2008, pp. 285–311.

[6] B. Kitchenham, H. Al-Khilidar, M. Babar, M. Berry, K. Cox, J. Keung, F. Kurniawati, M. Staples, H. Zhang, and L. Zhu, "Evaluating guidelines for reporting empirical software engineering studies," *Empirical Software Engineering*, vol. 13, pp. 97–121, 2008.

[7] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design Science in Information Systems Research," *MIS Quarterly*, vol. 28, no. 1, pp. 75–105, 2004.

[8] J. Bowring, A. Orso, and M. J. Harrold, "Monitoring Deployed Software Using Software Tomography," *SIGSOFT Software Engineering Notes*, vol. 28, no. 1, pp. 2–9, 2003.

[9] A. Nusayr and J. Cook, "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*. ACM, 2009, pp. 7–10.

[10] B. Kristjánsson and H. van der Schuur, "A Survey of Tools for Software Operation Knowledge Acquisition," Department of Information and Computing Sciences, Utrecht University, Tech. Rep. UU-CS-2009-028, 2009.

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

[12] S. Narayanasamy, G. Pokam, and B. Calder, "BugNet: Continuously Recording Program Execution for Deterministic Replay Debugging," in *ISCA '05: Proceedings of the 32nd annual International Symposium on Computer Architecture*. IEEE Computer Society, 2005, pp. 284–295.

[13] B. Cornelissen, D. Holten, A. Zaidman, L. Moonen, J. J. van Wijk, and A. van Deursen, "Understanding Execution Traces Using Massive Sequence and Circular Bundle Views," in *ICPC '07: Proceedings of the 15th IEEE International Conference on Program Comprehension*. IEEE Computer Society, 2007, pp. 49–58.

[14] J. A. Jones, A. Orso, and M. J. Harrold, "GAMMATELLA: visualizing program-execution data for deployed software," *Information Visualization*, vol. 3, no. 3, pp. 173–188, 2004.

[15] 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.

[16] "The Mono Project," http://mono-project.com/.

[17] ".NET Framework," http://msdn.microsoft.com/netframework/.

[18] "The PostSharp Platform," http://www.postsharp.org/.

[19] L. Nachmanson, G. Robertson, and B. Lee, "Drawing Graphs with GLEE," *Graph Drawing*, pp. 389–394, 2008.

[20] "Paint.NET Roadmap and Change Log," http://www.getpaint.net/roadmap.html.