Becoming Responsive to Service Usage and Performance Changes by Applying Service Feedback Metrics to Software Maintenance

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

Abstract

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

1. Introduction

Nowadays, a lot is asked from software vendors. Their organizations have to be dynamic and agile, highly responsive to changes, while developing software at high speed, low cost and according to high quality standards. Two trends can be observed. First, a shift from Object-Oriented Development [1] to Service-Oriented Development [17] can be perceived: software vendors focus on the development of (SOA) Web services and online, rich internet applications instead of offline, component-based desktop applications [8]. In past years companies like Google, Yahoo and Microsoft have developed successful online applications like Gmail, Flickr, and Office Live. Secondly, while software vendors may have implemented a particular form of software usage data gathering mechanism analogue to Microsoft's error reporting functionality [3], few software vendors actually use the data that is collected. For example, still only very few software vendors make use of usage reports. The Dutch National Customer Configuration Benchmark Survey 2007 [9] concludes that only 28% of the software vendors sell software that automatically composes usage reports and shows that only 30% of the software vendors automatically compose and send bug reports.

In this paper we introduce Service Knowledge Utilization (SKU) as an approach to increase a software vendor's flexibility, and responsiveness to changes in the performance and usage of its service-based, online software, at specific customers and concerning its software in general. Furthermore, we present the SKU report, a report that quantifies the usage and feedback of a software vendor's service-based software and contains three metrics-based indices that express software quality. In literature, metrics are used to measure the quality of a service (QoS) [14, 28] and suggested as useful information that an infrastructure for managing and monitoring software can collect [5]. In the SKU report presented in section 2, metrics are used to illustrate and summarize changes in software performance and usage, based on gathered software usage and feedback data. Section 5 details Nuntia, a software prototype based on aspect-oriented programming that provides a concrete SKU implementation, including data gathering and SKU report generation functionality. A case study at a Dutch software vendor was performed to validate both the report and the prototype. While software usage data gathering may easily result in significant performance loss of the software being traced, results show that the integration of our prototype with target software has a negligible effect on the performance of the target software. Furthermore, research validation shows that the SKU report is considered valuable and supportive in management meetings on release management, requirements management and software maintenance. All case study results are presented in section 6. Finally, section 7 details conclusions and future research.

2. Service Knowledge Utilization

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

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

As already argued by Mendelson and Ziegler [15], *information awareness* (knowing what is going on in the surrounding world) enables enterprises to anticipate changes in the demands to their products and services. The authors define 'Listening to the Customer' as the first process that can be utilized to increase the information awareness of an enterprise [15]. In order to improve their customer and information awareness, enterprises should consider their customers as partners and as the primary source of information, innovation and renewal.

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

2.1. SKU Report

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

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

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

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

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

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

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

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

2.2. SKU Metrics

The research community has developed a wide range of metrics to measure and quantify the quality of services [10]. The metrics listed below were selected by way of a literature study focussing on the extent the metrics express service performance, usability and usage. This range of metrics (Reliability, Throughput, Latency, Accuracy, Availability, Usability, Reputation) is suggested by Farrell and Kreger [5] as useful information that a software infrastructure for managing and monitoring Web services can collect for a Web service, and is also used in QoS-related research [14, 28]. In the context of this paper, the metrics are used to construct three service indices that express the status of service-based software.

Reliability The term 'Reliability' has a number of definitions. It is defined as '*the probability that a request is correctly responded within a maximum expected time* *frame*' [29] or '*the quality aspect of a Web service that represents the degree of being capable of maintaining the service and service quality*' [21]. Analogue to [13], we define reliability as '*The rate of successful responded requests* '.

Throughput We define the throughput of a service as '*the number of successfully completed service requests over a time period*' [20]. Often, throughput is measured in requests per time unit [26].

Latency Latency is defined as the '*time taken between the moment the service request arrives and the moment the request is being serviced*' [20]. We define the latency of a service as '*the round-trip time between sending a request and receiving the response*' [21]. The throughput of a system (service) is related to its latency.

Accuracy In a generic way, 'accuracy' is defined as '*percentage of objects without data errors such as misspellings, out-of-range values, etc.*' [16]. More specifically, we define accuracy as '*the error rate produced by a service*' [20]. In practice, this metric indicates the amount of exceptions (both handled and unhandled) a service or service method generates.

Availability The term 'Availability' has various definitions: '*The quality aspect of whether the Web service is present or ready for immediate use, and represents the probability that a service is available.*' [21], '*The percentage of time a source is accessible based on technical equipment and statistics*' [16] or '*The probability a system is up*' [20, 26]. In this paper, we use the latter definition.

Usability Seffah et al. [22] developed a consolidated usability measurement model based on the research of Shackel [23] and Nielsen [18]. Among other criteria, the 'minimal action' criterion is identified: '*The capability of the software product to help users achieve their tasks in a minimum number of steps*'. Given the nature and role of Web services in the context of this research — users actively 'interact' with a (process-centric or enterprise level) Web service via its graphical user interface — the usability of Web services is defined as the 'minimal action' criterion: the number of steps a user has to perform to achieve a particular task.

Reputation '*The reputation of a service is a measure of its trustworthiness. It mainly depends on end user's experiences of using the service. Different end users may have different opinions on the same service*' [29]. In the context of this research, reputation of a service is expressed in a '*user grade from 1 to 10 based on personal preferences and professional experience*' [16].

The notation of the metrics defined above is depicted in table 1. m_{jk} represents a method of a service s_j published by a software vendor.

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

Table 1. Metrics notation

Before further detailing how the service indices are constructed, it is important to observe that services may be bound to service contracts and corresponding service levels, specifying the required performance of a service. Specifically, such a contract is a Service Level Agreement (SLA) between the service provider and the service requester, inter alia describing (un)acceptable service levels [24, 12]. A service level is defined as some mark by which to qualify acceptability of a service parameter [12], for example a guaranteed minimal availability of a service. Concerning the construction of the service indices, for all metrics, service levels are used to calculate the metric *scores*. Depending on the type of service level (minimal or maximal), two types of metric scores can be identified. The first type applies to the metrics Reliability, Throughput, Availability and Reputation. If the Reliability, Throughput, Availability or Reputation of a service method is equal to or greater than the corresponding minimal value of that metric prescribed in the SLA, $q_{\{Rel,T,Av,Rep\}_{SLA}}(s_j)$, the metric score of the service method on that metric $r_{\{Rel,T,Av,Rep\}}(m_{jk})$ is equal to 1.0 — the method is operating according to its service level. Otherwise, the metric score is equal to the Reliability, Throughput, Availability or Reputation metric value, divided by the corresponding SLA value for that metric. If the metric score of a method is equal to 0.0, the method is failing maximally (because its service is offline, or always throwing an unhandled exception, for example) with respect to its SLA.

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

3. SKU Report Indices

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

Service Performance Index Represents the average performance of the services the software vendor has published.

Service Usability Index Represents the average usability of the services the software vendor has published. Gives an indication of to which extent users value and rate the services of a software vendor, and to which extent they are satisfied with using those services.

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

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

3.1. Service Performance Index

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

$$\pi(m_{jk})) = r_{Rel}(m_{jk}) \cdot r_T(m_{jk}) \cdot r_L(m_{jk}) \cdot r_{Acc}(m_{jk}) \cdot r_{Av}(m_{jk}) \tag{1}$$

Equation 2 shows how the Service Performance Index is calculated for a service $s_{ix} \in S_i$ — where S_i is the set of services licensed by customer c_i — as the average performance of its methods.

$$\pi(s_{ix}) = \frac{\sum_{k=1}^{q} \pi(m_{ixk})}{q} \tag{2}$$

The equation below illustrates how to calculate the Service Performance Index for a customer c_i. This index expresses the average performance of the set of services that this customer has licensed. The constant factor M is added to stress the aggregate character of the customer-level service indices, and improve the comparability of customers based on those indices. With this constant factor included, customer-level service indices 'rate' the software vendor's customers based on the services they have licensed, on a scale of 0 to M. $M = 5$, $M = 7$ and $M = 10$ are proven intuitive by research [4].

$$\pi(c_i) = M \cdot \frac{\sum_{x=1}^{v} \pi(s_{ix})}{v} \tag{3}$$

3.2. Service Usability Index

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

$$\upsilon(m_{jk}) = r_U(m_{jk}) \cdot r_{Acc}(m_{jk}) \cdot r_L(m_{jk}) \cdot r_{Rep}(m_{jk}) \tag{4}$$

Equation 5 shows how the Service Usability Index is calculated for a service $s_{ix} \in S_i$ — where again, S_i is the set of services licensed by customer c_i — as the average usability of its methods.

$$\upsilon(s_{ix}) = \frac{\sum_{k=1}^{q} \upsilon(m_{ixk})}{q} \tag{5}$$

Equation 6 illustrates how to calculate the Service Usability Index for a customer c_i. This index expresses the average usability of the set of services that this customer has licensed.

$$\upsilon(c_i) = M \cdot \frac{\sum_{x=1}^{v} \upsilon(s_{ix})}{v} \tag{6}$$

3.3. Service Client Utilization Index

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

$$\kappa(m_{jk}) = r_T(m_{jk}) \cdot r_L(m_{jk}) \cdot r_{Rep}(m_{jk}) \qquad (7)$$

Equation 8 shows how the Service Client Utilization Index is calculated for a service $s_{ix} \in S_i$ — where again, S_i is the set of services licensed by customer c_i — as the average client utilization of its methods.

$$\kappa(s_{ix}) = \frac{\sum_{k=1}^{q} \kappa(m_{ixk})}{q} \qquad (8)$$

Equation 9 illustrates how to calculate the Service Client Utilization Index for a customer c_i. This index expresses the average utilization of the set of services that this customer has licensed.

$$\kappa(c_i) = M \cdot \frac{\sum_{x=1}^{v} \kappa(s_{ix})}{v} \qquad (9)$$

Having presented all service index constructions on all levels (methods, services and customers), note that all metric scores $r_{\{Rel,T,L,Acc,Av,U,Rep\}}(m_{jk})$ have equal weight in the construction of each of the service indices. As detailed in section 6, a software vendor may assign different weights to each of the metric scores, in line with its strategy focus. A number of observations can be made. First, the domains of the index values $\pi(s_{ix}), \upsilon(s_{ix}), \kappa(s_{ix}), \pi(m_{jk}), \upsilon(m_{jk})$ and $\kappa(m_{jk})$ are equal to the domains of the metric scores.

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

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

Obviously, the change in domains is caused by the constant factor M included in the formulas of all customer-level service indices. Thirdly, note that of all concepts, the 'event' concept is not mentioned within the construction of the indices. This is because events are used to calculate the numerous metric values. Similarly, the concept of an event type is introduced to enable event categorization. In section 5, an example SKU report implementation is presented, as part of the description of the software prototype Nuntia that provides SKU report generation functionality.

3.4. Related Work

In research, QoS metrics are often used in the process of web service discovery, selection or composition. Yu and Lin [28] propose a set of service selection algorithms for web services with QoS constraints. Compared to the work of Yu and Lin and others, our research focuses on the application of metrics as a means to achieve software quality goals and increase the responsiveness of software vendors to performance and usage changes in their software. Farrell and Kreger [5] propose types of information that can be collected with respect to web service management, and indicate that this information could serve as a basis for service usage monitoring. However, the solutions the authors provide do not feature any form of (summarizing) report functionality similar to the SKU report presented in this paper. Finally, Papapetrou and Papadopoulos [19] present an AOP-based logging tool for a component-based software. While their application of aspect-oriented programming is similar, our solution is also utilized with service-based software and environments.

4. Research Approach

Part of the research was the development of the SKU approach and the SKU report, including its service metrics and indices. Furthermore, a software prototype (section 5) was developed to provide an service knowledge utilization implementation for software vendors developing service-based software. A case study at a Dutch software vendor was conducted to validate the service metrics and indices, as well as the SKU reports generated by the prototype.

4.1. Company Identification

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

4.2. Case Study Approach

In order to identify the case study company and gather facts on which the results of the case study are based, four case study techniques have been used. Yin [27] has defined six sources of case study evidence. Numerous sources are utilized as part of this research.

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

4.3. Hypotheses

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

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

Responsiveness Increase The software vendor's responsiveness (the speed at which changes are implemented based on performance and feedback data), for example to issues reported to its help desk, may increase when the vendor has implemented the SKU prototype. With the SKU presented, the usage of software is logged continuously. Furthermore, actions can be defined that have to be executed when particular events occur and are logged. For example, an action could describe that as soon as a severe exception occurs, the help desk of the software vendor has to be informed.

4.4. Validation Methods

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

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

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

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

5. SKU Software Prototype

The goal of the software prototype, Nuntia, is to gather software usage and feedback data from software products

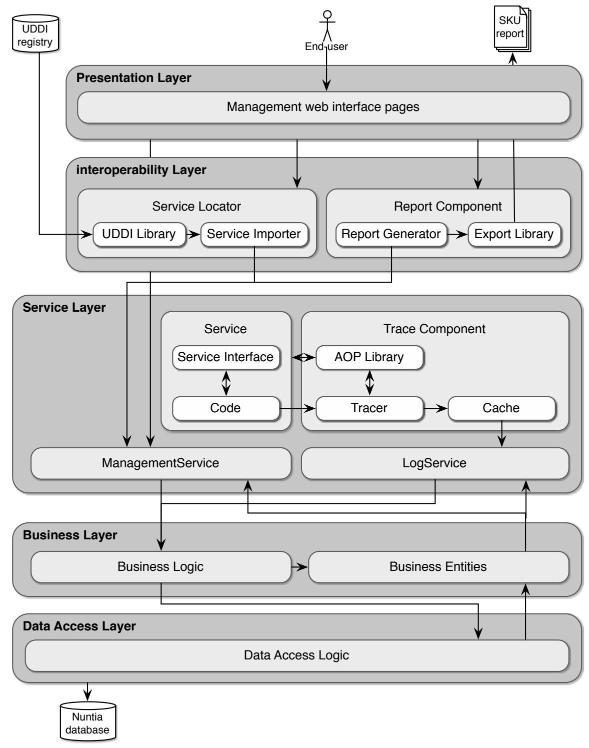


Figure 1. Architecture of Nuntia

and services that are developed by software vendors and are licensed by one or more participants in the vendor's software supply network, and to provide means to extract valuable real-time software (usage) knowledge from the data gathered, in order to increase a software vendor's flexibility and responsiveness to changes in software usage and performance.

In order to gather service usage data of a service oriented architecture's (SOA) services, Nuntia can be easily integrated in a SOA by using the UDDI-based Service Locator. To actually gather service usage and feedback data, Nuntia's trace component is integrated in the target service. In addition, Nuntia provides an SKU report generation implementation: when service usage and feedback data is gathered, SKU reports based on this data are generated with the Report Generator. Finally, two web services are part of Nuntia. First, the ManagementService forms the interface to the Nuntia database and is used by Nuntia's web interface to enable data manipulation. Secondly, the LogService is used to gather, cache and store service usage and feedback data. While the ManagementService is used by the software vendor itself to register its customers and services, the LogService is also accessible for (pilot) customers and other partners in the vendor's software supply network, in order to register their service usage and feedback. See figure 1.

Nuntia was developed using Visual Studio 2008 and the .NET framework 3.5, both from Microsoft [25]. All code was written in C# 3.0. Concerning Nuntia's trace component, one technique is utilized extensively: Aspect-Oriented Programming (AOP), created by Kiczales et al. [11]. AOP is a programming paradigm that contributes to the area of *separation of concerns*. Some concerns, *cross-cutting* concerns, cannot be separated cleanly, because they are related to numerous, distinct modules of a program (the concerns 'cut' across different program modules). Examples of cross-cutting concerns are security and logging. Both concepts are often needed throughout all modules a program consists of, because the need to log exceptions exists program-wide, for example. In AOP, code that is needed program-wide (a so-called *advice*), is injected into the existing code of the program, at locations (*point-cuts*) determined by the programmer. The combination of an advice and a point-cut is called an *aspect* [6]. For the process of advice injection (also referred to as *weaving*) to take place, the programmer does not have to adjust the program code structurally. Next, the trace component is described in more detail.

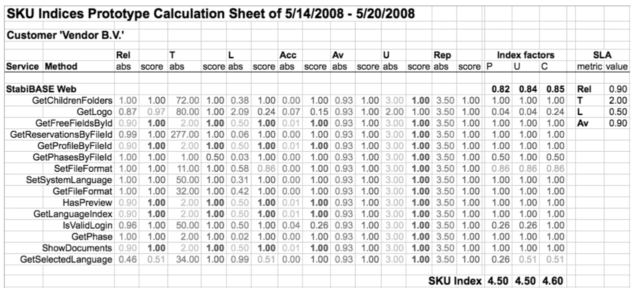
5.1. Tracing

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

AOP Library Nuntia utilizes the (compile-time) weaving functionality of the PostsharpAspNet to trace the usage and performance of services. PostsharpAspNet was released in February, 2008 and is an experimental library based on the PostSharp Laos aspect weaver [7].

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

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



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

Customer 'Vendor B.V.'

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

Figure 2. Screenshot of a SKU report generated by Nuntia

5.2. Report Generation

When receiving a report generation request, the report generator requests all relevant data related to the time frame, customers and services from the Nuntia database. Next, the report generator calculates metric scores and service indices for the selected services (including the services' web methods) of a selected customer in the selected time period. When no data is available for a particular metric, the metric score is set to 1.0 (the SLA requirement is met). This is done to prevent negative influences on the metric scores and service index values by methods that are not called within the selected time frame (and for which no data is stored)[1]. When this is the case, a 'not representative' annotation is added to the data. All metric score values have a color that indicates their status. Green indicates that the SLA value is met, orange indicates that the SLA value is not met and red indicates that the metric score is lower than half the SLA value. Concerning the customer-level service index calculation: the constant M is equal for all reports can be set via Nuntia's web interface. Apart from the metric score and service indices calculation, a list of all events occurred within the selected time frame is included in the SKU report. For example, for each exception event, the list contains exception details and stack trace and detailed environment information. Furthermore, the list contains data of all request and response events, as well as user interface paths for all user interface events. When all required data is gathered, the data is sent to an export library which compiles all data into a report that is presentable to the end-user. Figure 2 depicts a screenshot of a Nuntia-generated SKU report ($M = 5$).

6. Results

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

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

$$\Delta = \sum_{z=1}^{r} \delta_1(e_{jz}) + \delta_2(e_{jz}) = \sum_{z=1}^{r} \rho_T(e_{jz}) - \rho_O(e_{jz}) \quad (10)$$

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

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

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

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

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

Table 2. Performance test results

software and Nuntia were compiled and running in debug mode. Cache capacity was set to 30 events.

All test results show a low Δ/T rate: the delay caused by the tracing process is spread over a relatively long time. Furthermore, in all test sessions except B, $5 < \Delta < 15$. In test B, the test machine was completely restarted after each test. As detailed in table 2, this increases Δ significantly, due to required connection and plugin initialization. Test results also show a low average delay per event (in seconds, not listed in table 2):

$$\frac{\Delta_A + \Delta_B + \Delta_C + \Delta_D}{r_A + r_B + r_C + r_D} = 0.008 \qquad (11)$$

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

SKU Report Overall, the experts indicated that the report contains data that is valuable to software vendors and that the report supports informed decision making in the areas of software development and software maintenance. Concerning the report's service indices, experts indicated all service indices should be included in the SKU report. In their opinion, especially the service client utilization index was considered a valuable indication. This is in line with earlier interviews and observations: both developers and project managers indicated that the vendor has difficulties in determining whether or not an existing feature is still used (and should not be removed yet) or whether a new feature will be used by their end-users. Except for the throughput, availability and reputation metrics, the experts agreed that all metrics proposed should be included in the report. They denoted that the throughput and availability metrics especially are of importance with respect to process-centric services. Unlike the vendor's service software (which can be characterized as a public enterprise service), services of this type have to meet a specific demand, and thus guarantee a particular level of throughput or availability. Project and product managers also indicated that only proper and serious feedback will be valuable. Therefore, they proposed to include the reputation metric primarily with pilot customer tests. Furthermore, the availability metric was considered as less valuable because the software may not be the cause of a low availability score, the experts reasoned. The events overview in the report was considered optional and only valuable for debugging purposes. It was suggested to calculate the metrics on more levels of detail and to base the report structure on the structure of the target software. While the data contained in the report was expected to be taken into account when taking decisions related to operational, tactical and strategical management, product managers indicated that the SKU report will be primarily used in tactical management meetings (release management meetings, for example) and added that the report will have to be summarized in order to be of use in strategical meetings. It was also denoted that a successful SKU implementation would require some form of change management: people will have to get used to involve the knowledge contained in the report in their daily work, for example. The report will be generated once a month.

7. Conclusions and Future Research

Case study results show that Nuntia is expected to contribute to a software vendor's responsiveness to software performance and usage changes, and thus is an effective SKU implementation. The vendor has decided to integrate Nuntia with its live software in the near future because of the negligible performance loss this integration involves and the valuable information the SKU report provides. The SKU implementation is expected to contribute to a software quality increase on the short term and a time-to-market decrease on the long term: the vendor noted that it will first have to invest into the process of acting upon the knowledge, before it is able to structurally improve its responsiveness based on the gathered knowledge. The vendor expects to base software maintenance processes (requirements elicitation and bug fix priority assignment) on the SKU reports generated by Nuntia, and be able to react before an end-user calls for support. Finally, Nuntia's architecture (loosely-coupled services, use of AOP) enables easy integration with other software. We consider both hypotheses (decrease of software vendor's time to market and increase of vendor's responsiveness) supported by the research validation.

While the results of this research are promising, further research is needed to improve the applicability of the SKU report and the performance of the prototype. We will research data mining techniques and software tomography [2] to extract sophisticated statistics from large software knowledge repositories and integrate these in the report, and keep performance loss of the target software negligible under heavy use circumstances. Future research will also focus on integration of service knowledge in software development environments, SKU implementation change management requirements, and feedback visualization techniques.

References

[1] G. Booch. *Object-Oriented Analysis and Design with Applications (3rd Edition)*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004.

[2] J. Bowring, A. Orso, and M. J. Harrold. Monitoring Deployed Software Using Software Tomography. *SIGSOFT Software Engineering Notes*, 28(1):2–9, 2003.

[3] H. Brelsford, M. Toot, K. Kiri, and R. van Steenburgh. *Connecting to Customers*. February 2002.

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

[5] J. Farrell and H. Kreger. Web services management approaches. *IBM Systems Journal*, 41(2), 2002.

[6] R. E. Filman, T. Elrad, S. Clarke, and M. Akşit. *Aspect-Oriented Software Development*. Addison-Wesley, Boston, 2005.

[7] G. Fraiteur. User-friendly aspects with compile-time imperative semantics in .NET. To appear in the proceedings of the 7th International Confernce on Aspect-Oriented Software Development (AOSD.08).

[8] M. H. Ibrhaim, K. Holley, N. M. Josuttis, B. Michelson, D. Thomas, and J. deVadoss. The future of SOA: what worked, what didn't, and where is it going from here? In *OOPSLA '07: Companion to the 22nd ACM SIGPLAN conference on Object oriented programming systems and applications companion*, pages 1034–1038, New York, NY, USA, 2007. ACM.

[9] S. Jansen and S. Brinkkemper. A benchmark survey into the customer configuration processes and practices of product software vendors in the Netherlands. In *7th IEEE International Conference on Composition-Based Software Systems (ICCBSS)*, pages 82–91, 2008.

[10] S. Kalepu, S. Krishnaswamy, and S. W. Loke. Verity: a QoS metric for selecting Web services and providers. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering Workshops (WISEW '03)*, pages 131–139. IEEE Computer Society, 2003.

[11] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-Oriented Programming. In M. Akşit and S. Matsuoka, editors, *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997.

[12] L. Lewis. *Managing Business and Service Networks*. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[13] A. Mani and A. Nagarajan. Understanding quality of service for Web services. http://www.ibm.com/developerworks/java/library/ws-quality.html.

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

[15] H. Mendelson and J. Ziegler. *Survival of the Smartest: Managing Information for Rapid Action and World-Class Performance*. John Wiley & Sons, Inc., New York, NY, USA, 1999.

[16] F. Naumann, U. Leser, and J. C. Freytag. Quality-driven Integration of Heterogenous Information Systems. In *VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases*, pages 447–458, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[17] E. Newcomer and G. Lomow. *Understanding SOA with Web Services (Independent Technology Guides)*. Addison Wesley Professional, 2004.

[18] J. Nielsen. *Usability Engineering*. Academic Press, Boston, MA, USA, 1993.

[19] O. Papapetrou and G. A. Papadopoulos. Aspect oriented programming for a component-based real life application: a case study. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 1554–1558, New York, NY, USA, 2004. ACM.

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

[21] A. Sahai, J. Ouyang, V. Machiraju, and K. Wurster. Specifying and Guaranteeing Quality of Service for Web Services through Real Time Measurement and Adaptive Control. http://www.hpl.hp.com/techreports/2001/HPL-2001-134.html.

[22] A. Seffah, M. Donyaee, R. B. Kline, and H. K. Padda. Usability measurement and metrics: A consolidated model. *Software Quality Journal*, 14(2):159–178, June 2006.

[23] B. Shackel. Usability — context, framework, definition, design and evaluation. pages 21–37, 1991.

[24] D. Verma. *Supporting Service Level Agreements on IP Networks*. Macmillan Technical Publishing, 1999.

[25] .NET Framework Programming in Visual Studio. Microsoft.

[26] G. von Bochmann, B. Kerhervé, H. Lutfiyya, M.-V. M. Salem, and H. Ye. Introducing QoS to Electronic Commerce Applications. In *ISEC '01: Proceedings of the Second International Symposium on Topics in Electronic Commerce*, pages 138–147, London, UK, 2001. Springer-Verlag.

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

[28] T. Yu and K.-J. Lin. Service selection algorithms for Web services with end-to-end QoS constraints. In *Proceedings of the IEEE International Conference on e-Commerce Technology, 2004 (CEC '04)*, pages 129–136, 2004.

[29] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng. Quality Driven Web Services Composition. In *WWW '03: Proceedings of the 12th International Conference on World Wide Web*, pages 411–421, New York, NY, USA, 2003. ACM.