Thesis for the Degree of Master of Science

SKU: Software Vendor Meets End-user

On the Utilization of Service Software Knowledge in a Software Supply Network

Henk van der Schuur
INF/SCR-07-77

August 2008

Supervisor	Dr. S. R. L. Jansen	Universiteit Utrecht
Second supervisor	Prof. dr. S. Brinkkemper	Universiteit Utrecht
Daily supervisors	M. van der Elst B.	Stabiplan B.V.
	J. Schlingmann	Stabiplan B.V.

Center for Organization and Information
Department of Information and Computing Sciences
Utrecht University
Utrecht, The Netherlands

Stabiplan B.V.
Bodegraven, The Netherlands

Universiteit Utrecht

Stabiplan

Abstract

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 to service-oriented development can be perceived. Secondly, while software vendors may have implemented a particular form of software usage data gathering mechanism, few software vendors actually use the data that is collected. Software vendors are unaware of how their software performs in the field and 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 thesis, a concept called Service Knowledge Utilization (SKU) is introduced as an approach to increase a software vendor's flexibility, and responsiveness to changes in the performance and usage of its service-based online software, at specific customers and concerning its software in general. Furthermore, the SKU reporting toolset is presented. The real-time SKU report quantifies the usage and feedback of a software vendor's service-based software and contains three metrics-based indices that express software quality. Furthermore, a software prototype that provides a concrete SKU implementation is presented. The prototype is based on aspect-oriented programming techniques and includes data gathering and SKU report generation functionality.

An extensive 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 the prototype with target software has a negligible effect on the performance of the target software. Research validation shows that the SKU report is considered valuable and supportive in management meetings on release management, requirements management and software maintenance.

By using the SKU approach, vendors can make informed decisions with respect to software requirements management and maintenance. While validation shows high potential of the approach, a successful implementation will require change management at the software vendor.

Preface

This master's thesis is the result of eleven months of quite demanding and challenging work. It forms the conclusion of the bachelor and master courses I did during the past five years. Simultaneously, this thesis characterizes the type of contribution I want to deliver to the research world of information and computing sciences and to the world in general: inventive, theoretically and scientifically sound, and with a decent (and validated) level of practical, 'real world value' and applicability.

As I experienced during these months, the process of delivering that type of contributions can be tough and difficult. But simultaneously, the process can be exciting and gratifying. It challenges the mind and the gray matter, especially in its initial and final phases. I decided to express these experiences succinctly in the quotes that precede the first chapter of this thesis.

The last quote is from the Bible. The passage that is referred to says that knowledge tends to make people self-important, chesty and arrogant. To me, the passage is of value, because it keeps me (and other people) grounded. I want to take this opportunity to thank God for giving me wisdom and guidance throughout my life. It is under His grace that we live, learn and flourish.

During this thesis research, I have been accompanied and supported by many people. Now the thesis is finished, it is my great pleasure to thank them. I would like to thank dr. Slinger Jansen, my first supervisor, for the frequent thesis reviews, for the numerous interesting discussions we had and for remembering me that research concepts should be validated in practice in order to be sound and really valuable. Next, I would like to express my gratitude to prof. dr. Sjaak Brinkkemper, my second supervisor, for his critical but constructive notes and for helping me to make my PhD plans a reality. Finally, I want to thank dr. Rik Bos for his thorough and stimulating feedback.

Most of the research work took place at Stabiplan in Bodegraven. First and foremost, and want to thank my supervisors at Stabiplan, Martin van der Elst and Johan Schlingmann for providing me a pleasant environment for doing research. Johan, thanks for your enthusiastic input in (non-) work related conversations and for providing me your business view on my ideas and concepts. Gratitude also goes to Gijs Willem Sloof, Stabiplan's CEO. I really admire his ability to exactly pinpoint the weaknesses of imaginative concepts or abstract theories after having observed them very shortly. Thanks for the discussions and conversations we've had and for realizing my future PhD course. I also want to thank Stabiplan's Rob, Ronald, Paul, Jos, Krijn, Henk-Jan and all interviewees for their time and input. Finally, I want to thank Niels van Pelt for the numerous in-depth conversations we've had during our lunch break walks in Bodegraven. I really enjoyed them.

This thesis was completed thanks to the support of my loving parents, family and friends. Thanks for preventing me from getting too serious and for the overall relaxation during this master's thesis research. Finally, I want to thank Sacha, my beloved. It would have been much harder without your love, encouragement and support. Thank you.

Henk van der Schuur, August 2008

Contents

List of Figures

List of Tables

"Problems are to the mind what exercise is to the muscles. They toughen and make strong."

Norman Vincent Peale (American Protestant Clergyman and Writer, 1898–1993)

"If it's not frustrating, you're probably tackling problems that are too easy."

Peter Clive Sarnak (Mathematician, 1953–)

"Those who think they know something really don't know as they ought to know."

1 Corinthians 8:2 (GNB)

Chapter 1

Introduction

Nowadays, a lot is asked from software vendors: their organizations have to be highly dynamic and agile, and have to face new problems concerning the (among other areas) development and management of business software. A number of trends can be observed.

First, concepts like the Service-Oriented Architecture, Software as a Service and Web service composition are increasingly being accepted and implemented. Secondly, a shift from offline desktop applications towards online, (often service-based) Rich Internet Applications can be observed. Thirdly, in order to stay competitive, software vendors need to be responsive to changes in customer and market demands and situations more and more [60].

As will be explained in this thesis, these observations are the cause of a number of changes to the 'traditional' life cycle of component-based software. This introduction presents these observations in more detail and places them in context. Furthermore, the concept of (Continuous) Customer Configuration Updating ((C-)CCU) is introduced. Finally, the company at which this thesis was carried out, is identified. This thesis research approach is detailed in chapter 2.

1.1 Eras of Information Technology

Concerning the history and evolution of the Information Technology, three eras, or 'waves', can be identified. Nolan and Norton have developed a model that describes the growth processes and development of the use of IT in organizations [82]. Later, Mutsaers et al. [77], expanded their *Stages Theory* with three eras that all have their own characteristics in both business and IT terms.

The first era is known as the 'Data Processing' (DP) era. Subsequently, the era of 'Information Technology' (IT) started. Nowadays, the 'Network' era is recognized as the current era.

1.1.1 Data Processing Era

In the early days (the DP era dated from the 1960s until the 1980s) of information technology, companies focussed on management strategies like Total Quality Management (TQM), and business objectives of the companies of that time often were strongly related to cost reduction (effective resource control), process efficiency and processing speed. The logical processes were strictly algorithmic, running on workstations connected to and controlled by mainframes. Referring to the DIKW (Data, Information, Knowledge, Wisdom) hierarchy [1; 25], this era is often characterized by 'Data' — which was the main added value of the Information Technology in this era.

At the end of the Data Processing era, more structured (less 'creative') approaches to programming, testing and documentation came in use. Furthermore, more discipline was introduced with respect to 'change control procedures', specifications and testing [110].

1.1.2 Information Technology Era

The IT era dated from 1980 to around 1995. This period is characterized by the introduction of the Personal Computer (PC), accompanied by personal software (text processors, spreadsheets, Desktop Publishing software). Companies become more process-oriented and use IT to enable new methods of doing business. Information is spread over mainframe databases, departmental computers and personal computers. Sophisticated tools are needed to administer the company-wide network of distributed data, while new system development activities focus on realizing integration with suppliers and customers [77]. Companies focus on service improvement and management approaches like Business Process Reengineering (BPR).

Nearing the end of the IT era, IT is organized per division or business unit to decrease the dependency on a central IT facility. This is mainly done in the context of risk and responsibility spreading. IT shifts from a technological concept to a strategy-driven management resource. Comparing to the DIKW hierarchy mentioned before, this era is often characterized by 'Information'.

1.1.3 Network Era

The Network era began around 1995, and still is the current era. Mutsaers et al. [77] predict a continuous shift towards 'open' and public platforms such as Internet and intranet solutions. In this process, we (will) see attempts of office automation integrated into the enterprise, e.g. office software freely communicating with corporate applications. Obsolete systems from the IT-era will be completely taken out of production, and the entire applications portfolio of a (software) company will be hardware-independent. External interfacing will be completely standardized on a global level. *Networked* companies are built of loosely coupled processes [46], focusing on flexibility, customer satisfaction and collaboration with other commercial businesses in the business network. They require high information awareness and flexibility in order to adapt to new customer needs, to exploit new markets and to become a dynamic and transparent enterprise — especially in volatile environments with high speed changes and opportunities appearing and disappearing quickly [36].

Software is developed at high speed, low cost and according to high quality standards. The software vendor's product portfolio consists of temporary products which are discarded when not needed any more. Portfolios will not be expanded much, but will be changed rapidly because of fast changes in business needs. Referring to the DIKW hierarchy, the Network era is often compared to 'Knowledge'.

1.2 The Rise of Services

During the data processing era described in section 1.1.1, practically no attention was paid to sharing application logic and data across multiple machines — let alone sharing these across multiple organizations [110]. Developing systems to automate billing, accounting and order management was very challenging. One could imagine that basing all systems on a standardized and reusable architecture was even more challenging, and that few companies were in the position to do so.

In the current (network) era companies have automated various operational business functions and processes. However, the increasing demand on enterprises to be highly dynamic and agile organizations and to respond quickly to market situations and demands, force these enterprises to integrate their processes and collaborate with external companies [77]. Ultimately, services of external enterprises are not only utilized for the benefit of information retrieval, but also for the processing of supportive functions (stock market syndication, payroll administration, etc.). The company offering the service at the lowest price (or any other criteria) is automatically selected, the job is assigned to the specific service of the external company and the initiating service waits for response. Summarizing, enterprises are moving and transforming into *Service-Oriented Enterprise*s (SOEs).

1.2.1 The Service-Oriented Enterprise

The service-oriented enterprise promises to significantly improve corporate agility, speed time-to-market for new products and services, reduce IT costs, and improve operational efficiency [79]. In line with the network era characteristics mentioned in section 1.1.3, a service-oriented enterprise exploits a service network. This service network entails the concept of interface ubiquity, enabling equal base communication mechanisms to be used — within applications, as well as between applications, as well as between external partners [52]. According to Liu and Bouguettaya, SOEs are virtual, adaptive, extensible, market-driven and on-demand, Web service-based enterprises; operating in an extremely dynamic environment, outsourcing their functionalities via (third-party) Web services [71]. As described by Newcomer and Lomow [79], several industry trends are converging to drive fundamental IT changes around the concepts and implementation of service orientation. Four key technologies with respect to this convergence, can be identified.

- Extensible Markup Language (XML)
- Web services
- Service-Oriented Architecture (SOA)
- Business Process Management (BPM)

All technologies are described in more detail below. At the end of this section, figure 1.2 shows how the service-related concepts, techniques and parties described in this introduction are interrelated.

1.2.1.1 Extensible Markup Language

The Extensible Markup Language (XML) was developed by an XML Working Group (originally known as the SGML Editorial Review Board) formed under the auspices of the World Wide Web Consortium (W3C) in 1996. According to the definition of this consortium, XML describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them [14]. XML is an application profile or restricted form of SGML, the Standard Generalized Markup Language [41]. By construction, XML documents are conforming SGML documents.

Following the description of the W3C, XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constraints on the storage layout and logical structure. Assuming the XML Working Group has reached its design goals [13], XML supports a wide variety of applications and is straightforwardly usable over the Internet. Furthermore, XML is designed to be formal and concise.

Within a service-oriented enterprise, XML provides standard data types and structures, for example for defining business documents and exchanging business information between applications or external systems and partners. While XML is a mainstream language and forms a substantial part of many applications and processes nowadays [21; 29], it is also subject to criticism [28].

1.2.1.2 Web Services

What is a Web Service?

Many definitions of the concept 'Web service' exist, varying in specificity. Some define the (underlying) concept of a Web service as '*a set of self-contained, modular applications that can be published, discovered and invoked over a network*' [51].

According to the W3C — specifically, the Web Services Architecture Working Group within the W3C — a 'Web service' is '*a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.*' [11].

A more technical definition can be obtained from the Webopedia Computer Dictionary: '*The term Web services describes a standardized way of integrating Web-based applications using the XML, SOAP, WSDL and UDDI open standards over an Internet protocol backbone. XML is used to tag the data, SOAP is used to transfer the data, WSDL is used for describing the services available and UDDI is used for listing what services are available.*' [111].

As becomes clear observing the various definitions stated above, Web services can be seen as modular software applications, orchestrated in a network, to support machine-to-machine interaction over this network (or backbone). Web services — or more specifically, their interfaces — have a public description and communicate with other Web services or systems by sending messages. XML, SOAP, WSDL and UDDI are important protocols and techniques with respect to the organization of Web services. These techniques and protocols are described in more detail below — except for XML, which is described in section 1.2.1.1.

SOAP

To allow different services to interact with each other, a communication mechanism is needed. The specification of this mechanism involves three aspects: a common data format for the messages being exchanged, a convention for supporting specific forms of interaction (such as messaging or RPC) and a set of bindings for mapping messages into a transport protocol [3].

Concerning Web services, interactions are based on the Simple Object Access Protocol (SOAP). W3C started the development of SOAP in 1999. Since then, three major versions have been recommended by the consortium. At the time of writing, SOAP 1.2 is a recommendation [45].

By using SOAP, Web services can exchange data messages in a standardized way — service *requesters* (software modules that invoke a service implemented by a service provider to accomplish a task [79]) can invoke Web services of service *providers* (software modules that implement a service according to a service contract [79]) by exchanging SOAP messages. A service invocation is translated to an XML message (at the requester), the message is exchanged between two Web services, and finally the XML message is turned back into an actual service invocation (at the provider).

When a Web service receives a message from a service requester and then sends that message to a service provider, this Web service has the role of a *service broker*. Note that service providers can also be service requesters — within multi-tier architectures, for example. A service broker by definition plays both roles. Finally, the 'service locator' can be identified. A service locator is a specific kind of service provider that acts as a registry and allows for the lookup of service provider interfaces and service locations[1] (see figure 1.1). SOAP messages can be exchanged

[1] Service locators are often implemented as an UDDI registry. See section 'Universal Description, Discovery and Integration' (UDDI).

on top of HTTP(S), SMTP or other transport protocols. In practice, SOAP is most used over HTTP, because it allows easier communication behind proxies and firewalls than other transport protocols[2].

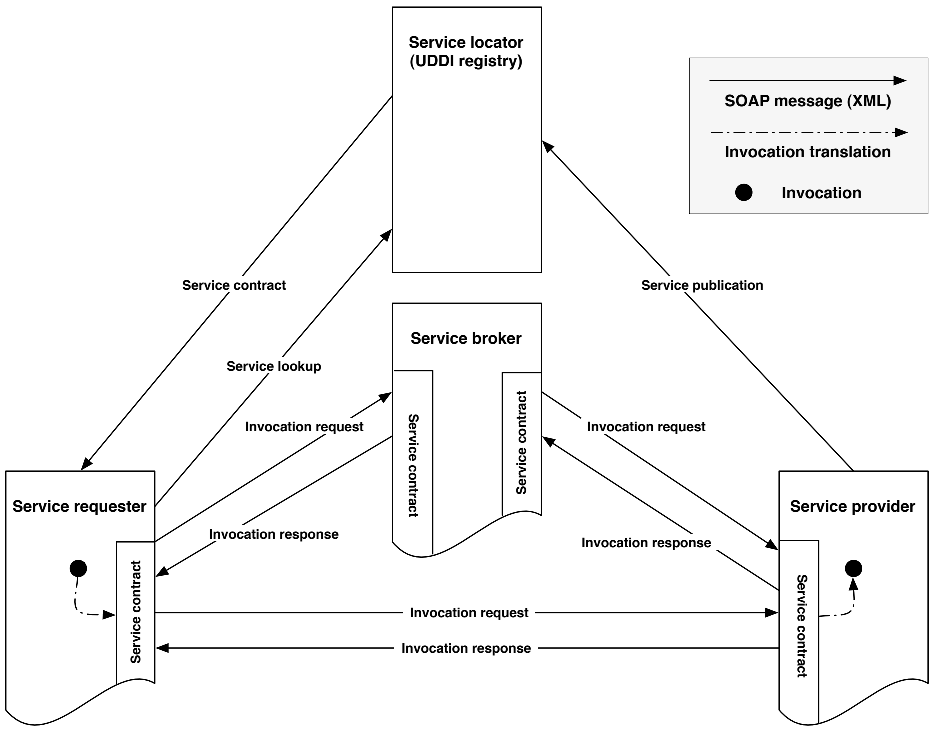


Figure 1.1: *Various roles within the context of SOAP*

Web Services Description Language

Web services have an interface describing the methods that are supported by the service, where each method could take one message as input and return another as output (also see the previous paragraph). To identify and interact with (the methods of a) Web service, the Web service (and specifically, its interface) has to be described. Software interfaces are described using an *Interface Description Language* (IDL). In Web services, this role is played by the Web Services Description Language (WSDL). This language is also developed by the W3C. According to the W3C, '*a WSDL 2.0 service description indicates how potential clients are intended to interact with the described service. It represents an assertion that the described service fully implements and conforms to what the WSDL 2.0 document describes. A WSDL 2.0 interface describes potential interactions with a Web service, not required interactions. The declaration of an operation in a WSDL 2.0 interface is not an assertion that the interaction described by the operation must occur. Rather it is an assertion that if such an interaction is (somehow) initiated, then the declared operation describes how that interaction is intended to occur*' [22]. At the time of writing, WSDL version 2.0 is a recommendation [22], being a major revision of the WSDL 1.2 working draft.

[2] Firewalls and proxies often do not block HTTP traffic, because most Internet browsers also generate this traffic. Blocking HTTP traffic would cause Internet browsers stop functioning.

Universal Description, Discovery and Integration

In order to use Web services more pervasively and on a more global scale, a standard way for publishing and locating services is needed. With this standardized manner, service requesters can look for services fitting their interests by reading the services' properties. To do so, the Web services registry (repository) has to be standardized. A project wherein such a standardization is developed, is the Universal Description, Discovery and Integration (UDDI) project. Currently, this project is continued by the Organization for the Advancement of Structured Information Standards (OASIS) consortium. According to this consortium, '*the focus of Universal Description Discovery and Integration (UDDI) is the definition of a set of services supporting the description and discovery of (1) businesses, organizations, and other Web services providers, (2) the Web services they make available, and (3) the technical interfaces which may be used to access these services. Based on a common set of industry standards, including HTTP, XML, XML Schema, and SOAP, UDDI provides an interoperable, foundational infrastructure for a Web services-based software environment for both publicly available services and services only exposed internally within an organization.*' [24]. In other words, UDDI is a specification of a framework for describing and discovering Web services. An important part of UDDI is a *business registry*, which in essence is a naming and directory service. At the time of writing, UDDI version 3.0.2 is the most recent version — incorporating errata to the v3 specification. UDDI v3 became an OASIS standard in February, 2005.

While the concept of a Web service entails many more aspects than these covered by XML, SOAP, WSDL and UDDI, it is important to note that Web services represent the first jointly coordinated and arranged effort to standardize interactions between information systems, in particular Web services. The protocols, languages and standards discussed before are developed in parallel and are strongly interrelated. Specific features in one of the standard specifications influence the way other protocols, languages and standards are used.

1.2.1.3 Service-Oriented Architecture

Burbeck [19] was one of the first using the term 'service-oriented architecture' (SOA). The term can be interpreted as an architectural style that guides all aspects of creating and using business processes, packaged as services, throughout their life cycle, as well as defining and provisioning the IT infrastructure that allows different applications to exchange data and participate in business processes, regardless of the operating systems and programming languages underlying these applications [79].

The main goal of SOA is transparent, flexible and dynamic interaction of services and their clients over multiple interconnected domains, while the benefits of SOA include increased efficiency through task outsourcing and component reuse, easier integration, increased flexibility and agility at business and IT level and on-demand interconnection with business partners [76].

SOA supports the connection of various (existing) applications and data sharing — the data extensively being encapsulated and / or described by XML. Applications are structured as a collection of services — similar to the services types discussed in section 1.2.1.2 — which can be used by different groups of people inside or outside the company. SOA services are loosely coupled[3], in contrast to the functions a linker binds together to form an executable, a dynamically linked library (DLL), or an assembly.

A service-oriented architecture is supported by the Web services platform (in short, consisting of a Web services registry, service providers and service requesters) in combination with an *Enterprise Service Bus* (ESB) to connect the services and the various parties. Other characteristics of SOAs include discovery mechanisms to register connected (available) services, and service-level Quality of Service (QoS) management. One of the main goals of QoS is to optimize system resources

[3] Services can be developed and evolved independently, while being dynamically discovered and utilized.

and activate computing mechanisms, in order to satisfy the QoS requirements that are set for applications based on a specific SOA [109].

One should notice that there is a difference between (1) an application that uses Web services and (2) an application based on a service-oriented architecture. An existing application can be extended with specific service-based functionality in order to meet project requirements, for example. However, this does not automatically imply that a service-oriented architecture is used or implemented. According to Erl [35], Web services do represent an implementation of an SOA when used to establish a concept of encapsulating application logic within services that interact via a common communications protocol.

As one may expect based on the developments in the various eras of Information Technology described in section 1.1 and specifically, the characteristics of the service-oriented enterprise discussed in section 1.2.1, SOAs are adapted and implemented more and more. This expectation is confirmed by research [53; 86].

Specifically considering software vendors in service-oriented environments, the role of such types of companies is twofold. First, software vendors maintain business-to-consumer (B2C) relationships with both (service-oriented) enterprises, as well as end-users. Both types of 'consumers' may have licensed one or more services developed by one or more software vendors. Second, software vendors in a service-oriented environment maintain business-to-business (B2B) relationships with other software vendors (which may also operate in a service-oriented environment), for example by licensing software of these vendors (illustrated by examples in section 3.4.2). As also depicted in figure 1.2, service-oriented enterprises may maintain a B2B relationship with other SOEs, too. Communication infrastructures may differ.

1.2.1.4 Business Process Management

Business Process Management (BPM) is defined as '*the support of business processes using methods, techniques, and software to design, enact, control, and analyze operational processes involving humans, organizations, applications, documents and other sources of information*' [105].

BPM and SOA, both being important concepts within a Service-Oriented Enterprise, have developed independently. However, a relationship between the two is acknowledged. As described earlier in this introduction, SOA promotes and supports the creation and use of loosely coupled (business) services. Woodley and Gagnon [113] find that BPM can help improve the execution of SOA projects because BPM leverages such services, tying them together to solve and streamline broad business challenges. As a backbone for SOA components, BPM can integrate important functionalities to extend the value of the SOA investment. Similarly, BPM can provide a platform for SOA service management. Process-centric tools can be used to rapidly develop, publish and orchestrate services, and to ensure greater coherence between a SOA solution and its overarching processes. According to Newcomer and Lomow [79], a SOA provides the ideal level of abstraction for defining reusable business functionality, completely encapsulating underlying applications from BPM systems.

1.2.2 Service-Oriented Development

The trends identified at the beginning of this introduction and discussed in sections 1.2 and 1.3, challenge software vendors to adopt the paradigm of service-oriented development (SOD) based on Web services. SOD should be seen as complementary to the well-known and well-adopted Object-Oriented Development[4] (OOD) approach. Newcomer and Lomow [79] describe four SOD characteristics, which are listed below.

[4] Also known as Object-Oriented Design.

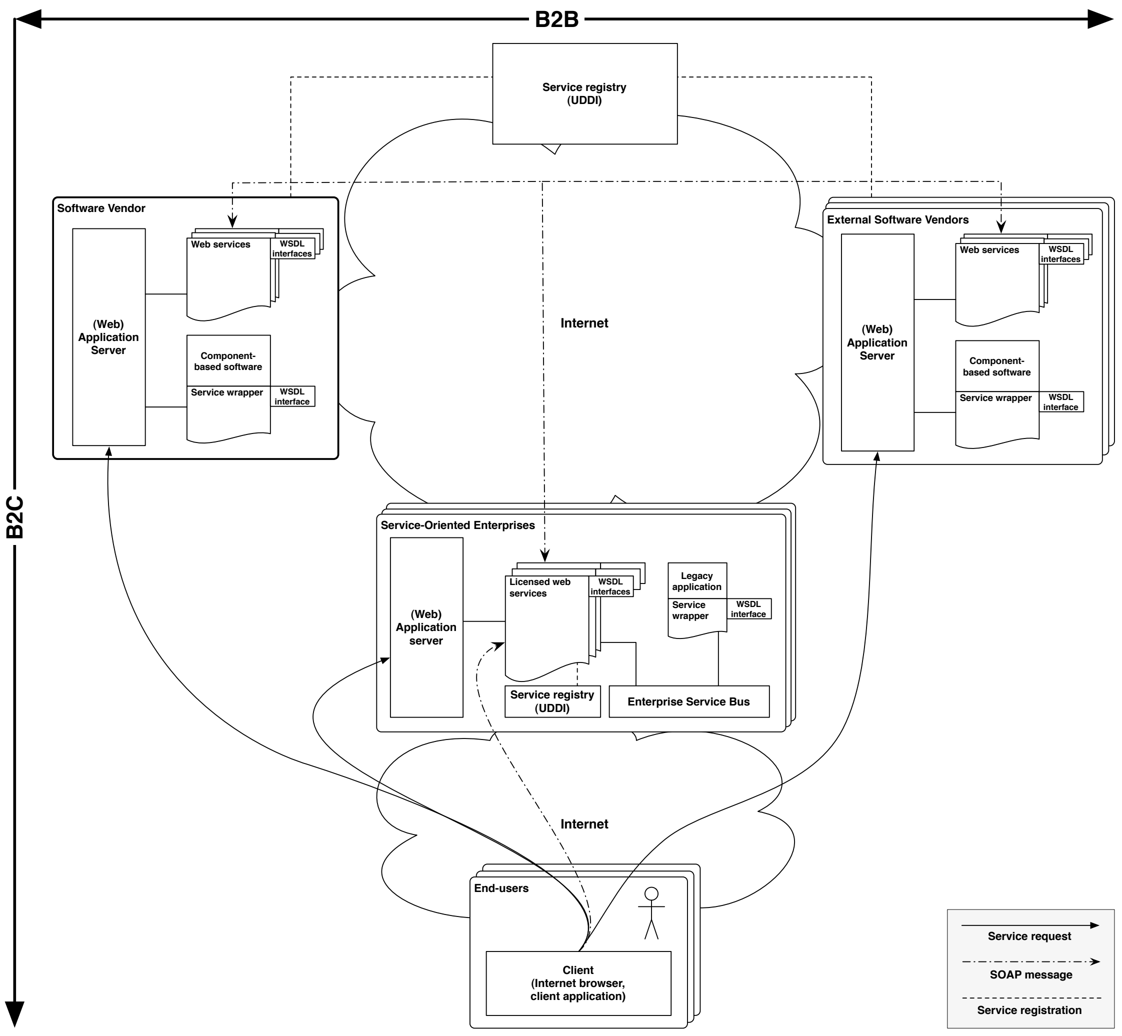


Figure 1.2: *An overview of all service-related concepts, techniques and parties described in this introduction. Both software vendor organizations and service-oriented enterprises maintain business-to-consumer and business-to-business relationships*

Reuse The ability to develop services that are reusable in multiple applications or contexts.

Efficiency The ability to quickly and easily develop new services and applications by combining new and old services, along with the ability to focus on data sharing (instead of sharing the underneath implementation).

Loose technology coupling The ability to model services independently of their execution environment and design messages that can be sent to any service.

Division of responsibility The ability to more easily allow business people to concentrate on business issues, technical people to concentrate on technology issues, and for both groups to collaborate using the service contract.

Looking at the characteristics above, one may find them quite similar to the often-referenced benefits of (component-based) OOD [10; 55; 68; 88]. So while the names (OOD versus SOD) and subjects (components or objects versus services) of both development types is different, the *characteristics* of both development approaches are quite similar.

A concept that is closely related to SOD and the trends described in section 1.3, is 'Software as a Service' (SaaS). The term is mostly associated with an online article of O'Reilly [83], later published in [39]. Generally, SaaS is more associated with business software than with consumer software. Consumer-oriented web-native software (including a number of the applications referenced in section 1.3, for example) is generally known as — the frequently misused, misinterpreted and bloated [5; 67] term — Web 2.0 and, to a lesser extent, as SaaS.

When a software vendor develops its software 'as a Service', the company develops web applications that are hosted and operated at the software vendor (or a third party): SaaS shifts the burden of running and maintaining hardware and software to the vendor. Furthermore, customers do not pay for owning the software itself, but rather for using it. Software application delivery is closer to a one-to-many model (single instance, multi-tenant architecture) than to a one-to-one model, including corresponding architecture, pricing, partnering, and management characteristics. Often, software features are updated in a centralized way, obviating the need for separate downloadable patches and upgrades. Recently, Microsoft launched Microsoft Online Services[5], a strategy to publish a part of its software as an online service, therewith adopting the SaaS concept: customers take a subscription on (service-based) software, running on their own hardware or on hardware of the software vendor.

Under most conditions, software vendors will invest more applying a SaaS-compliant licensing model, than when applying a perpetual licensing model — leading to higher software quality, higher profits and higher social welfare [23].

1.3 The Desktop Online

Another observable trend that is contributing to the rise and adoption of (web) services and service-oriented development, is the increasingly adoption and maturation of the Rich Internet Application (RIA) — Web applications that have the features and functionality of traditional desktop applications.

RIAs become more and more common. End-users are connected to the Internet ('on-line') more frequently, and they are connected for a longer time continuously [50; 99]. As a consequence, they are less (or even not at all) restricted to an off-line working environment. Examples of well-known RIAs are Google Docs[6], Google Maps[7] and Gmail[8].

[5] http://www.microsoft.com/online/
[6] http://docs.google.com
[7] http://maps.google.com
[8] http://gmail.com

With the maturation of RIAs, the boundaries between desktop and web applications begin to blur. Examples of development that even more eliminate the differences between both type of applications are Adobe Integrated Runtime (AIR), Google Gears and Mozilla Prism. Adobe AIR is a platform-independent runtime environment for building RIAs[9], using Flash, Flex, HTML and Ajax, that can be deployed as a desktop application; Google Gears, a technology developed by Google that enables offline Web applications, can be seen as an similar concept[10]. Mozilla Prism[11] is also comparable to AIR.

An important difference between a 'traditional' RIA and a RIAs developed with a technology similar to Adobe AIR is that the latter type of applications do not need a browser to run and often can be executed in an off-line environment, too. Considering the key process areas of product software, Rich Internet Applications have advantages compared to traditional desktop applications. For example, the release, delivery and deployment phases of (fully or partly) online RIAs, are shorter (or even completely absent) compared to the phases of desktop applications that are executed offline. Similar to what is discussed in section 1.2.2, changes and updates to the code or the interface of a RIA are propagated to the end-user immediately, i.e., the end-user does not have to install updates or patches to be able to use the latest version of the software.

1.4 (Continuous) Customer Configuration Updating

A third observation, supported by research conclusions, is that nowadays still only few software vendors explicitly plan releases and only very few vendors make use of usage reports. The Dutch National Product Release Benchmark Survey 2007 [59] concludes that only 40% of the questioned product software companies utilize a release planning with various release moments planned. Furthermore, only 28% of the software vendors sell software that automatically composes usage reports.

The concept of Customer Configuration Updating (CCU) was created to improve this situation, especially for software vendors developing component-based software. CCU is defined as the release, delivery, deployment and usage and activation processes of product software [57], particularly fitting and resembling the software life cycle processes [42]. The CCU model, as depicted in figure 1.3, displays the states a customer can move through after a product or update release (of component-based software) on the right side [57]. On the left side, the organizational structures that facilitate interaction are displayed.

As software products and updates are changed and released more often, effort is saved by automating steps in the CCU process — automating steps in this process enables a software vendor to become more responsive to changes in customer and market demands [60].

One of the tools that potentially automates CCU and that was developed to serve this purpose is Pheme [58]. Pheme is an infrastructure that enables a software vendor to communicate about software products with end-users and enables system administrators to perform remote deployment, policy propagation, and policy adjustment. Moreover, Pheme potentially enables software vendors to publish software knowledge in the form of licenses, software updates, software content and software news, and it potentially enables end-users to send knowledge in the form of usage and error feedback. While comparable tools are available, most of these tools focus on one particular aspect of the customer configuration updating process. Pheme is designed to automate all CCU key policies, mainly focusing on component-based software products. A basic prototype of Pheme has been developed, mainly covering the first three processes of CCU (release, delivery and deployment).

[9] http://www.adobe.com/products/air/
[10] http://gears.google.com/
[11] http://wiki.mozilla.org/Prism

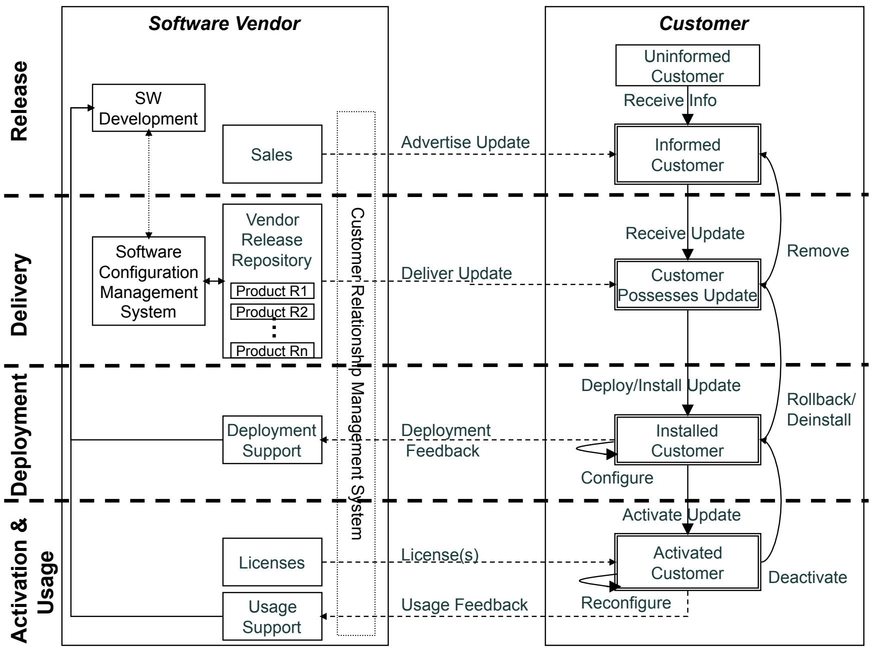


Figure 1.3: *Customer Configuration Updating (CCU) model [57]*

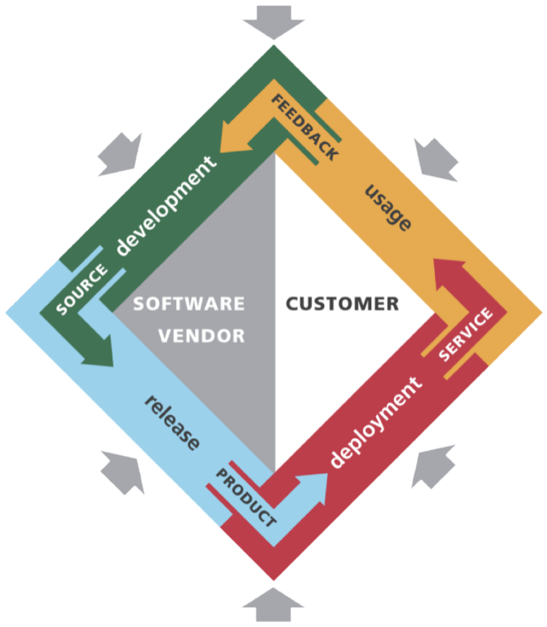


Figure 1.4: *Continuous Customer Configuration Updating (C-CCU) model [60]*

As described by Jansen [56], automating steps in the CCU process ultimately contributes to *Continuous* Customer Configuration Updating (C-CCU), enabling a software vendor to become more responsive to changes in customer and market demands. Jansen [56] defines *Continuous Customer Configuration Updating* (C-CCU) as follows.

Continuous Customer Configuration Updating Being able to continuously provide any stakeholder of a software product with any release of the software, at different levels of quality.

In [60], Jansen shows that implementing C-CCU in a software vendor's organization reduces overhead from the development, release, delivery, deployment, activation and usage processes, enabling a software vendor's organization to become more responsive to change (see figure 1.4). Similar conclusions can be found in [9]. However, as with CCU, research mainly focused on applying and implementing the concept with respect to component-based software.

1.5 About Stabiplan B.V.

This section describes the software vendor, Stabiplan B.V., at which this thesis was carried out. First, a company profile is provided. Next, the vendor is identified in terms of the software products and services it develops. Finally, the vendor's customers, organization structure and software development processes are identified.

1.5.1 Company Profile

Stabiplan [98] develops and distributes CAD software products for the building services industry, creating designs in the '.dwg' file format. Stabiplans market leading product, StabiCAD, is now used by more than 7000 draftsmen every day in the Netherlands and Belgium.

Founded in 1990, Stabiplan has set the basis for CAD software for contractors in the Netherlands and Belgium. StabiCAD is used daily by most companies in the Dutch speaking building services industry. With over 3800 customers and more than 8000 licenses sold, Stabiplan is market leader in its segment.

Stabiplan approximately employs 100 employees. The software development activities are mainly performed in the Netherlands. Since 2006, a separate software development team is located in Romania. In that year, Stabiplan Belgium was launched, too. Stabiplan is one of the largest AutoCAD dealers in the Netherlands and a long-standing partner of Autodesk [6], the vendor of AutoCAD.

Recently, Stabiplan made its StabiCAD 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.

Stabiplan is the vendor of a number of software products and services. In the next sections, these products and services are described.

1.5.2 Software Products

1.5.2.1 StabiCAD V, LT

Before the release of StabiCAD 8, StabiCAD V was Stabiplan's flagship software product. With the software, draftsmen are able to draw sewers, air tubes, pipelines and cable routes. Furthermore, complex surface, volume and resource calculations can be performed. The package includes symbol libraries to depict all kinds of radiators, pumps, valves, ducts and other technical or mechanical products, including corresponding parameters and technical information.

StabiCAD V can be run with Autodesk AutoCAD [6] or IntelliCAD [54] and has a modular design, which allows customers to set up their own module configuration. Each module is related to a specific market branch. Examples of modules are 'Ventilation and Air conditioning', 'Electrical Installation Engineering', 'Gas Pipe Calculation' and 'Fire Safety'.

While StabiCAD V provides full functionality for full-time draftsmen, StabiCAD LT, the 'light' edition of StabiCAD, is positioned as a cost-efficient solution for the infrequent CAD draftsman. Compared to StabiCAD V, StabiCAD LT lacks calculation- and 3D viewing capabilities. Both StabiCAD V and StabiCAD LT are component-based, *off-line* software and run with most recent versions of Microsoft Windows.

1.5.2.2 StabiCAD 8

After having developed StabiCAD V for around five years, on the 8th of August 2008, Stabiplan released the next version of StabiCAD: StabiCAD 8. Compared to StabiCAD V and LT, the improvements of this new version are mainly in the area of the user interface consistency, as well as in the areas of drawing and calculation intelligence and database uniformity. With StabiCAD 8, the 'light' and 'full' editions of the StabiCAD software are integrated in one software package, available in two editions: 'StabiCAD 8 LT' and 'StabiCAD 8 PRO'. Similar to StabiCAD V, StabiCAD 8 runs with both AutoCAD and IntelliCAD kernels.

1.5.2.3 StabiBASE

StabiBASE can be seen as the StabiCAD control centre. This software module is a solution for document, database and project administration management, project settings configuration and symbols overview generation. StabiBASE organizes the drawings and other documents of a draftsmen by project, object or engineering application. Draftsmen often utilize this software as the starting point for their drawing activities.

1.5.3 Software Services

1.5.3.1 StabiBASE Web

Recognizing the shift from mostly component-based desktop applications towards service-based online applications described earlier, Stabiplan realized that an online, service-based alternative of its StabiCAD control centre should be developed. Enabling draftsmen and project leaders to access, preview and download their drawings and project documents online via a web browser, StabiBASE Web is the first of a new generation of service-based applications at Stabiplan.

1.5.3.2 CADsymbols.nl

Manufacturers of certified branch materials for engineering (boilers, pumps, pipelines, etc.) provided information and drawings of their products to Stabiplan for incorporation within various StabiCAD modules. However, new versions of Stabiplan software and new versions of symbols are not always released synchronously. To enable draftsmen to use new or updated drawing symbols in their drawings, Stabiplan launched CADsymbols.nl. At this website, a large collection of CAD symbols is presented. Manufacturers can register as a symbol supplier, draftsmen may download CAD symbols free of charge and incorporate these in their drawings.

1.5.4 Customers

Until now, Stabiplan roughly sold 8000 licenses to more than 3800 customers. At time of writing, the company's customer base mainly consists of Dutch and Belgian companies operating in the building services and management sector. Whereas StabiCAD V only was released in a Dutch version, StabiCAD 8 is released in Dutch, English, French and Romanian localizations. The somewhat larger customers of Stabiplan include Wolter & Dros, Stork N.V. and Imtech N.V. The software vendor distinguishes two types of users for its software. Draftsmen spending more than 20 hours a week are advised to use StabiCAD 8 PRO, otherwise they are advised to license StabiCAD 8 LT.

1.5.5 Organization Structure

Currently, Stabiplan approximately employs 100 people, divided over the Netherlands, Belgium and Romania. Most employees (80 people) are located in Stabiplan's head office in Bodegraven, the Netherlands. Apart from the head office, a separate team of around 15 people is located in Romania. Furthermore, Stabiplan Belgium consists of 3 people. Because of its international activities, the Stabiplan holding now exists of four companies: Stabiplan B.V. (figure 1.5), Stabiplan Romania, Stabiplan Belgium and Stabiplan International B.V. (figure 1.6).

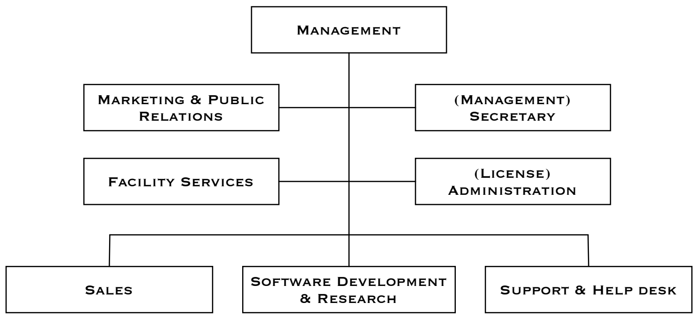


Figure 1.5: *Organization chart of Stabiplan B.V.*

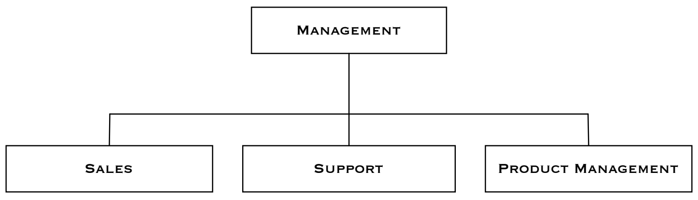


Figure 1.6: *Organization chart of Stabiplan International B.V.*

1.5.6 Software Development

This section describes the situation at Stabiplan based on facts that were gathered during this thesis research, making use of the research methods presented in chapter 2.

1.5.6.1 StabiCAD V, LT

Since practically all company resources were dedicated to the development and release of StabiCAD 8, there was almost no StabiCAD V- and LT-related development. Only small maintenance updates for StabiCAD V and LT were released.

1.5.6.2 StabiCAD 8, StabiBASE (Web)

Stabiplan's development department is split into a number of teams. Each team has a project leader and is responsible for its corresponding modules and calculations. Towards the release of StabiCAD 8, numerous pre-releases have been defined, each release representing a major milestone, related to a set of new features and functionality. Project leaders met very frequently with their team to discuss and review the progress of their work. With respect to the communication with the separate development team in Romania, Skype conversations and conferences are initiated on a regular basis.

Functional requirements elicitation and prioritization concerning the development of the software package are performed by a product management team, consisting of the CEO, national and international project managers, as well as a number of project leaders.

Another group of people was responsible for testing all new calculations, modules, interface elements and other functionality introduced in StabiCAD 8. Other people were dedicated to the localization, graphical aspects, installation procedure and compatibility of the software package. Major results and milestones concerning the development of both StabiCAD and StabiBASE were presented during development meetings, possibly combined with a Skype session.

1.5.6.3 CADsymbols.nl

Concerning the development of CADsymbols.nl, hardly any development was done. Again, mainly because of the focus on the release of StabiCAD 8, practically no effort was put into the development of this website. That is, no new features were added, although the provision and registration of new symbols and engineering materials information to the website continued uninterruptedly.

1.6 Thesis Structure

In the next chapter, this thesis research approach is detailed. Chapters 3 and 4 present the research outputs of this thesis. Chapter 4 specifically introduces the developed software prototype (Nuntia) and specifies it on functional and technical level. Chapter 4 also includes the validation of the prototype with respect to its performance. Chapter 5 presents the results, conclusions and discussion of this thesis research. This chapter also positions the contribution of this thesis research in the context of related research and other work. Finally, in chapter 6, future research is discussed.

Chapter 2

Research Approach

This chapter details this thesis research approach. First, the research problem and research questions are defined. Secondly, research steps and outputs are specified. Thirdly, the case study that is carried out as part of the research is explicated. Finally, the research validation process is detailed.

2.1 Research Problem

In the introduction of this thesis, four trends concerning software vendors and software development were observed.

- Nowadays, software vendors need to be very agile organizations, responsive to changes in customer and market demands, while developing software at high speed, low cost and according to high quality standards.
- Concepts like the Service-Oriented Architecture, Software as a Service and Web service composition are rising and increasingly accepted and implemented, ultimately within a Service-Oriented Enterprise.
- Software vendors shift from the Object-Oriented Development paradigm to the Service-Oriented Development paradigm, focusing on the development of Web services and online, Rich Internet Applications instead of offline, component-based desktop applications.
- Still, only few software vendors explicitly plan releases or make use of usage reports nowadays, and many product software vendors deliver their product software on a 'manual pull' basis.

Furthermore, apart from the results already discussed in section 1.4, the Dutch National Product Release Benchmark Survey 2007 [59] presents another set of interesting findings.

- Almost two third of the participating software vendors indicate that their customers report bugs via telephone or e-mail; 6% of the companies indicate that bugs are registered and collected via automatically composed bug reports.
- 30% of the software vendor's software automatically composes and sends bug reports.
- 95% of the software vendors indicate using licenses, while 18% indicates that licenses are automatically generated from sales contracts.
- 52% of the participants facilitates customer-specific adjustments or features to their software.

Given the trends and findings above, a solution that enables to apply C-CCU specifically to service-based software (and therewith specifically enabling a software vendor developing service-based software to become more responsive to change), is needed. In this thesis, a solution that fulfills this need is presented. As is explicated and motivated in chapter 3, the concept of C-CCU as defined by Jansen [56], is only partly applicable to service-based software.

Main Research Question

RQ *How can Continuous Customer Configuration Updating be applied to service-based software and Web services?*

Sub Questions

SQ1 *What are differences and similarities between software components and services with respect to C-CCU?*

SQ2 *What are measurable critical success factors of C-CCU applied to service-based software and Web services?*

SQ3 *When is C-CCU, applied to service-based software and Web services, implemented successfully?*

The solution presented is threefold: it consists of (1) a comparison of the life cycles of component-based and service-based software to analyze to which extent C-CCU can be applied to service-based software; (2) a concept that describes this application; (3) a software prototype that provides a concrete implementation of this concept.

2.2 Introduction to Design Research

The research that has been done in the context of this thesis can be characterized as design research. Design research is an information technology research methodology which offers guidelines for evaluation and iteration within research projects [48]. The methodology is an outcome-based information technology research methodology, which offers specific guidelines for evaluation and iteration within research projects [103]. Within this methodology, the focus is on the development and performance of (designed) artifacts with the explicit intention of improving the functional performance of the artifact.

Design research is characterized by five process steps, where each process step is related to a specific output type. Furthermore, the methodology defines various knowledge flows that 'flow' from specific process steps to process steps preceding these steps. In the next section, this thesis research is described in terms of the design research methodology. See figure 2.1.

2.3 Design Research in the Context of This Thesis

As visualized by figure 2.1, the design research methodology defines five process steps. In this section, each process step will be linked to one or more parts of this thesis.

2.3.1 Awareness of the Problem

The first process step is titled 'Awareness of the Problem'. In the introduction of this thesis (chapter 1), the problem description of this thesis research is described, and summarized by a

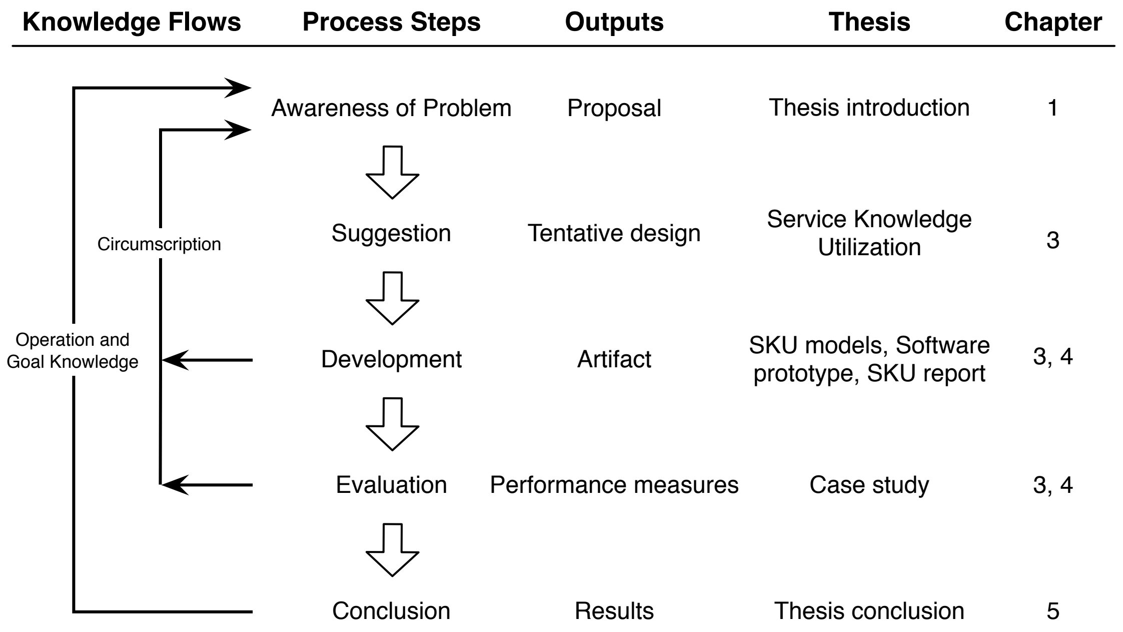


Figure 2.1: *The Design Research Methodology in the context of this thesis (adapted from [103])*

main research question. The problem consists of three elements. The first element is the shift from component-based software to service-based software software vendors are making. The second element is formed by the application of Jansen's C-CCU model (and its advantages) to service-based software. Finally, the third element of the problem is the status of automatic performance, usage and feedback logging, monitoring and reporting at software vendors in general.

2.3.2 Suggestion

A solution to the problem defined in section 2.3.1 is suggested by this thesis research in the form of a concept called 'service knowledge utilization' (SKU). The SKU concept, which can be seen as 'tentative design' output in the design research methodology, is designed to address each of the elements the problem consists of. In chapter 3, the concept of service knowledge utilization is presented.

2.3.3 Development

Part of the SKU concept is a set of artifacts. Each artifact is tightly related to the concept of SKU and is developed within this thesis research. Furthermore, as visualized by figure 2.1, each artifact is an output of the 'Development' process step of the design research methodology. Three SKU models, a software prototype and the SKU report are included in this set of artifacts. The software prototype is called Nuntia and is the main artifact of the design research performed. Nuntia is presented in chapter 4. The SKU models and the SKU report are described in chapter 3.

2.3.4 Evaluation

With respect to this thesis research, the ‘Evaluation’ process step of the design research methodology concerns the validation of the outputs of the ‘Development’ process step. This validation occurs in order to investigate to which extent the developed artifacts (as part of the suggested solution to the problem perceived) contribute to the solution proposed, and (therewith) to obtain a more profound insight in and understanding of the problem.

The evaluation of the artifacts is done by means of a case study, which is described in section 2.4 of this chapter. The performance measures that are mentioned as outputs by figure 2.1 are presented in chapter 4.

2.3.5 Conclusion

‘Conclusion’ is the last process step of the design research methodology. Having evaluated the research outputs, the last process step entails the formulation of research conclusions. As a consequence, final results and conclusions form the outputs of this process step. Again, as depicted in figure 2.1, both the results and conclusions contribute to the awareness of the problem. Specifically, more knowledge concerning the research operation techniques and research goal are gathered during the ‘Conclusion’ process step. The results and conclusions of this thesis research are presented in chapter 5.

2.4 Case Study

This section describes the case study research approach. First, an overview of the various case study phases is presented. Next, case study techniques that were used to describe the case study company (Stabiplan B.V.) and gather facts on which the results of the case study are based, are detailed and research hypotheses are presented.

2.4.1 Overview

As mentioned, the goal of this case study is to evaluate the software prototype and a number of SKU concepts (the concepts, or artifacts are listed in figure 2.1). An overview of the various phases of this case study with respect to the software prototype is depicted in figure 2.2. Similar phases apply to other artifacts.

First, the current situation at the software vendor in the context of SKU was identified (see section 3.8.3). Next, the software prototype was developed and implemented at Stabiplan. Section **??** presents details of this phase. The last phase of the case study is related to the validation of the software prototype and SKU concepts. Different research methods are utilized to validate these research outputs. Which research methods are used to validate which research outputs is detailed in section 2.5; validation results are presented in section 5.1. The research questions of this research defined in the introduction of this thesis (section 2.1) are partly answered based on results of the case study.

The next section described the research methods that were utilized in the identification phase of the case study. Section 2.4.3 lists three hypotheses which illustrate potential effects of a successfully operating SKU implementation. The validation process of this thesis research is described in section 2.5.

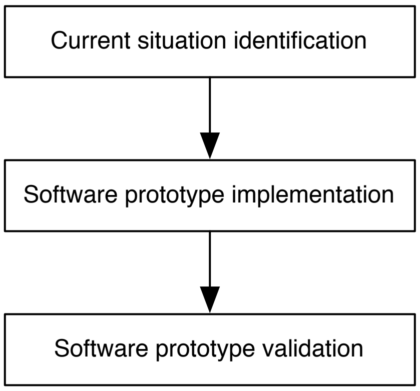

Current situation identification
Software prototype implementation
Software prototype validation

Figure 2.2: *Case study phases*

2.4.2 Research Methods

In order to gather facts on which the results of the case study are based, various case study research methods have been used. Yin [116] has defined six sources of case study evidence: documentation, archival records, interviews, direct observations, participant-observation and physical artifacts. Four sources are utilized within the context of this thesis research.

Direct Observations Various 'direct observations' were made during the presence at Stabiplan. First, a company introduction tour was done, giving an impression of the structure and departments of the company. Furthermore, this tour aided with indicating the key people within the organization because during this introduction, a number of short introductory conversations were held. Second, a number of development meetings were attended. During these meetings, software developers and project leaders presented the progress of their work or projects and presentations about new project management methods and programming language features were given. Because of the substantial amount of 'around the coffee machine' chatter during these meetings, the meetings helped in getting a realistic view of the company, its culture and its employees. Employees were very engaged and motivated to deliver on-time, in order to meet the release date of StabiCAD 8.

Documentation Stabiplan provided documents such as software architecture specifications, software manuals, process descriptions, meeting agendas and various memos. During the case study, these documents have been studied, partly as part of the preparation on the case study interviews (see below).

Software A training session was attended in order to get familiar with the Stabiplan software, its end-users and Stabiplan's training sessions. Furthermore, during the presence at Stabiplan, the software being developed — mainly StabiBASE Web — was observed on both code and usage level. Being positioned near the main Skype PC of the development department of Stabiplan, many programming problems, reviews and solutions were observed[1].

Interviews Yin [116] defines three types of interviews: open-ended interviews, focused interviews and (semi-) structured or survey. In the context of the case study of this thesis, twenty *semi-structured* interviews have been conducted. Moreover, a survey has been designed specifically for each of the departments within the organization of Stabiplan employing people that were invited to the case study interview. This survey was sent to the interviewees, and had

[1] As indicated in section 1.5.6, Skype was used to communicate with the development team in Romania.

to be returned filled-in, before the start of the interview. The filled-in surveys were also studied before the start of the interviews and frequently led to new interview questions. During the interviews, which were always one-on-one, the surveys served as a guide to make sure that all relevant aspects of the situation at the company with respect to SKU would be addressed. Mostly, interviewees were asked to elaborate on their survey answers and describe the 'current' situation at their department. Eventually, interviewees were asked to share their understanding or opinion concerning a possible future situation and the consequences of such a situation.

2.4.3 Hypotheses

The SKU characterization model presented in chapter 3 defines four types of software vendors in terms of SKU level, flexibility and responsiveness. After having implemented the software prototype successfully (second phase of the case study, see figure 2.2), it is expected that three potential effects of this implementation can be observed.

Decrease of Time To Market (TTM) It is expected that, utilizing a complete SKU implementation, 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 consequently 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, changes in customer and market demands potentially are expected and incorporated in the vendor's projects at an earlier stage, in a shorter amount of time. As motivated in chapter 3, a short time to market contributes to an increase of the vendor's flexibility.

Responsiveness Increase The software vendor's responsiveness, 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 could be defined that have to be executed when specific 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.

SKU Level Increase By implementing the software prototype (which provides a service knowledge utilization implementation for software vendors developing service-based software), it is expected that Stabiplan's SKU level will increase. As detailed in section 3.8.3, Stabiplan currently has no objective data on the usage and feedback on its software. When Stabiplan succeeds in effectively involving the service knowledge the prototype provides in its decisions (concerning the development of a new service feature, for example), it is expected that the vendor's SKU level will increase.

In chapter 5, research conclusions are presented. Part of the conclusions is an overview of which hypotheses could be confirmed, and which hypotheses have to be rejected.

2.5 Research Validation

The case study evidence, research outputs and the hypotheses have to be validated in order to be reliable and useful. This validation is accomplished in several ways.

Interviews In order to determine the correctness of the information interviewees provided and the information gathered during direct observations, various reflective questions were asked during the interviews, especially during the interviews with key people of a department or the total organization. Furthermore, a number of project leaders, lead developers and help desk employees were asked to give their opinion about the final prototype and its fit within the 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 Stabiplan. Furthermore, Stabiplan's head developers and CEO were asked to review the SKU report generated by the prototype. Specifically, they were asked to validate and judge the value, relevance and usefulness of the information contained in the report, with respect to the factors that determine the SKU level and the flexibility of a software vendor (as described in section 3.8.2), the goal of the report (improve the software vendor's service knowledge utilization and responsiveness), as well as the hypotheses listed before. All experts received a set of statements related to the research outputs, which they were asked to confirm or reject using a Likert scale (1: strongly disagree, 5: strongly agree).

Testing A number of Stabiplan's head developers and testers, as well as Stabiplan's CEO were asked to test the software prototype developed during the case study. This was done to validate and test the software in practice, inter alia in terms of stability and performance.

Table 2.1 details which party or research method is used to validate which (part of) research design output. As mentioned, results of the validation process at Stabiplan are presented in section 5.1.

Research Output	**Research Method**		
	Interviews	**Expert Validation**	**Testing**
SKU Characterization Model	✓	✓	
SKU Report		✓	
Software Prototype		✓	✓
Hypotheses	✓	✓	

Table 2.1: *Research validation overview*

2.6 Summary

Based on the research triggers described in this thesis introduction, the main research question that is answered by this thesis research is 'How can Continuous Customer Configuration Updating be applied to service-based software and Web services?'. The research that has been done in the context of this thesis can be characterized as design research. Design research is characterized by five process steps, where each process step is related to a specific output type. The first output step of the design research methodology is titled 'Awareness of the Problem'. The research problem of this thesis consists of three elements: the shift from component-based software to service-based software software vendors are making, the application of Jansen's C-CCU model (and its advantages) to service-based software and the status of automatic performance, usage and feedback logging, monitoring and reporting at software vendors in general. A solution is suggested in the form of a concept called 'service knowledge utilization' (SKU). The SKU concept can be seen as 'tentative design' output.

Direct observations, documentation, software, and interviews are used in order to gather facts on which the results of the performed case study are based. After having implemented the SKU concept, it is expected that a vendor's TTM will decrease and that both a software vendor's responsiveness as its SKU level will increase. Research outputs and hypotheses are validated by means of interviews, expert validation and testing.

Chapter 3

SKU: Service Knowledge Utilization

Before the application of Continuous Customer Configuration Updating to service-based software is studied, the underlying concepts of this application are defined and discussed. First, differences and similarities between components and services are discussed. Secondly, life cycles of on the one hand software based on components, and on the other hand software based on services are studied. After having analyzed the groundwork, the concept of service knowledge utilization is presented. The models that were identified as design research artifacts in chapter 2 and are part of the SKU concept, are described. The SKU process model, SKU in a software supply network (SKU-SSN) model and SKU characterization model, as well as the SKU report are detailed in this chapter. Finally, Stabiplan's situation is identified in terms of service knowledge utilization.

3.1 Services vs. Components

3.1.1 What is a Component?

The term ‘component’ is used to characterize many different concepts. This is reflected in the definition of the term in the online dictionary Dictionary.com [26]: it defines a component (among others) as (1) *a constituent part; element; ingredient*; (2) *a part of a mechanical or electrical system: hi-fi components*; and (3) *being or serving as an element (in something larger); composing; constituent: the component parts of a computer system.*

In the context of this thesis research, the term ‘component’ is understood to be a *software* component. According to Wikipedia, ‘*a software component is a system element offering a predefined service and able to communicate with other components*’ [27]. More specifically, Szyperski defines a component as ‘*a binary unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties*’ [100]. Allen [2] describes a component as ‘*an executable unit of code that provides physical black-box encapsulation of related services. Its service can only be accessed through a consistent, published interface that includes an interaction standard. A component must be capable of being connected to other components (through a communications interface) to a larger group*’ — indicating the similarities between a software component and a software service, as is discussed in section 3.1.3.

Components are self-contained — can be deployed and versioned independently. Within object-oriented programming, software components often take the form of objects or collections of objects [100], defined by a method signature. Objects that form parts of component 'instances' are almost never deployed — a software component is what is actually deployed. Other characteristics of high quality software components include reusability and composability. Composite software that is composed of software components is called *component(-based) software*.

3.1.2 What is a Service?

Similar to the term 'component', the notion of a service can be applied to many different concepts in the daily life. Many organizations provide services to customers, clients, employees or business partners. A post office, for example, provides various services to business clients and citizens. Typical, basic services include post delivery-, direct mail- and insurance services. These services are divided over a number of counters, some counters possibly providing the same service to ensure high service availability. To the customer, it does not matter what exactly happens behind the counter to fulfill the specific service — as long as it is fulfilled. Processing and completing a complex transaction may require the customer to visit a number of counters, in fact following a business process flow. This is regularly the case with municipality services (passport acquirement, etc.). According to Bennet et al. [72], the following definition of a service is widely accepted.

Service An act or performance offered by one party to another. Although the process may be tied to a physical product, the performance is essentially intangible and does not normally result in ownership of any of the factors of production.

However, in the context of this thesis research, the term 'service' is understood to be a *software* service. According to Eindrei et al., a software service is generally implemented as '*a coarse-grained, discoverable software entity that exists as a single instance and interacts with applications and other services through a loosely coupled, message-based communication model*' [34].

As already stated in the introduction of this thesis, software services are loosely-coupled (services can be developed and evolved independently, while being dynamically discovered and utilized) and support interoperability. Software services have a clear functional meaning, often related to business goals. Composite software that is composed of software services is called *service-based software*.

3.1.3 Differences and Similarities

While services and components may appear similar at first sight, services and components are not the same, and they do not represent the same concept. Neither is a service equal to an object in programming language terms. Instead, a service is close to the concept of a business transaction (as may be clear from the 'daily life' example discussed earlier). A service consists of a collection of components that work together in order to deliver the business function that the service represents. Furthermore, a service normally accepts more data in a single invocation and consumes more computing resources than a component[1] [79]. Therefore, in comparison, components are finer-grained than services. However, services vary in granularity, depending on their type and goal (an overview of service types is presented in figure 3.4). In addition, while a service maps to a business function, a component typically maps to business entities and the business rules that operate on them [34]. The following example clarifies the differences between and similarities of both concepts. See figure 3.1.

The Customer Licenses class diagram visualizes three classes ('components') that provide methods to obtain information concerning the customer's licenses and applications, including its total license

[1] A service consumes more computing resources than a component because of the need to map to an execution environment, process the XML, and often access it remotely [79].

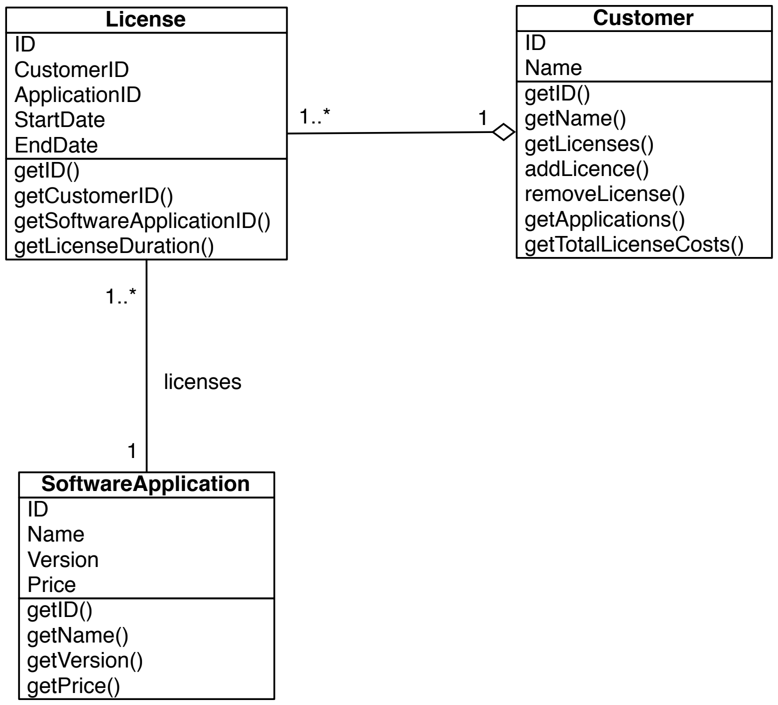


Figure 3.1: *Customer Licenses class diagram*

costs. The SoftwareApplication component provides methods to identify the software application that is licensed by a particular license owned by the customer. The License component provides methods to identify both the customer owning the license and the software application that is licensed, as well as a method to determine the period of time the license is valid. The components are created to closely match business entities and encapsulate the behavior that matches the entities' expected behavior [34].

In service-oriented software design, services are not designed to match business entities — a service manages operations across a set of business entities. Referring to the Customer License example again, a Customer service will respond to any request from any other system or service that needs to access customer information. The customer service 'owns' all the data related to the customers it is managing. Furthermore, the service is capable of making other service inquiries on behalf of the calling party. This means a service is a *manager object* that creates and manages its set of components [34]. Services are designed to solve interoperability issues between applications, or to use within the process of new application composition. Table 3.1 provides a services and components comparison summary.

Research has been done to unify the component and service concepts and characteristics. Yang and Papazoglou [115] propose the concept of a *service component* to raise the Web services abstraction level by facilitating Web service reuse, extension, specialization and inheritance. The service component consists of a single, public interface, which describes via what types of messages and ports it is able to communicate. A service component's *composition logic* specifies how the service is constructed out of Web services by implementing two constructs: the *composition type*, which specifies how the constituent services are executed, and the *message dependency*, which specifies the message dependency relation between the service component and its constituent services.

Criterion	Components	Services
Abstraction	Business entity, including business rules	Business function (set of business entities)
Goal	Solve interoperability issues; Use within new application composition	Create detailed business logic for applications
Invocation	Restricted amount of data accepted; low use of computing resources	More data accepted; high use of computing resources
Granularity	Fine-grained	Relatively coarse-grained, variable
Definition	Method signature	Message signature (WSDL)

Table 3.1: *Services and components comparison*

The service components of Yang and Papazoglou should not be confused with the service components described in the Service Component Architecture (SCA). SCA is based on the idea that a business function is provided as a series of services, which are assembled to create solutions that serve a particular business need. Service Component Architecture provides a model both for the composition of services and for the creation of service components, specifically for building applications and solutions based on a SOA [95]. In SCA, the unit of construction is the *component*, consisting of a configured instance of a piece of program code providing business functions. The business function is offered for use by other components as *services*. SCA describes the content and linkage of an application in assemblies called *composites*. Composites can contain components, services and references, including the descriptions of the connections between these concepts. The core of the architecture is the *Assembly model*, which describes applications and solutions in terms of components and composites, which group, organize and connect components into larger assemblies [63].

While effort is made to unify the concepts of a component and a service, and to merge the characteristics of both concepts, one should not neglect the differences between the notion of a component and a service. The efforts towards an unification of both concepts can be seen as an acknowledgement of the contrast between the concepts.

3.2 Views on the Software Life Cycle

Before discussing software life cycles, it is important to note that the term 'software life cycle' is not equivalent to, and should not be confused with 'Systems (or Software) Development Life Cycle' (SDLC). While both concepts are frequently used interchangeably, in this thesis, we use the term 'software life cycle' to refer to the complete software life cycle — including the 'life' of software *outside* the organization of a software vendor. The concept of SDLC mainly focuses on the (development phase in the) life cycle of software *at* a software vendor [30].

First, this section describes the life cycle of component-based software. Next, the life cycle of service-based software is presented.

3.2.1 Component-based Software Life Cycle

The International Organization for Standardization (ISO) published a standard for software life cycle processes, defining all tasks required for developing and maintaining software [42; 61]. Furthermore, a number of software life cycle models have been developed [20; 31]. The view on the component-based software life cycle described in this thesis is based on the CCU and C-CCU models of Jansen [56] and is similar to the staged model of Rajlich and Bennet [89]. Based on this literature, the component-based software life cycle consists of six phases: development, release, delivery, deployment, activation and usage. These phases of the component-based software life cycle are discussed and explained below; see table 3.2.

Development	The software vendor develops its software (main software product, software updates, major and minor patches, etc.), including tools and manuals. This includes the composition of different components in order to create *composite* software.
Release	In the release phase, the software vendor announces and releases a new version of its main software product, an update, or a patch.
Delivery	Concerns the delivery of software, licenses, manuals and knowledge to customers.
Deployment	Software is deployed (installed and configured), removed (roll-back) or updated at the end-users' systems. In large enterprises for example, the deployment of software updates may affect thousands of computers and involve terabytes of data.
Activation	In the activation phase, software is activated with a license (software code, hardware dongle, etc.). The 'retirement' or *de*activation of software — that for example, takes place when the company using the software decides the software has to be replaced by other software — also happens within this phase.
Usage	Concerns the usage logging, error reporting and feedback management of the deployed and activated software.

Table 3.2: *Phases of the component-based software life cycle*

As concluded by Rajlich and Bennet [89], each phase has very different technical solutions, processes, staff needs, and management activities. Being well applicable to component-based software, the fit between the component-based software life cycle described above and service-based software is being questioned [56]. In the next section, a service-based software life cycle is presented. Next, differences and similarities between both life cycles are discussed.

3.2.2 Service-based Software Life Cycle

In literature, the phases that are part of the (web) service life cycle are often presented in a layered fashion. Papazoglou and Georgakopoulos [85] define three layers (or service types) within the service-oriented architecture: basic services, composite services and managed services. The first layer, *basic services*, deals with service discovery, selection, binding, composition and publication. The second layer, *composite services*, concerns coordination, conformance, monitoring and Quality of Service. The upper layer, *managed services*, is concerned with certification, Service Level Agreements (SLA), and assurance and support processes. As also observed by Milanovic [76], only the basic services layer is well-defined and standardized. Within the Web services architecture, the composition, selection, binding and publication of services are realized by WSDL, SOAP and UDDI (see section 1.2.1).

In this thesis, the definition of (positioning of, and processes within) the management phase of the service-based software life cycle is based on other literature, overlapping both the composite and managed services layers mentioned before. With respect to the service management framework presented in [52], 'service management' entails the creation, updating, deployment, executing and monitoring of services. According to [87], service management includes visibility and control capabilities, as well as interactive monitoring, alerting services, administration, and reporting.

When a service is online for a specific period, the service is managed (redeployed or reconfigured, for example). In most cases[2], services, end-users or other 'clients' are able to utilize the service — in its new deployment state and configuration settings — again when the process of managing the service has been completed. At that point in time, the processes of usage monitoring and usage feedback (re)initiate, because of the new deployment state or configuration settings. The usage phase is the last phase in the life cycle of service-based software. See table 3.3.

Development	The development phase concerns the development of a service by a software vendor.
Discovery	In the discovery phase, services are published and discovered — often by utilizing a service repository (UDDI or other service registry).
Selection	Based on characteristics of the service that is requested, as well as other requirements, a service(s) selection is made.
Composition	When needed, a new service is composed and published, based on the services selected in the previous phase.
Management	This phase includes the deployment, configuration, licensing and activation of services.
Usage	In the usage phase, the utilization of a service is monitored and logged. This phase also includes the process of usage feedback gathering and reports and statistics generation.

Table 3.3: *Phases of the service-based software life cycle*

3.2.3 Differences and Similarities

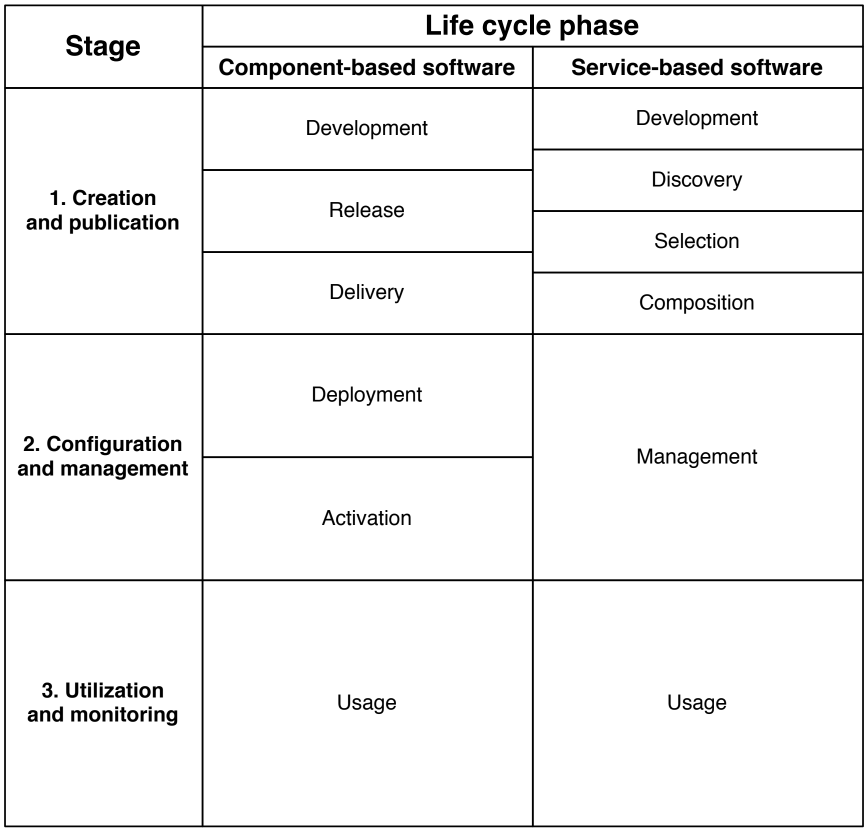
Observing both life cycles, one could conclude that both cycles are quite similar in terms of *what* the cycles are life cycles of — namely, software — and *how* this software is evolving through the different phases of its life cycle. In both life cycles, a similar pattern with respect to the characteristics of the life cycle phases can be observed. The phases of both life cycles can be divided in three stages: 'creation and publication', 'configuration and management' and 'utilization and monitoring'. See figure 3.2.

1. First, the software is created and published. In the component-based software life cycle, the process of software creation and publication comprises the development, release and delivery phases. Analyzing the service-based software life cycle, the phases 'development' and 'discovery' are comprised by this process. Web services that are composed of other, more fine-grained services (*composite* services), also pass through the selection and composition phases. When this stage (stage 1) is completed, the software that is subject to management, utilization and monitoring, is defined. In the component-based software life cycle, this stage is completed after completing the delivery phase. With respect to the service-based software life cycle, stage 1 is completed after completing the composition phase.

[2] The management of a particular service could also entail the discontinuity of this service.

2. Second, the software is configured and managed. Observing the component-based software life cycle, the process of software configuration and management comprises the deployment and activation phases. In the service-based software life cycle, this process is implemented by the management phase. Having completed stage 2, software is ready to be utilized and monitored.

3. Third, the software is utilized and monitored by humans or other software. Furthermore, feedback concerning the software is gathered. Obviously, this stage is implemented by the usage phase of both the component-based software life cycle and the service-based software life cycle.



Stage	**Life cycle phase**	
	Component-based software	**Service-based software**
1. Creation and publication	Development	Development
		Discovery
	Release	Selection
	Delivery	Composition
2. Configuration and management	Deployment	Management
	Activation	
3. Utilization and monitoring	Usage	Usage

Figure 3.2: *Differences and similarities between the component-based software life cycle and the service-based software life cycle*

Once component-based or service-based software is deployed, its life cycle continuously runs through the stages 'configuration and management' and 'utilization and monitoring' — until the software is discontinued (e.g. uninstalled at the configuration and management stage). Independent of the software type (component-based or service-based), software vendors update their software regularly. At that moment, the software switches from 'utilization and monitoring' to 'configuration and management'. When the software is updated, or unsolvable update issues occur, the reverse switch is made. A more detailed illustration of these situations is provided in section 3.5.

A number of differences between both life cycles can be observed. First, a noticeable difference is the absence of a release phase in the service-based software life cycle. Regularly, service software is accessed via its address (URL) and a web browser. New releases, updates and patches are directly published and applied ‘from behind the scenes’ to the client or end-user — at most, the client or end-user is redirected to another location in order to detect the changes. Google [44] added several features (e.g. chat, color labels, rich text editing) to its mail service Gmail[3] just by updating the service ‘server-side’ — no specific action was needed from the end-users in order to be able to use the new features. The online personalized start page service Netvibes[4], a typical Rich Internet Application (RIA) similar to Gmail, is updated similarly: several releases are published, without requesting any significant effort from the end-users.

Secondly, a difference that is also observed by Jansen [56], is the absence of a delivery phase (or a comparable phase) in the service-based software life cycle. This is because with respect to service-based software (life cycles), no delivery start- or endpoint (or periods) in time can be indicated. The delivery of service-based software takes place in a completely transparent manner, mostly virtually and through widely-used communication protocols of the Internet, like SOAP (see section 1.2.1.2) or the Hypertext Transfer Protocol [40] — unlike component-based software, service-based software usually is not delivered in a physical form (e.g. a software package with a CD or DVD).

Thirdly, the service-based software life cycle has no explicit deployment phase. While service-based software has to be deployed before it can be used, the process of service deployment is of a smaller scale than that of component-based software. This is mainly because of the fact that in order to use the software, services often are installed (or updated) at one server (‘entry point’), while the use of component-based software may require installation in numerous software and hardware environments (on all PCs of the employees of an enterprise department, for example). In some cases, service-based software does not have to be deployed at all, for example when software development is done at a web server.

Concluding, the release and deployment phases of the service-based software life cycle are less complex and of a smaller order than these of the component-based software life cycle, while the delivery phase is even completely absent in the former life cycle. Therefore the release and deployment phases are incorporated in the discovery and management phases, respectively, since ‘discovery’ includes the publication of a service and in the service-based software life cycle, ‘management’ is the only phase in the configuration and management stage.

3.3 Service Types

Whether a service passes through the life cycle phases ‘selection’ and ‘composition’, depends on the type and granularity of a service. Krafzig et al. [66] present a service type classification by defining four types of services:

- Basic services
- Intermediary services
- Process-centric services
- Public Enterprise[5] services

Depending on the type and granularity of a service, the life cycle phases ‘discovery’, ‘selection’ and ‘composition’ are optional. Table 3.4 gives an overview of the different service types[6].

[3] http://gmail.com
[4] http://netvibes.com
[5] The type and role of the ‘Public Enterprise’ mentioned here are corresponding to those of the service-oriented enterprises in figure 1.2 of the thesis introduction.
[6] The table is adapted from, and based on the work by Krafzig et al. [66]

Criterion	**Basic**	**Intermediary**	**Process-centric**	**Public Enterprise**
Description	Simple data- or logic-centric services	Technology gateways, adapters	Services encapsulating process logic	Services shared with other enterprises or partner organizations
Impl. complexity	Low–moderate	Moderate–high	High	Service-specific
State management	Stateless	Stateless	Stateful	Service-specific
Reusability	High	Low	Low	High
Change frequency	Low	Moderate–high	High	Low
Granularity	Fine-grained	Fine-grained–coarse-grained	Coarse-grained	Service-specific
Full life cycle	No	Yes	Yes	Service-specific
Example	Stock quote Web service	ERP adapter service	Airline ticket reservation service	Shipment tracking service

Table 3.4: *Service type comparison (adapted from [66])*

Analyzing this service type overview, one can observe that typically services that have high implementation complexity, low reusability and a high change frequency ('intermediary' and 'process-centric' services) evolve during the complete service-based software life cycle — including the life cycle phases 'selection' and 'composition'. These characteristics indicate that such a service is considerably complex, likely consisting of a number of finer-grained services and relating to a specific business process. These types of services are difficult to implement and are subject to many changes. Because a high composite service implementation change frequency may imply a high service composition change frequency (e.g., the selection of basic services a composite service exists of, is altered), it is likely that coarse-grained, composite services (i.e., legacy systems adapter services or ticket reservation services) pass the the optional phases of the service-based software life cycle more frequently than basic services.

On the contrary, basic and highly reusable services (like a stock quote service or a weather forecast service) have a low change frequency and are relatively easy to implement. Because of the fine-grained character of basic services, it is not likely that basic services are composite services. Mainly because of the low change frequency, it is unlikely that such services go through both selection and composition phases of the service-based software life cycle. See figure 3.4.

3.4 C-CCU and Service-based Software

Having described both software life cycles, in this section, the C-CCU model is discussed with respect to service-based software and the service-based software life cycle.

3.4.1 Definition

Jansen indicated that automating steps in the process of Customer Configuration Updating (CCU), which he defines as '*the release, delivery, deployment and usage and activation processes of product software*' [57], ultimately contributes to Continuous CCU, enabling a software vendor to become more responsive to changes in customer and market demands.

Obviously, the definitions of both CCU and C-CCU (see section 1.4) are mainly targeted on, and applicable to component-based software — which becomes clear by observing the life cycle phases of the component-based software life cycle. Concerning the application of the C-CCU concept to service-based software, this is problematic.

As described in section 3.2.3, the life cycles of both software types only match in stage 3 ('utilization and monitoring'). While both life cycles entail creation, configuration, and utilization stages, the cycles differ in the first two stages: the service-based software life cycle lacks a delivery phase, and its release and deployment phases are significantly shorter than the component-based equivalents (the phases are incorporated in the discovery and management phases of the life cycle, respectively). See figure 3.2.

Apart from its focus on component-based software, the concept of C-CCU relies on the (quality of) releases of software — as illustrated by the definition of the concept. Obviously, this is also problematic in applying C-CCU to service-based software, since the release phase of the service-based software life cycle forms a small and relatively little time-consuming phase compared to other phases in its life cycle.

Observing these limitations, C-CCU can only partly be applied to service-based software and Web services. Analyzing figure 1.4, only the logging and feedback aspects (policies) of the C-CCU model are completely applicable to this type of software. This is also visualized by the life cycle comparison in figure 3.2. Both life cycles only completely match in the utilization and monitoring stage (usage phases).

Therefore, a new term is introduced, taking into account the limitations described above: 'Service Knowledge Utilization' (SKU). SKU describes the application of C-CCU to service-based software, and expresses to which extent (or: on which area) the concept of C-CCU is applicable to service-based software. SKU focuses on the third stage of the software life cycles ('utilization and monitoring') and is defined 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 may be clear, to enable a software vendor developing service-based software to become more flexible and responsive to performance and usage changes in its software, the role of service knowledge (and the utilization of this knowledge, SKU) is of high importance. This is explicated in section 3.4.2 and 3.8.

3.4.2 Software Vendor Responsiveness

The introduction of this thesis already pointed out the increasing demand on enterprises (and specifically software vendor organizations) to be highly dynamic and agile organizations and to respond quickly to market situations, demands and changes. As concluded earlier by Jansen [60], and confirmed by findings in [9], implementing C-CCU in a software vendor's organization reduces overhead from various software life cycle processes, enabling this organization to become more responsive to change.

In the context of concept of service knowledge utilization presented in this thesis, we define a software vendor's responsiveness as follows.

Responsiveness The speed at which changes are implemented based on service performance, usage and feedback data, relative to the size of the changes.

While the ability to be responsive to changes in customer and market demands is important to software vendors developing component-based software, this ability is important to companies that develop service-based software (e.g. Web services), too — both in business to business (B2B) and business to consumer (B2C) environments (see figure 1.2). Software vendors developing software services for companies that have implemented a service-oriented architecture successfully, may need to respond even earlier and quicker. This is illustrated by examples below.

> A small software company called *Terra Computing* develops a route planning service. Based on a start- and end location, the route planning service presents the fastest route between both locations, based on recent traffic data. The service is based on two 3rd party services: one for acquiring map data, and one for acquiring traffic data. Terra Computing's service views maps — on which the actual route is drawn — by utilizing the acquired map data. The software vendor has contracted Microsoft to provide this geographical data. To this end, Terra Computing licensed Microsoft's Live Search Maps Web service. However, at one day, the software vendor receives an offer from Google concerning the use of its Google Maps service. After having investigated the Google's service, the Terra Computing decided it wanted to replace the Live Search Maps Web service (and its corresponding license) by the map service of Google, based on the extra functionality it provides and its lower costs.

In the example of Terra Computing above, Microsoft's map service is replaced by a similar service of Google. Especially when the offered functionality of the service is similar, both services are easily interchangeable, enabling the software vendor to switch services without vast architecture changes. At the same time, another software company (Microsoft) is losing a customer within one day.

> *Web Mobile* offers a short message service (SMS) Web service, enabling 3rd parties as well as end-users to send free SMS messages via the Internet. The service was an enormous success in the first months after its initial release. However, by analyzing the web server's logs, Web Mobile's system administrator discovered a steady decrease in the amount of visitors and messages sent per day. Further analysis of the market situation points out that the day before the amount of visits started to decrease, Web Mobile's main competitor published a similar service, with extra features (sent multiple messages at once, receive read confirmation) and an easy-to-use user interface.

As the examples above illustrate, customer relationship management (CRM), customer communication and service knowledge are important to software vendors, especially during the time software is actually used by customers and clients. When software vendors gather and utilize knowledge about the performance (number of exceptions, response time, etc.) and usage of their software and feedback and wishes of their customers, situations described in the example above might be prevented, or prevented earlier. Instead of directly switching services, the software vendor of the example above could also demand extra functionality of Microsoft — with the effect that Microsoft's service offers the same functionality as Google's service and the software vendor decides not to change services — for example. Web Mobile's customers may have kept using Web Mobile's message service instead of 'blindly' switching to another SMS service, when Web Mobile introduced new features earlier.

Factors that influence the responsiveness of a software vendor are detailed in section 3.8. In the next section, the SKU process model is presented. By visualizing important knowledge streams and processes with respect to the communication between the software vendor developing service-based software and the customer, this model forms the basis of the SKU-SSN model (presented in section 3.6).

3.5 SKU Process Model

The previous section illustrates that for a software vendor developing service-based software, knowledge about the service performance, usage and feedback is required to be an agile and dynamic organization that responds quickly to market changes and customer demands. In this section, the SKU process model is presented. The SKU process model depicts important knowledge streams and activities within the process of service knowledge utilization (i.e., continuous customer configuration updating for service-based software). The model is described by using the process-deliverable diagram (PDD) technique [104], see figure 3.3.

3.5.1 Concepts

The SKU process model consists of six main activities (development, discovery, selection, composition, management and usage), every activity corresponding to a particular phase of the service-based software life cycle. Various concepts are important to complete the activities. Important concepts within this model (and within the service-based software life cycle) are described below.

COMPONENT A binary unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties [100]. It can form a part of a service. See section 3.1.1.

SERVICE A coarse-grained, discoverable software entity that exists as a single instance and interacts with applications and other SERVICEs through a loosely coupled, message-based communication model [34]. Often registered in a SERVICE REPOSITORY. Can be composed of COMPONENTs or other, finer-grained SERVICEs. See section 3.1.2.

INTERFACE DEFINITION Defines a set of public SERVICE method signatures, logically grouped but providing no implementation [34]. An interface defines a contract between the requester and provider of a SERVICE. Any implementation of an interface must provide all methods. When a SERVICE is published, its interface identifies and represents the SERVICE and enables its dynamic discovery. See section 1.2.1.2.

SERVICE REPOSITORY A registry framework that enables the description and dynamic discovery of web SERVICEs. See section 1.2.1.2.

USAGE LOG A data record containing software (service) usage data. Based on [38], this log contains software service identification and configuration information, as well as operations and events details.

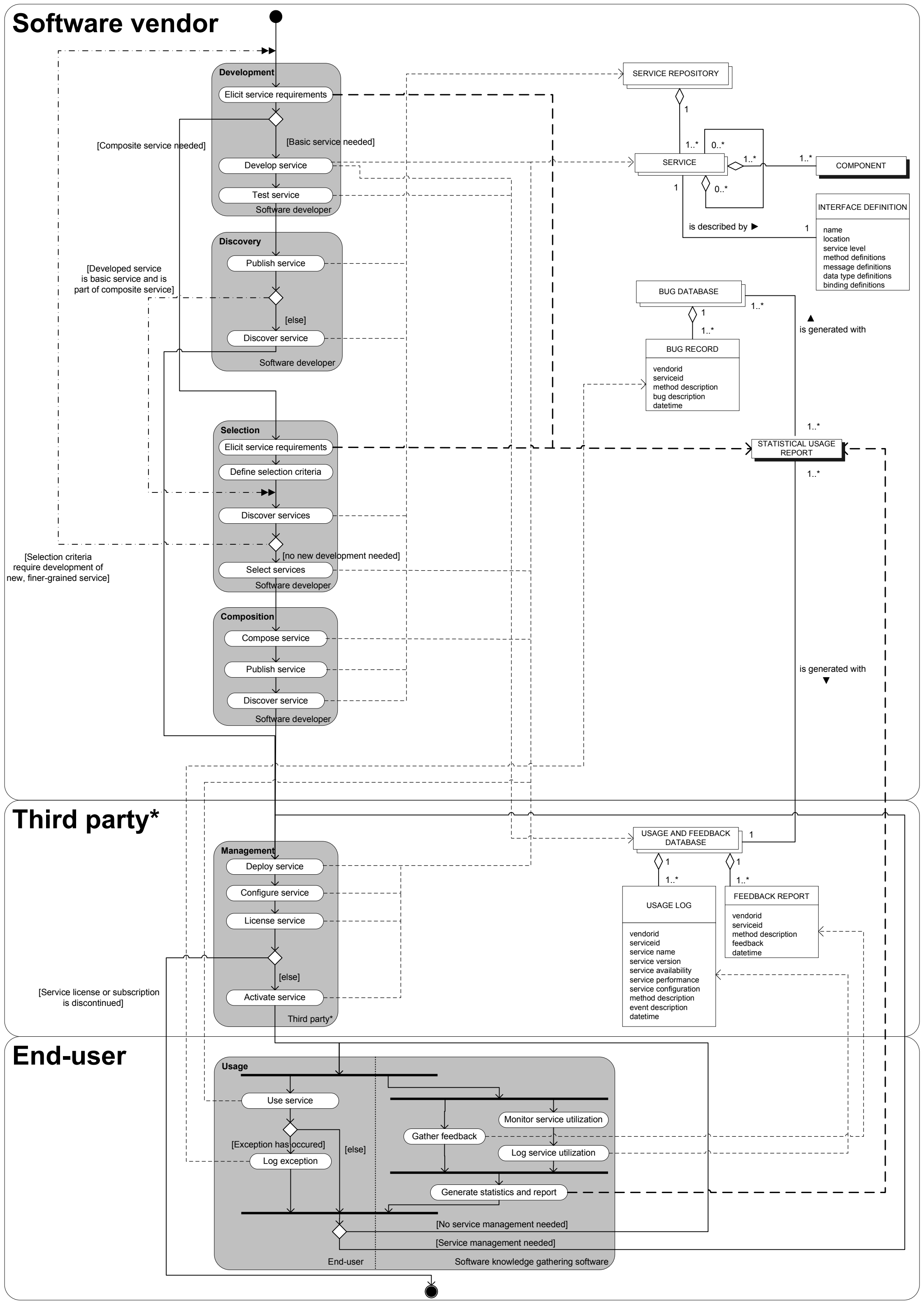


Figure 3.3: *SKU process model*

FEEDBACK REPORT A data report containing user feedback and user experience data.

USAGE AND FEEDBACK DATABASE A database containing USAGE LOGs and FEEDBACK REPORTs.

STATISTICAL USAGE REPORT A report, containing performance, usage and feedback statistics, based on USAGE LOGs and FEEDBACK REPORTs in the USAGE AND FEEDBACK DATABASE.

BUG DATABASE A database containing BUG RECORDs.

BUG RECORD A data record capable of containing bug information, including vendor-, service- and method data.

In the next section, the parties depicted in the model are characterized.

3.5.2 Parties

All activities and concepts within the SKU process model are grouped within a party: software vendor, end-user or third party. Like figure 3.3 depicts, the usage and feedback database could be located at a 'third party'. This is the case, for example, when neither the software vendor nor the end-user's organization has the capacity or resources to host this database. A software vendor or an end-user's organization may decide to outsource the service management activities and responsibilities to a third party likewise.

However, one should note that regarding the location of the usage and feedback database, various scenarios are plausible. This database may be hosted at the software vendor, at a third party, or at the end-user's organization. Nevertheless, the usage and feedback database may also well be hosted at *both* organizations — as is assumed in the SKU-SSN model (see section 3.6). The management activities may well be executed at one of the three parties, too[7].

3.5.3 Activities

The first four activities of the SKU process model are typically passed by and passed at a software vendor. However, depending on the service granularity, the discovery, selection and composition activities of both the SKU process model and the Web service life cycle are optional (as indicated in section 3.2.3). The last two activities, 'Management' and 'Usage', are phases the end-user (or third party) goes through. Within each activity, the main sub activities of that activity are depicted.

It is important to note that the activities at both the software vendor and the end-user's organization are separate processes that take place continuously. The activity link between software vendor ('Discover service') and end-user's organization ('Deploy service') is only passed when the end-user's organization decides to deploy a new service. Apart from that situation, the SKU process model represents two separate processes — one at the software vendor, and one at the end-user's organization — taking place continuously.

Another point of attention is formed by the first three activities taking place at the software vendor: development, discovery and selection. If the software vendor initiates the 'Development' activity with the intention to only create a basic service, the 'Development' and 'Discovery' activities are passed and sequentially, the 'Management' activity is entered. However, if the 'Development' activity is initiated to develop a composite (intermediary or process-centric) service, the discovery activity is skipped and the selection and composition activities are passed. In that case, a new development activity may be initiated if the activity 'Discover services' (within 'Selection') does not deliver the services needed to create the planned composite service. Because the initiation of

[7] This is indicated by the asterisk (*) in figure 3.3.

a new 'Development' activity may be recursive (a heavily composite service may consist of finer-grained, intermediary services, which consist of smaller, composite services, which are composed of basic services[8]), this initiation is depicted with a double-headed arrow[9]. So when the development activity is entered, the number of activities within the process depends on the type of the service that is being developed, as well as wether (in the case of a basic service) the service is part of a composite service. When the development activity is entered as part of a recursive loop (a composite service is developed), the recursion ends in the 'Discovery' activity before the 'Discover service' sub activity — when the granularity of the developed service is such that it can be considered a basic service, that is. In that case, a second double-headed arrow is passed, returning to the 'Selection' activity again, skipping the activities 'Gather requirements' and 'Define selection criteria'. Otherwise, the 'Discovery' activity was skipped and the 'Selection' activity was initiated, possibly invoking a new recursion.

As depicted in the last two activities of the model, the end of the service-based software life cycle could end from within the 'Management' activity. Within this activity, services are deployed, configured, licensed and — in case of a valid license or subscription — activated. The life cycle ends when the service license or service subscription is discontinued. Otherwise, the service is activated and the 'Usage' activity is initiated. Obviously, this activity represents the usage of the service by end-users. When an (unhandled) exception occurs, a bug record is created which is stored in the bug database. During and concurrently with the service usage, service usage is monitored and logged. Usage logs and feedback are stored in the usage and feedback database. A statistical usage report can then be generated based on the data in this database.

Note that, apart from the logging of usage and feedback (provided) by end-users, a software vendor may decide to log the data associated with both concepts already in the phases of (pilot) testing or quality assurance (not depicted in the SKU process model), in order to gather service knowledge in an early stage.

When service management is needed, the 'Management' activity is initiated again. One should note that this activity can pass very quickly, for example in the case that the activity is initiated only in order to change one or two configuration parameters. When no service management is needed, the usage activity is reinitiated.

3.5.4 Relations

The dotted lines represent the relations between concepts (data) and processes (activities). For example, the model makes clear that the usage and feedback database is filled in the first and last activities defined in the model ('Development' and 'Usage'). The accentuated lines are essential in the communication between the software vendor and the end-user. In the SKU process model, this communication is based on the statistical usage report, and takes place in the following activities.

1. Gather requirements (Development) → Statistical usage report
2. Gather requirements (Selection) → Statistical usage report
3. Generate statistics and report (Usage) → Statistical usage report

Activities 1 and 2 take place at the software vendor, and activity 3 takes places at the end-user's organization. When a software vendor decides to develop or compose a new service, it is important to know about the usage and performance of older or related services (as motivated earlier). In passing these activities, the software vendor is able to acquire this knowledge by requesting the

[8] The number of 'composition levels' is actually variable, making this a recursive process.
[9] The double-headed arrow is not described in the PDD drawing technique definition, since this technique does not (yet) support activity recursion [104], and is therefore supplementary.

statistical usage report. This report is filled at activity 3, based on exception data, service usage data and feedback data.

Apart from within the software vendor's software service development activities, service knowledge is potentially of use in a number of other activities. This is depicted by the SKU-SSN model, presented in the next section.

3.6 SKU in a Software Supply Network

Having discussed the SKU process model, the service knowledge utilization in an SSN (SKU-SSN) model is presented in this section. Mainly focusing on the development department within a software vendor's organization, the SKU process model visualizes the process of service knowledge utilization in the context of the service-based software life cycle. The SKU process model clarifies which parts of the life cycle occur where (at which parties) and clarifies which concepts play an essential role in the communication between a software vendor developing service-based software and its customers.

The SKU-SSN model illustrates the role of a software vendor developing service-based software in its software supply network (SSN) by visualizing the communication and role of service knowledge usage (SKU) between on the one hand, the software vendor and external software vendors that have licensed a one or more services of the vendor (B2B), and on the other hand, the software vendor and its customers (SOEs) and end-users of that customers (B2C). As such, a software vendor's software supply network is composed of its B2B and B2C relations.

As explained below and depicted by the SKU-SSN model, software (usage) knowledge and feedback is potentially of use within different management levels — within the software vendor's organization itself, or within these of external partners or customers.

Three parties, and a number of relations between them, can be identified.

The Software Vendor The software vendor is the main organization of the SKU-SSN model. It develops and publishes service-based software that is licensed by service-oriented enterprises (B2C) and external software vendors (B2B), as indicated by the dashed lines. Service-oriented enterprises have licensed the software for use within their activities or SOAs, while external software vendors license services of the software vendor in order to build or support their own software. In section 3.4.2 for example, it is described how a relatively small external software vendor, Terra Computing, has licensed Microsoft's Live Search Maps Web service to provide data for the route planning service they develop and publish. The services of both the service-oriented enterprises as the external software vendors are used by end-users. Note that a software vendor may make its service-based software also directly accessible to end-users. A vendor may launch a free-of-charge SMS Web service, for example.

External Software Vendors External software vendors may have licensed software of the software vendor as an essential part of their own software and services. Note that, as described in section 1.1.3, the (external) software vendors and their customers (the service-oriented enterprises) may all be networked companies and be part of a highly-integrated business network, sharing software and services. In that case, the SKU-SSN model is applicable to all the external software vendors, interchanging the roles of the software vendor and the external software vendor but maintaining the B2B relationship between both organizations.

Service-Oriented Enterprises The service-oriented enterprises (SOEs; also see section 1.2.1) in the SKU-SSN model represent the customers of the software vendor. As illustrated in the model, the SOEs have licensed service software of the vendor. This software may well be part of the enterprise's SOA service portfolio and may be needed for internal use (by employees)

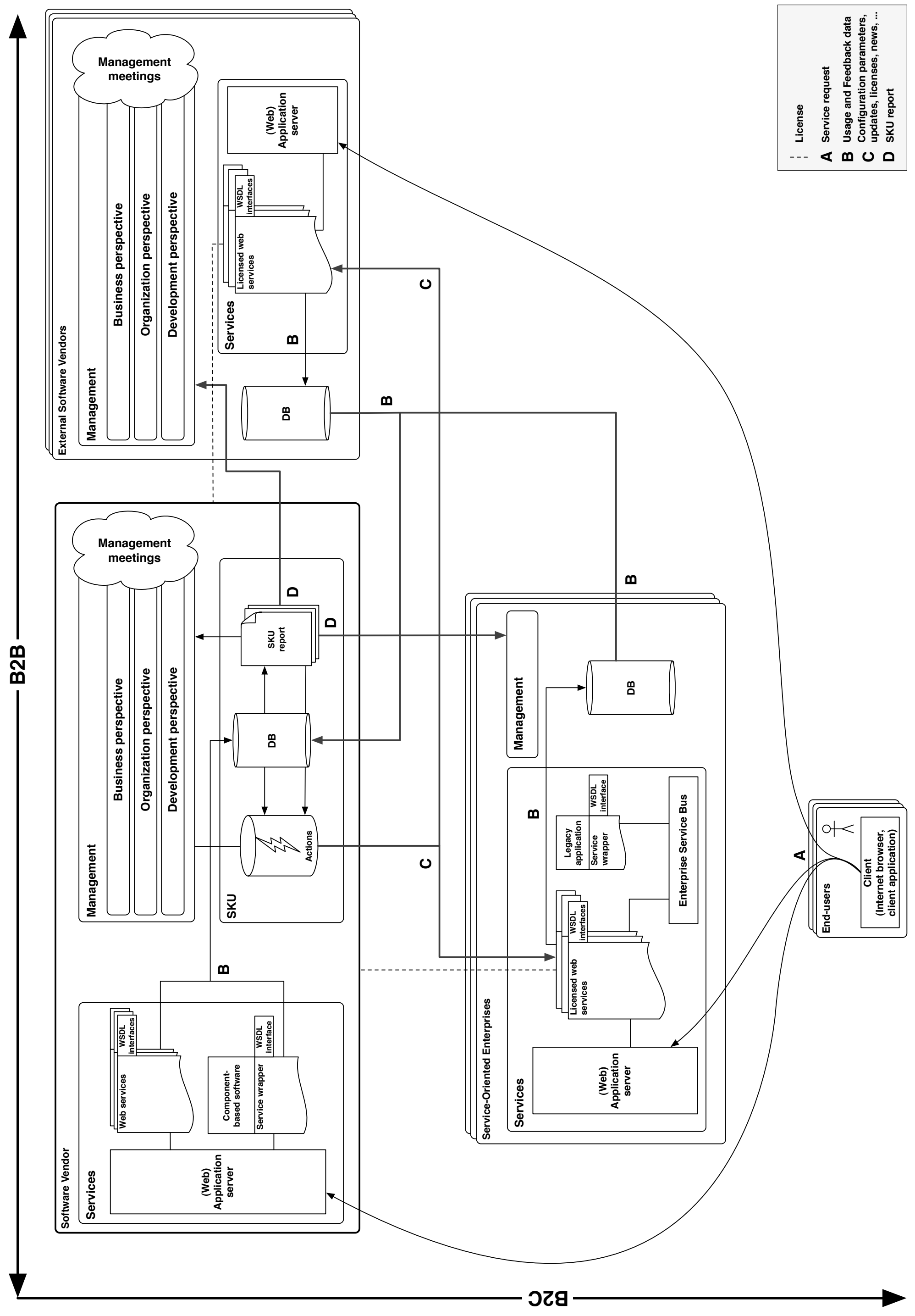


Figure 3.4: *SKU in a software supply network (SKU-SSN) model*

or external use (by end-users). Note that a service-oriented enterprise may license and use service-based software of the software vendor and one or more external software vendors concurrently.

End-users End-users represent the people that use service-based software of (external) software vendors and service-oriented enterprises. End-users generate 'service demand' (see the A-lines in the model) and drive all other parties in the SKU-SSN model to develop and publish their software and services. Described in the introduction of this thesis, software vendors increasingly try to adapt to market (end-user's) demands. The SKU-SSN model illustrates a manner to achieve a high level of responsiveness to changes in market, customer and end-user demands. Note that end-users do not directly obtain licenses for the service-based software they use.

At all parties that have licensed and utilize one or more services, the usage and feedback of these services is logged. The resulting usage and feedback data is stored in a local database[10]. On a regular basis, relevant data in these local databases is transported to a main usage and feedback database at the software vendor that has developed and published the service-based software of which the usage and feedback is logged[11]. See the B-lines in the model.

After a pre-defined period, the main usage and feedback database at the software vendor is filled with usage and feedback data of all services licensed by external software vendors and service-oriented enterprises. As depicted in the model, two products, both based on the received services usage and feedback data, can be identified: the SKU report (equivalent to the statistical usage report of the SKU process model) and a set of actions.

Detailed in section 3.7, the SKU report quantifies the usage and feedback of the service-based software involved and contains three indices that express the software quality in terms of performance, usability and customer usage. The report has a number of applications (see D-lines in model). First, this report is of use within the software vendor itself, giving the vendor insight in the usage and feedback behavior of its customer base. Secondly, the report is potentially of use in maintaining both B2B (software vendor to external software vendors) and B2C (software vendor to service-oriented enterprises) relationships. See section 3.7.4.

Apart from the SKU report, the software vendor may define a set of actions that may be based on particular aspects of the usage and feedback data in the main usage and feedback database. Each action definition consists of one or more conditions. When all (or any) conditions are met, the action is triggered. For example, one could define an action describing that an e-mail has to be sent to Support when a customer has experienced more than one exception on one day. Another action could describe that a 'Software Change Request' has to be created when a service method is not used for more than one month. Moreover, based on one or more conditions, actions may also describe to send specific service configuration parameters, updates, license information or company news to a particular customer in specific situations (C-lines in model), therewith fulfilling the demands and wishes of that customer or adapting to a new market situation.

In the next section, the SKU report is presented. As part of the construction of this report, three indices are calculated. These indices represent the status of a software vendor's services in terms of performance, usability and customer usage. Being generated from usage and feedback data, the SKU report forms an essential part of the SKU-SSN model and the success of an SKU implementation.

[10] Note that, as illustrated by the SKU process model, this database may also be located at a third party.
[11] While in the SKU-SSN model, distinct databases are depicted at each party, the amount of databases may vary. One database located at the software vendor, with all B-lines directed to it, is another plausible situation.

3.7 SKU Report

As already argued by Mendelson and Ziegler [75], *information awareness* (‘knowing what is going on in the surrounding world’) is becoming increasingly important for enterprises. The authors state that this awareness is critical for survival in the information technology era — let alone the network era. The first process that can be utilized to increase the information awareness of an enterprise is titled ‘Listening to the Customer’ [75]. 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.

The purpose of this SKU report is to contribute to a (customer) information awareness increase, as well as an increase of the utilization of this information (SKU). This section describes the structure, contents and potential applications of the SKU report.

3.7.1 Terms and Definitions

The SKU report contains data and information about a number of concepts. Before detailing the structure of and information in the report, these concepts have to be defined. See the list below.

- **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 $S_i = \{s_{i(1)}, ..., s_{i(v)}\}$.
- **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 $M_j = \{m_{j(1)}, ..., m_{j(q)}\}$.
- **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 $E_j = \{e_{j(1)}, ..., e_{j(r)}\}$.
- **Event Types** A number of event types exist, which are defined as $T = \{t_1, \ldots, t_w\}$.

The concepts above are subject to two constraints:

- In the context of the report, 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. Furthermore, the SKU report only contains data and information about licensed services: $S_i \subseteq S$.
- Every event $e \in E$ has an unique type $t \in T$: $\forall e \in E : t(e) \in T$.

When studying the metrics and indices described in section 3.7.3, one may note that of all concepts, the ‘event’ concept is not mentioned within that context. For example, events are not explicitly used as part of the service indices construction. This is because an event has a rather abstract and intangible character, making the concept of an event unsuitable to measure performance, usability or utilization of. Therefore, events mainly play a role in the construction of metrics and in the context of service usage logging, for example in the registration of exceptions that occur while using a service. Similarly, the concept of an event type is introduced to enable event categorization in the process of service usage logging.

3.7.2 Structure and Contents

For each method of each service, the report contains metrics, calculated in order to quantify and measure the status and quality of the services used by customers. The report contains a lot of data because, for example, the software vendor has published a large number of services, or because the software vendor has a large customer base. Furthermore, the SKU report may well be generated on a frequent basis: as described in the introduction of this thesis, nowadays, software vendors have to respond quickly to market changes and customer demands. Therefore, a report that increases the information awareness, may be generated frequently.

The content of an SKU report should be easily interpretable. Specifically, the statuses of the services contained in the SKU report should be comparable with those of earlier reports. Therefore, in an SKU report, the status of a software vendor's services is summarized by the set of metrics-based indices.

3.7.3 Metrics and Indices

The research community has developed a wide range of metrics to measure and quantify the quality of services [62]. 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 [38] 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 [74; 117]. In the context of this thesis research, 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*' [119], or as '*the quality aspect of a Web service that represents the degree of being capable of maintaining the service and service quality*', referring to the assured and ordered delivery for messages being sent and received by service requesters and service providers [93]. We define Reliability as '*The rate of successfully finished transactions*' [73].

Throughput We define the throughput of a service as '*the number of (successfully) completed service requests over a time period*' [90]. When the throughput of a given system increases, the system response time increases. Often, throughput is measured in requests per time unit [108]. One should note that this metric especially is of importance with respect to process-centric services. Generally, services of this type have to meet a specific demand, and therefore guarantee a specific throughput level.

Latency Latency is defined as the '*time taken between the moment the service request arrives and the moment the request is being serviced*' [90]. We define the latency of a service as '*the round-trip time between sending a request and receiving the response*' [93]. The throughput of a system (service) is related to its latency.

Accuracy Generically, 'accuracy' is defined as '*percentage of objects without data errors such as misspellings, out-of-range values, etc.*' [78]. More specifically, we define accuracy as '*the error rate produced by a service*' [90]. In practice, this metric indicates the amount of unhandled exceptions 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.*' [93], '*The percentage of time a source is accessible based on technical equipment and statistics*' [78] or '*The probability a system is up*' [90; 108]. In this thesis, the latter definition is used.

Usability A lot of research has been done on the subject of usability and Human Computer Interaction (HCI) [7; 33; 80; 92; 97] and with respect to the development of usability metrics specifically [37; 49; 96; 120], resulting in different definitions. Frequently, 'usability' is defined as '*a technology's capability to be used easily, enjoyably, safely and quickly*' [33; 92]. According to the ISO, usability is the '*extent to which a product can be used by specified users to achieve specified goals with effectiveness (accuracy and completeness with which users achieve specified goals), efficiency (resources expended in relation to the accuracy and completeness with which users achieve goals) and satisfaction (freedom from discomfort, and positive attitudes towards the use of the product) in a specified context of use.*' [43]. Seffah et al. [96] developed a consolidated usability measurement model based on the research of Shackel [97] and Nielsen [80]. The model identifies a number of usability criteria, among others 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 within the context of this research — users actively 'interact' with a Web service via its graphical user interface (when applicable) — the usability of Web services is defined as the 'minimal action' criterion.

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*' [119]. Within 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*' [78].

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

Table 3.5: *Metrics notation*

The notation of the metrics defined above is depicted in table 3.5. $m_{j(k)}$ represents a method of a service s_j published by a software vendor. Note that the values of the metrics will always be real, non-negative numbers, so

$$\{q_{Rel}, q_T, q_L, q_{Acc}, q_{Av}, q_{Rep}\} \in \mathbb{R}$$

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[12]. Specifically, such a contract is a Service Level Agreement (SLA) between the service provider and the service requester (for example, between Terra Computing and Microsoft in the example of section 3.4.2), among others describing (un)acceptable service levels [70; 107]. A service level is defined as some mark by which to qualify acceptability of a service parameter [70] (aspect), for example a guaranteed minimal availability of a service. Research has been done to specify and monitor SLAs specifically for Web services [64; 65].

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 and is defined as follows.

$$r_{\{Rel,T,Av,Rep\}}(m_{j(k)}) \begin{cases} 1.0 & q_{\{Rel,T,Av,Rep\}}(m_{j(k)}) \geq q_{\{Rel,T,Av,Rep\}_{SLA}}(s_j) \\ \frac{q_{\{Rel,T,Av,Rep\}}(m_{j(k)})}{q_{\{Rel,T,Av,Rep\}_{SLA}}(s_j)} & q_{\{Rel,T,Av,Rep\}}(m_{j(k)}) < q_{\{Rel,T,Av,Rep\}_{SLA}}(s_j) \end{cases}$$

In other words, 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, the metric score of the service method on that metric 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. This metric score type is defined as follows.

$$r_{\{L,Acc,U\}}(m_{j(k)}) \begin{cases} 1.0 & q_{\{L,Acc,U\}}(m_{j(k)}) \leq q_{\{L,Acc,U\}_{SLA}}(s_j) \\ \frac{q_{\{L,Acc,U\}_{SLA}}(s_j)}{q_{\{L,Acc,U\}}(m_{j(k)})} & q_{\{L,Acc,U\}}(m_{j(k)}) > q_{\{L,Acc,U\}_{SLA}}(s_j) \end{cases}$$

In other words, 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, the metric score of the service method on that metric 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.

Because of the inclusion of the metric SLA values $q_{\{Rel,T,L,Acc,Av,Rep,U\}_{SLA}}(s_j)$ in the calculation of the metric scores, both $r_{\{Rel,T,Av,Rep\}}(m_{j(k)})$ and $r_{\{L,Acc,U\}}(m_{j(k)})$ values always are between 0 and zero (including boundaries):

$$\{r_{\{Rel,T,Av,Rep\}}(m_{j(k)}), r_{\{L,Acc,U\}}(m_{j(k)})\} \subseteq [0,1]$$

The metric scores are calculated to eliminate the differences between the domains of the metrics listed in this section. Based on the resulting metric score values, three service indices are constructed.

[12] This was already shortly described in the introduction of this thesis, in the context of service providers.

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 these 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 indices are calculated, representing the status of all services that are licensed by a customer. Next, the construction of the service indices is described.

3.7.3.1 Service Performance Index

The Service Performance Index can be applied on different levels: method level, service level or customer level. In other words, this index is calculated for all methods $m_{j(k)} \in M_j$ of a service s_j, for one service s_j or for all services $s_{i(x)}$ of customer c_i.

$\pi(m_{j(k)})$ represents the value of the Service Performance Index for a service method $m_{j(k)}$. Given a particular service method $m_{j(k)}$, its Service Performance Index is constructed as follows.

$$\pi(m_{j(k)}) = r_{Rel}(m_{j(k)}) \cdot r_T(m_{j(k)}) \cdot r_L(m_{j(k)}) \cdot r_{Acc}(m_{j(k)}) \cdot r_{Av}(m_{j(k)}) \tag{3.1}$$

In words, the Service Performance Index expresses the quality of a method (or service, or collection of services) in terms of its reliability, capacity (throughput), response time (latency), error rate (accuracy) and on-line time (availability). Equation 3.2 shows how the Service Performance Index is calculated for a service $s_{i(x)} \in S_i$ — where S_i is the set of services licensed by customer c_i — as the average performance of its methods.

$$\pi(s_{i(x)}) = \frac{\sum_{k=1}^{q} \pi(m_{i_x(k)})}{q} \tag{3.2}$$

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 these 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 [32].

$$\pi(c_i) = M \cdot \frac{\sum_{x=1}^{v} \pi(s_{i(x)})}{v} \tag{3.3}$$

3.7.3.2 Service Usability Index

Like the Service Performance Index, the Service Usability Index can be applied on method level, service level or customer level. This index is constructed of two metrics that are not utilized with respect to the construction of the Service Performance Index: Usability and Reputation.

$\upsilon(m_{j(k)})$ represents the value of the Service Usability Index for a service method $m_{j(k)}$. Given a particular service method $m_{j(k)}$, its Service Usability Index is constructed as follows.

$$\upsilon(m_{j(k)}) = r_U(m_{j(k)}) \cdot r_{Acc}(m_{j(k)}) \cdot r_L(m_{j(k)}) \cdot r_{Rep}(m_{j(k)}) \tag{3.4}$$

In words, 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. Equation 3.5 shows how the Service Usability Index is calculated for a service $s_{i(x)} \in S_i$ — where again, S_i is the set of services licensed by customer c_i — as the average usability of its methods.

$$\upsilon(s_{i(x)}) = \frac{\sum_{k=1}^{q} \upsilon(m_{i_x(k)})}{q} \tag{3.5}$$

Equation 3.6 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.

$$\upsilon(c_i) = M \cdot \frac{\sum_{x=1}^{v} \upsilon(s_{i(x)})}{v} \tag{3.6}$$

3.7.3.3 Service Client Utilization Index

Analogue to the other indices, the Service Client Utilization Index can be applied on method, service or customer level. $\kappa(m_{j(k)})$ represents the value of the Service Client Utilization Index for a service method $m_{j(k)}$. Given a particular method $m_{j(k)}$, its Service Client Utilization Index is constructed as follows.

$$\kappa(m_{j(k)}) = r_T(m_{j(k)}) \cdot r_L(m_{j(k)}) \cdot r_{Rep}(m_{j(k)}) \tag{3.7}$$

In words, 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. Equation 3.8 shows how the Service Client Utilization Index is calculated for a service $s_{i(x)} \in S_i$ — where again, S_i is the set of services licensed by customer c_i — as the average client utilization of its methods.

$$\kappa(s_{i(x)}) = \frac{\sum_{k=1}^{q} \kappa(m_{i_x(k)})}{q} \tag{3.8}$$

Equation 3.9 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.

$$\kappa(c_i) = M \cdot \frac{\sum_{x=1}^{v} \kappa(s_{i(x)})}{v} \tag{3.9}$$

Having presented all service index constructions on all levels (methods, services and customers), note that all metric scores $r_{\{Rel,T,L,Acc,Av,Rep,U\}}(m_{j(k)})$ have equal weight in the construction of each of the service indices. A software vendor may assign different weights to each of the metric scores, in line with its strategy focus (section 5.1 describes the weights Stabiplan assigns to the metric scores). Finally, of observations can be made. First, the domains of some indices are equal to the domains of the metric scores:

$$\{\pi(m_{j(k)}), \pi(s_{i(x)}), \upsilon(m_{j(k)}), \upsilon(s_{i(x)}), \kappa(m_{j(k)}), \kappa(s_{i(x)})\} \subseteq [0, 1]$$

In other words, when the value of, for example, $\pi(s_{i(x)})$ for a service $s_{i(x)}$ 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.

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

Obviously, the change in domains is caused by the constant factor M included in the formulas of all customer-level service indices.

An example of an SKU report can be found in appendix D. In chapter 4 of this thesis, a prototype providing an SKU implementation (including SKU reports) is presented. An example SKU report implementation is explicited also in that chapter, as part of the software prototype description. In the next section, internal and external report applications are detailed.

3.7.4 Report Applications

3.7.4.1 Internal Applications

As depicted in figure 3.4, the SKU report may be of use in three perspectives: the business perspective, organization perspective and the development perspective. The perspectives find their origin in the product software research framework of Brinkkemper and Xu [16] and can be linked to certain management levels. Below, applications of the report in these perspectives are discussed.

Business Perspective (Strategical Management)

The usage and feedback data in the SKU report is a representation of recent service performance, usage and feedback by customers and end-users. This data may help to characterize a software vendor's customer base and support the process of strategical decision-making on the long term. The service indices provide an indication of the status of the software vendor's services in terms of performance, usability and customer usage. Comparing a report with earlier reports, progress in terms of SKU could be observed and monitored.

Company Perspective (Tactical Management)

Potentially, the report is also of use within a software vendor from an organizational perspective. For example, concerning a vendor's marketing department (tactical decision making), SKU reports may support the sales and customer relations process. A deeper and more thorough insight in the characteristics of the vendor's client base may be provided by the reports, which potentially supports the communication in particular situations (i.e., customer complaints) or with particular clients.

For product and release managers, the SKU report may support the process of midterm decision-making concerning the services developed by the vendor. The usage and feedback data in the report may help to identify customer base characteristics, define main issues, plan milestones and divide programmers capacity. Furthermore, based on the information in the SKU reports, a software vendor could decide to offer customized software licenses and contracts, to all or a specific group of customers.

Development Perspective (Operational Management)

The role of usage information and statistics from a development perspective (within a software vendor's development department) is twofold. First, because the usage data may include the description of exception occurrences, frequent analysis of the the SKU reports potentially discloses severe bugs earlier and may help to reproduce bugs, possibly resulting in shorter overall development cycles. Secondly, the service indices defined in section 3.7.3 give the vendor a clear overview of the status of its software in terms of performance, usability and customer usage. Thirdly, the reports give insight in the use frequency of (the methods of) a service. Knowledge about customer wishes and demands could be used well too in the process of software development, possibly resulting in improved incorporation of 'demand-driven' functionality into future upgrades or releases of the software.

Within the support department (also part of operational management), SKU reports may provide a more thorough understanding of, and insight in the types and amounts of service software bugs that occur in the software. Furthermore, the reports contain a clear overview of the customer's (end-user's) configuration, as well as an indication of the appreciation of the software by the end-users.

Furthermore, based on the customer usage, usability and feedback data in the report, training sessions could be adjusted to focus more on often-used functionality, or on functionality on which most negative feedback is given by end-users. Finally, as part of the communication from the software vendor to its customers, the vendor could distribute the customized licenses (defined by tactical management) and contracts to its customers digitally.

3.7.4.2 External Applications

In line with the developments and observations described in the introduction of this thesis, many enterprises are in the process of becoming more responsive and flexible with respect to changes in customer and market demands. With respect to software vendors developing service-based software specifically, this process can be seen as equivalent to the SKU implementation process. By increasing the information awareness of the software vendor, the SKU report contributes to this implementation process.

With respect to both B2B and B2C relationships of a software vendor (depicted earlier in figures 1.2 and 3.4), as well with respect to SKU level increase, the SKU report and its utilization are essential: the report contains service usage and feedback knowledge which may be key to management decisions (at strategical, tactical or operational level) of the enterprise that has licensed the software.

While the software vendor controls the level of service usage and feedback knowledge sharing to external software vendors and enterprises, a high level of knowledge sharing may strengthen its relationship with these companies. Similarly, the software vendor may strengthen the relationship with both customers and end-users by providing specific configuration parameters, tips and tricks, company news, etc. (as visualized by the SKU-SSN model).

Cisco Systems, a company known as a networked, virtual enterprise (a very mature extended enterprise), shares a lot of (for example) procurement and sales information and knowledge to its suppliers and customers [17; 46]. Furthermore, the company has standardized the way it communicates knowledge with its external relations (customers, suppliers, etc.). However, one should note that companies similar to Cisco, will not share their core knowledge and technologies [46].

Different types of software vendors can be identified based on their service knowledge utilization. These types are defined in the next section. Each software vendor type is characterized with a number of corresponding factors.

3.8 SKU Characterization

3.8.1 Introduction

Many factors influence and determine the characteristics of an enterprise. Specifically to software vendors developing software in this (network era) and the next era, in which the sharing of software usage and feedback knowledge is essential in order to be (and stay) adaptive and responsive to changes in customer and market demands, a number of vendor types can be identified, each with typical properties and characteristics. In section 3.8.2, these types are described in terms of SKU.

As already mentioned in section 3.7, Mendelson and Ziegler [75] argued that *information awareness* (knowing what is going on in the surrounding world) is becoming increasingly important for enterprises. Recognizing the importance of information integration for enterprises, Evgeniou [36] presented a framework typifying enterprises in terms of flexibility and visibility. Illustrated in figure 3.5, Evgeniou defines four types of enterprises.

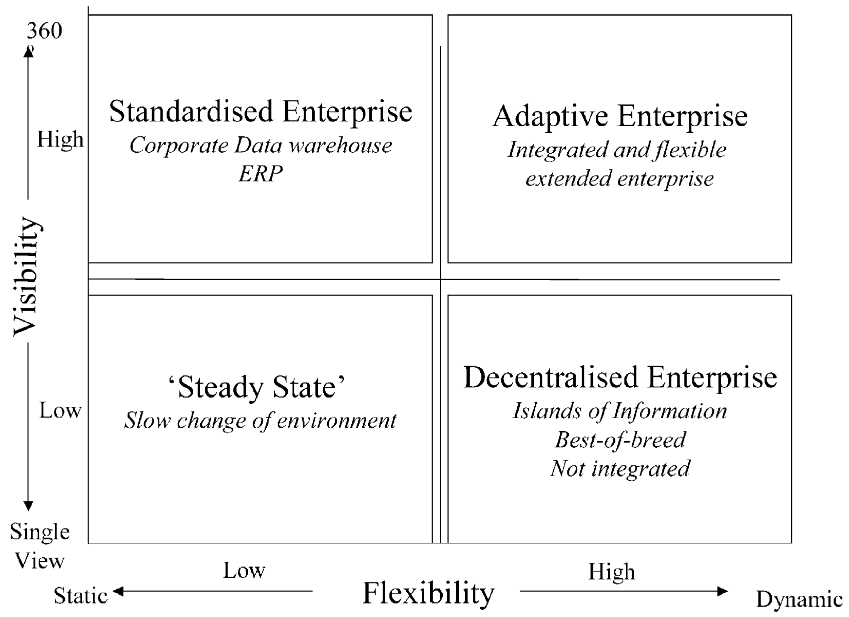
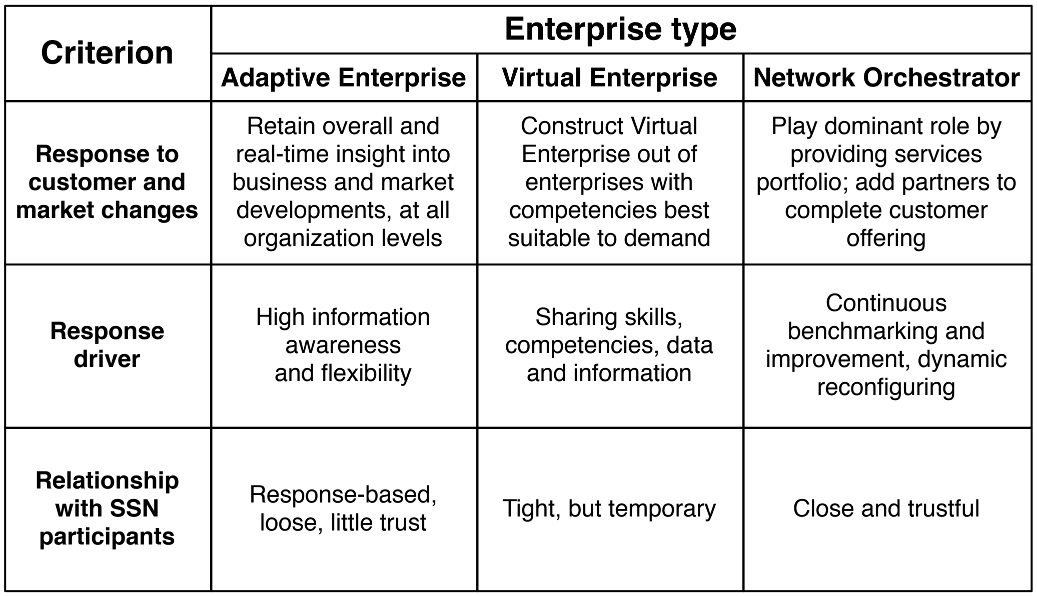


Figure 3.5: *Enterprise types defined by Evgeniou [36]*

Characterized as the state to which both standardized and decentralized enterprises should aspire, becoming both standardized in terms of visibility and decentralized in terms of flexibility, the Adaptive Enterprise implies high information awareness and high flexibility. The SKU characterization model presented in section 3.8.2 resembles Evgeniou's model to certain extent. For example, the horizontal axis of the characterization model is also represents flexibility and the characterization model also defines an 'Adaptive' characterization.

Research has developed a number of enterprise types that, based on certain characteristics, roughly correspond with the Adaptive Enterprise. One should note that, to some extent, the Adaptive Enterprise characterization is equivalent to the network orchestrator described in [17; 46] and to the concept of the Virtual Enterprise (VE) [91; 94; 114]. Specifically focusing on software vendor enterprises, these enterprise types can be compared on characteristics particularly relevant in the context of SKU and mainly related to response to changes in customer and market demands.

While all enterprise types play a dominant role in their software supply network and try to be highly responsive to participants in their SSN, small differences between the enterprise types can be identified. A comparison of the three enterprise types is detailed in figure 3.6[13].



Criterion	Enterprise type		
	Adaptive Enterprise	Virtual Enterprise	Network Orchestrator
Response to customer and market changes	Retain overall and real-time insight into business and market developments, at all organization levels	Construct Virtual Enterprise out of enterprises with competencies best suitable to demand	Play dominant role by providing services portfolio; add partners to complete customer offering
Response driver	High information awareness and flexibility	Sharing skills, competencies, data and information	Continuous benchmarking and improvement, dynamic reconfiguring
Relationship with SSN participants	Response-based, loose, little trust	Tight, but temporary	Close and trustful

Figure 3.6: *Comparison of the Adaptive Enterprise [36], Virtual Enterprise [94] and the Network Orchestrator [17]*

3.8.2 SKU Characterization Model

Related to the enterprise types defined by Evgeniou and the enterprise type comparison presented in section 3.8.1, in this section, four software vendor types are presented in terms of SKU. Each type corresponds to a particular SKU level and flexibility level. One should recognize that the SKU characterization model does not define scientific enterprise maturity levels, but provides a set of enterprise types to identify and recognize software vendors in terms of their service knowledge utilization.

The software vendor types of the SKU characterization model are characterized based on two criteria: service knowledge utilization level and flexibility. A number of factors that determine the SKU level and the flexibility of a software vendor can be defined.

SKU Level

Service Knowledge Gathering The level of service knowledge gathering is directly related to the level of service knowledge utilization. More specific, a software vendor that has realized means to gather service knowledge (for example with a service usage and feedback logging system implementation and SKU reports) and is ready to utilize this knowledge, has a high SKU level.

Service Knowledge Significance A software vendor will not improve its flexibility by gathering software (usage) knowledge alone. In order to leverage from service

[13] In the context of this thesis research, the 'Network Orchestrator' compared in this figure specifically is a 'knowledge services' network orchestrator [17].

knowledge, a software vendor will have to be prepared to implement and apply the gathered service knowledge in and to its existing infrastructure and processes. In other words, the higher the importance and the actual deployment of the gathered service knowledge, the higher the flexibility and service knowledge utilization. For example, a software vendor that analyzes its SKU reports regularly and has defined actions that should take place when specific conditions are met, continuously shares its service knowledge with its external partners and concurrently receives service knowledge from its partners, has a higher SKU than a software vendor that has implemented means to gather service knowledge, but considers service knowledge as insignificant — and does not actively utilize the knowledge.

Flexibility

Software Supply Network The environment a software vendor is operating in, influences the flexibility of a software vendor. For example, regulations imposed by a local government may limit the flexibility of a software vendor. To the contrary, subsidies granted by the government or a public body may assist starting software vendors in hiring employees and building their position in the software supply network they are part of. This software supply network is of influence to the flexibility of the vendor. A precondition for leveraging from service knowledge and resources of external partnerships in the software supply network is a trustful relation between the software vendor and these external partnerships. Such a relation is required for sharing service knowledge (e.g. usage and feedback data). So while a software vendor may be part of a very large and extensive software supply network, if the relations with external partners in the network are mistrustful, the software vendor may be less flexible than a software vendor located in a small software supply network that has a few trustful external relationships. Ultimately, the relation of a software vendor with other participants in the software supply network (both business to consumer as business to business) is such that all participants are ready to share software (usage) knowledge, on a regular basis, and to each related participant (partner).

Time To Market Release times of a software vendor (and the response times of the vendor in general) influence the flexibility of the software vendor. Especially in the frequently changing markets in which software vendors operate nowadays, a short time to market contributes to a high software vendor flexibility and responsiveness. For example, a software vendor that has a short TTM is able to incorporate unforeseen requirements — caused by changes in customer and market demands — during the process of software development relatively easy, without completely disrupting a project (which, in turn, contributes to the flexibility and responsiveness of the vendor). Contrarily, a software vendor with a long time to market executes long-term and complex projects, is less able to incorporate unforeseen changes in these projects and is therefore limited in its flexibility and responsiveness to changes in customer and market demands.

Having detailed the factors that determine the SKU level and flexibility of a software vendor, a software vendor's ability to respond to changes in market and customer demands can be seen as a combination of both the SKU level and flexibility of a software vendor. In other words, a software vendor developing service-based software is responsive when it has a high SKU level (the software vendor has implemented means to gather and share service knowledge) and a high level of flexibility (the software vendor leverages from, or is not significantly limited by environmental factors or local regulations). The definition of responsiveness in the context of this thesis research was already given in section 3.4.2.

Based on the factors described above, four SKU types can be defined. See figure 3.7.

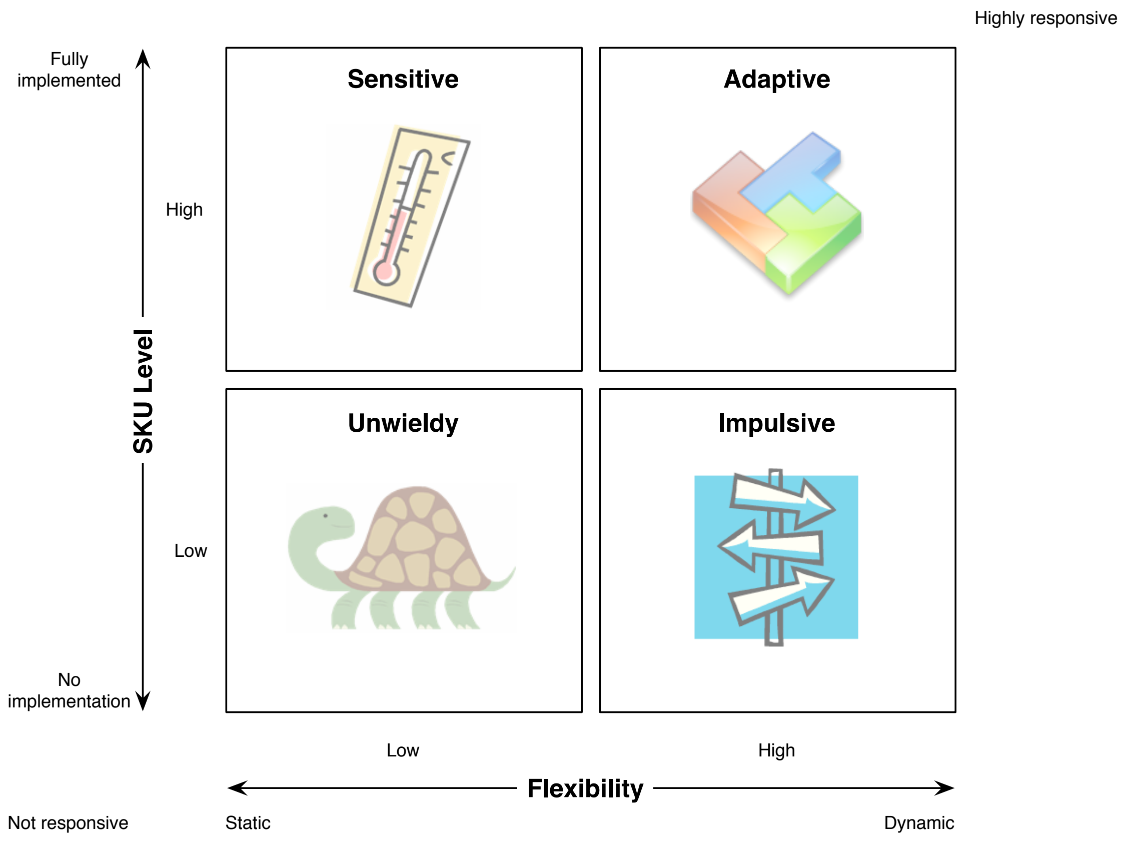


Figure 3.7: *SKU characterization model*

Unwieldy Unwieldy software vendors are practically unable to respond to changes in customer and market demands, for example because of imposed regulations or a lack of resources that limit the software vendor in its flexibility. Unwieldy software vendors have no realized means for gathering (let alone sharing) service usage and feedback knowledge, and the role of this knowledge within the organization is of low importance. Hence, the SKU level of this type of software vendors is low. Software vendors classified as 'unwieldy' might operate in a network of enterprises and have a number of (trustful) important enterprise relations — but not with respect to service knowledge usage sharing, however. Furthermore, unwieldy software vendors often execute long-term, complex projects and have a long time to market. Examples of such vendors are large governmental organizations or institutes (like the software development department of a tax registration agency, for example).

Impulsive Impulsive software vendors are very responsive to new market trends and customer demands. The impulsive software vendor mainly depends on, and develops software for one or two major customers. When one of these major customers presents a set of new software requirements, the software vendor accepts the new demand relatively easy, after a short period of negotiations. The software vendor has a high flexibility level because of the trustful and 'intimate' relationships with its external partners (main customers) in the software supply network it is located in, and because of its short time to market: while projects may be complex and long-term, the right amount of human resources needed to complete all projects is determined by the vendor easily, since the software vendor has a limited number of customers and projects. Since the software vendor has not implemented a way to log, monitor and extract service usage and feedback knowledge, the vendor does not have any objective data to support its arguments (for not implementing or changing a particular feature, or fulfilling a particular demand, for example) or maintain its relation with external enterprises (B2B) or important customers (B2C). Often, impulsive software vendors are relatively small companies with ten to fifty employees.

Sensitive While the sensitive software vendor does log the usage and feedback data of its (service-based) software of a small number of customers — the vendor has a high SKU level — the vendor is still quite static. In other words, most relations of the vendor with customers and external partners in the software supply network the vendor is located in, (still) are quite mistrustful and the time to market of the software vendor (still) is long. The vendor is ready to fill its usage and feedback database with data from more participants in its software supply network, extract service knowledge from this database, and to decrease its time to market by actively deploying this knowledge within its organization. However, the relations with other participants in the software supply network of the software vendor should be improved, in order to do so. Sensitive software vendors are finalizing their SKU strategy and by investing into their external relations, sensitive software vendors should be making progress in the process of becoming an adaptive software vendor.

Adaptive The adaptive software vendor has successfully implemented a way to log and monitor the usage and feedback of its (service-based) software. Furthermore, the vendor has implemented infrastructure and processes to successfully utilize the service knowledge extracted from the service usage and feedback data. For example, the vendor has defined a set of actions (with corresponding conditions) that are performed when the (usage and/or feedback data points out that those) conditions are met. Moreover, in order to maintain a dominant role in its SSN and keep relationships with the external software vendors or customers in its SSN trustful, the software vendor purposely shares its service knowledge (possibly in the form of an SKU report) with these partners. Both business relations (both B2B and B2C) are such, that service usage and feedback data is synchronized with the software vendor frequently. Finally, the vendor has a relatively short time to market. By utilizing service knowledge effectively, changes in customer demands are expected and are quickly incorporated in the vendor's projects, at an early stage. In short, the adaptive software vendor has a high level of service knowledge utilization, has a high flexibility level and is therefore highly responsive to changes in customer and market demands.

While numerous level-based process improvement approaches exist, none provides adequate descriptions or guidelines for improving the process of responding and adapting to changes in customer and market demands, based on the usage and feedback of service-based software. For example, the Capability Maturity Model Integration for Development (CMMI-DEV) [101] approach mainly focuses on improving the development processes of product software, while a proposal for a variant focusing on services [81] does not address the area of service usage and feedback knowledge — again, in order to improve the process of responding and adapting to changes in customer and market demands — either.

3.8.3 Identifying the Vendor's SKU

Based on facts and observations gathered during the case study, this section describes the situation at Stabiplan B.V., in the context of SKU. The observations are divided over the areas 'Service Knowledge', 'Organization' and 'Communication', since knowledge about these topics is necessary in order to determine the responsiveness, service knowledge utilization level and flexibility of a software vendor (as motivated earlier).

First, the situation at Stabiplan with respect to the concept of service knowledge is described. Secondly, Stabiplan's organizational and management structure with respect to its software development is explicited. Thirdly, the way the vendor communicates internally (within its organization) and externally (with the participants of its software supply network) is described. Finally, the vendor's current status with respect to service knowledge usage (SKU) is presented.

3.8.3.1 Service Knowledge

The software vendor has no data on the usage of its product and service software. As described elsewhere in this section, based on the issues reported to the help desk, the software vendor gets little indication of how frequent the software is used ('bug is usage'). Furthermore, based on information of the 'Orders and Licenses' department administration, Stabiplan has insight in which editions, modules and languages are *licensed*, but not how frequently they are *used*.

As a result, the software vendor has little to no knowledge concerning the usage of its software available. While usage information would be welcome to support several meetings and discussions ('Should we remove or add this feature? Is it (or will it) still be used?'), the software vendor only logs the usage of the software on a very basic level: major and severe exceptions are written in a text file on disk for debugging and support purposes.

Stabiplan has little to no information about how end-users experience and appreciate the software. Based on support and help desk e-mails, training courses, exhibitions and the oral feedback of pilot customers, the vendor gets an indication of how its customers and end-users experience and appreciate its software products and services.

3.8.3.2 Organization

This section describes Stabiplan's organizational and management structure based on the management perspectives defined by Brinkkemper and Xu [16] and embedded in the SKU-SSN (figure 3.4).

Development and Organization Perspective

Three teams within Stabiplan are managing the development and release of the company's software.

- Product Management team
- Release Management team
- Project Managers team

First, the Product Management team consists of two product managers. One is responsible for national (Dutch) product management, the other focuses on international product management. While national product management is concerned with the fundamental features of the software, international product management focuses on language, culture and localization issues. Apart from the two product managers, the team consists of three key project managers. Product Management determines the main milestones (Windows Vista support, for example) and intervenes in the case of a major issue.

Secondly, the Release Management team consists of both product managers, as well as the CEO and a number of key project managers (other than these in the Product Management team). The Release Management team maintains a list with major issues and ensure that these issues are solved before a particular date or milestone. In other words, Release Management assigns features and issues to future releases. When an implementation deadline is exceeded, the Release Management team investigates the consequences and provides resolution suggestions.

Thirdly, the Project Managers team consists of the project managers of the various development teams of Stabiplan. While the project managers mainly communicate via backlogs, the managers meet regularly to divide issues and discuss implementation problems. These problems may be common, but may also occur when one development team is dependent on another. This dependency may occur in the context of database conversions or documentation writing.

Since very little to no software usage or feedback data is available, no recent knowledge based on this data can be used during the numerous meetings of the management teams.

Business Perspective

Observing figure 1.5, Stabiplan's management board at the top of the chart ('Management') manages the organization with respect to the business perspective. Stabiplan's management board consists of the vendor's CEO, and sales director. Based on information from lower organization departments, external partners and major customers, the management makes major, long-term decisions concerning the future of the organization and its software. Furthermore, the management board initiates and maintains business relations with several external partners. Again, since very little to no software usage or feedback data is available, no recent knowledge based on this data can be used during the numerous meetings of the company management board.

3.8.3.3 Communication

As motivated in previous chapters of this thesis, communication between a software vendor and its external partners, customers and end-users in its software supply network is of great importance. This section illustrates the situation at Stabiplan, based on facts gathered during the case study research.

Internal Communication

The management teams described before (section 3.8.3.2) meet and discuss frequently. Most decisions are taken after a short discussion. However, discussions concerning the usage of the software or customer demand and appreciation often take a longer time. Decisions with respect to these objects are experienced as difficult to take. A main cause of this experience can be identified: the software vendor does not log or monitor the usage of its software by its customers and end-users. Therefore, no usage and feedback data (and consequently, less knowledge about these subjects) is available to serve as an objective source supporting management meetings and discussions. Especially during interdepartmental meetings, the differences in points of view of different departments is experienced. For example, a salesman may argue a particular feature has to be implemented in the next software release because a large customer has requested it, while a software developer may object because of implementation complexity of this feature, or because of the implementation priority of other (possibly often-requested) features. This decision difficulty is also experienced on a more operational level: during their SCRUM meetings, developers and project leaders frequently discuss about the addition or removal of a particular feature. Because no recent or updated data concerning software usage and feedback is available, such decisions are experienced as difficult to take.

External Communication: Business to Business (B2B)

Stabiplan delivers its main software, StabiCAD, to many large external building services enterprises. Furthermore, the vendor cooperates with a number of companies. For example, Stabiplan cooperates with foreign enterprises that distribute the StabiCAD software package to foreign markets. Furthermore, Stabiplan cooperates with several external software vendors, or software departments within external organizations, to realize import and export compatibility between different software packages. Stabiplan has realized export functionality to LISE, a French software package that is used in France to acquire building certifications and licenses, for example. Stabiplan also cooperates with Dutch fire departments, to realize compatibility between StabiCAD drawings and route navigation software used by these departments to localize and identify buildings on fire. The software vendor has realized several other import and export modules, for example to provide compatibility with facility management and cost calculation packages.

By means of the CADsymbols.nl website, Stabiplan maintains B2B relations with a large number of manufacturers of engineering materials (see section 1.5.3). With this website, Stabiplan offers these manufacturers a platform to publish (symbols of) their materials, while ensuring a steady (and updated) symbol input flow to its CAD software, making StabiCAD the de facto standard for draftsmen in the building services industry.

Finally, Stabiplan maintains a tight relationship with PeoplePower, a small company that provides CAD drawing (management) services. PeoplePower is frequently involved in pilot testing Stabiplan's StabiCAD package, and provides Stabiplan with in-the-field test results.

Currently, Stabiplan practically has very little to no knowledge about the performance, usability or usage of its CADsymbols.nl website or its StabiCAD software: no relevant data is logged. Consequently, no knowledge about the usage of the website (for example, the number of downloads of a particular symbol, or within a particular group of symbols) or the CAD software is shared with the engineering materials manufacturers or other external parters. However, in general, the relation between Stabiplan and its external partners can be characterized as trustful.

External Communication: Business to Consumer (B2C)

Stabiplan's customer communication mainly takes place around the departments, events and media that are listed below.

Support and Help Desk The support and help desk department within Stabiplan is the department that communicates most with the customers and end-users. When problems occur while using the software, customers contact this department via a support web site, telephone, mail, or fax. In case of a basic issue, it is handled by the 1st line help desk. Complex issues are redirected to the 2nd line and in the case of very complex issues, the 3rd line help desk is contacted.

While this department communicates very frequently with end-users of the software vendor, and people employed at the department have a clear sense of what are the most frequent and severe bugs and problems in the software, employees do not know for sure how and how frequent users use the vendor's software. Based on the issues reported to the help desk, the software vendor gets a small indication of how (frequent) the software is used — 'when an issue is reported, the software is used' —, but on the other hand, a customer rarely reporting bugs may still use the software very intensively and a customer reporting a bug once a week may still only use a very small part of the functionality the software package offers.

Mailings On a frequent basis, Stabiplan sends its customers a newsletter. This newsletter is published in two ways: via e-mail and traditional mail. The e-mail newsletter contains an agenda with upcoming events and exhibitions, and an overview of the most important events happened since the last e-mail newsletter. The newsletter sent via traditional mail is a booklet containing all the information in the e-mail newsletter, plus a number of success stories concerning the usage or purposes of StabiCAD. This form of communication is mainly one-way, and the vendor is not able to extract valuable information from this communication process.

Training Courses Stabiplan organizes training courses for both beginning and advanced draftsmen. While the goal of the starters coursers mainly is to get familiar with the Stabiplan software, students participating in the advanced courses might provide feedback about their software usage and experience. If applicable, Stabiplan trainers channel this feedback — which also may contain a bug report — to the Support and Help desk department.

Exhibitions For Stabiplan, Exhibitions are important events were the company presents itself to the market it is serving. At the larger exhibitions, new customer relations are founded and existing relations are strengthened. Furthermore, exhibitions represent points in time at which new (a version of) software is released. Customers and end-users take notion of, or request new functionality.

Pilot Customers Pilot customers are invited to test new major versions of the StabiCAD software (for example version 8) or newly added functionality. While the communication with customers during pilot testing is mainly about new features, feedback about old usage situations might be received.

Summarizing this section, Stabiplan does not actively gather software usage and feedback data. Consequently, no knowledge based on this data is shared with business partners or customers.

3.8.3.4 Conclusion

Considering the status of SKU at Stabiplan, based on the situation descriptions in this chapter, a lot can be improved. This is expressed by the characterization model depicted in figure 3.8. Stabiplan is located at the center bottom of this model. This implies that the vendor is considered reasonably flexible, but immature in terms of its SKU implementation.

The communication between Stabiplan and its customers is still quite manual and old-fashioned: to a certain extent, the software vendor is aware of the demand of its customers via its help desk and pilot customer initiative. Stabiplan evaluates this demand once in a time. The software

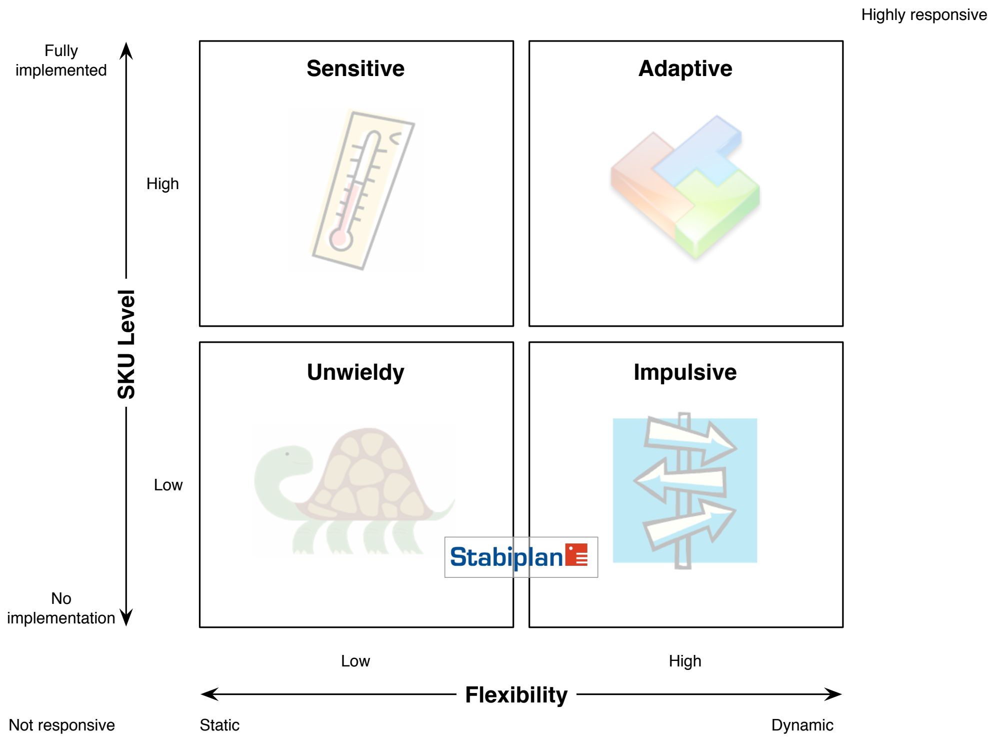


Figure 3.8: *Characterization of Stabiplan's SKU*

vendor's time to market for the new major version of its StabiCAD software (StabiCAD 8) is high (more than 2,5 years between development initiation and final release), but this is mainly due to the fact that it is the first new major release of the software. It is expected that because of the renewed architecture of StabiCAD 8, future major releases will have a much lower time to market. However, no software usage and feedback data is gathered or analyzed at Stabiplan. While the role of service knowledge within the organization is considered significant, no service knowledge is actually deployed or utilized in a structured way. As a result, the vendor is qualified as having a low service knowledge utilization level.

The relation between Stabiplan and its external partners can be characterized as trustful. Stabiplan cooperates with numerous external (foreign) software vendors or departments in order to realize compatibility between the StabiCAD software and the software of these external vendors. Furthermore, a small number of Stabiplan's customers are involved in the process of pilot testing the vendor's CAD software and provide Stabiplan with pilot test results. However, Stabiplan is not yet willing to share usage and feedback knowledge with external partner enterprises (B2B) or clients (B2C). Therefore, the vendor can be qualified as reasonable flexible. Concluding, the software vendor can be qualified as moderately responsive to changes in market and customer demands.

3.9 Summary

This chapter introduces and describes the concept of service knowledge utilization. Except for the software prototype, this chapter describes all developed design research artifacts. The SKU concept is introduced based on a comparison of component-based and service-based software, a comparison of the life cycles of both software types, and the concept of C-CCU. This chapter describes the SKU process model and the SKU in a software supply network model, as well as the SKU report and the SKU characterization model. As part of the SKU report description, the construction of the service indices is also detailed in this chapter. Finally, an identification of the case study software vendor in terms of its service knowledge utilization is presented.

Chapter 4

Nuntia

This chapter is considered confidential. Therefore, this chapter is not published. Please contact Henk van der Schuur at hwschuur@cs.uu.nl for more information.

Chapter 5

Results and Conclusions

This chapter presents conclusions that can be drawn based on research performed. Validation results of the artifacts developed as part of this thesis research are presented. Furthermore, the hypotheses formulated in chapter 2 are validated. Finally, the research questions formulated in the introduction of this thesis are answered.

5.1 Validation Results

Table 2.1 details which research outputs are validated by which research methods. In this section, the results of the research validation process are described.

5.1.1 SKU Characterization Model

Experts were asked to give their opinion about the SKU characterization model. Specifically, they were asked to confirm both axes (SKU Level and Flexibility) that potentially contribute to the diagonal axis (Responsiveness). Both product and project managers considered the flexibility of a software vendor tightly related to the responsiveness of a vendor. When a software vendor is flexible towards changes in customer and market demands or software performance and usage, for example by fixing unexpected bugs quickly, a vendor is able to (re)act quickly and can therefore be considered responsive, they reasoned.

Experts also confirmed that the SKU Level of a software vendor is of influence to a vendor's responsiveness. Lead developers and project managers consider a mature SKU implementation as a way to make informed decisions, for example concerning human resource management (based on service usage and feedback knowledge, dedicate employees to the software modules that need most attention) or software maintenance (set bug fix priority based on service knowledge, number of end-users a bug occurs). However, experts indicated that their organization would have to invest into implementation of a SKU process before being able to benefit from a responsiveness increase, which requires 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.

5.1.2 Software Prototype

While the software usage data gathered may be of great value to a software vendor and its partners, SKU tools (like the Nuntia prototype) should not be integrated with target software at any price. The usage tracing and the receiving, gathering and storing of usage data uses resources in the form of CPU time and storage space. Even though the performance loss caused by the Nuntia prototype was considered negligible, particularly the usage of CPU time by software usage data gathering software is notable by the end-user: the target service may be experienced as 'unusually slow'.

Section **??** details the setup and results of the performance testing that was done to validate the usability of the prototype once implemented in target software. The results of the performance test sessions are in line with the opinion of 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 the vendor's 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.

This observation is recognized and acknowledged: as described in chapter 6, future research includes work on the integration of the service knowledge utilization process into a software vendor's existing infrastructure and processes.

5.1.3 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 throughput, availability and reputation metrics, experts agreed that all metrics proposed should be included in the report. They denoted that the throughput and availability metrics particularly are of importance with respect to process-centric services. Unlike the vendor's service software (which can be characterized as a public enterprise service), process-centric services have to meet a specific demand, and therefore guarantee a particular level of throughput or availability.

Project and product managers also indicated that only proper and serious feedback will be useful. Therefore, they proposed to include the reputation metric primarily with pilot customer tests. Furthermore, the availability metric was considered as less useful because the low availability score may not be caused by the vendor's software and may not be the responsibility of the software vendor, 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. The report will be generated once a month.

5.1.4 Hypotheses

The case study validation results show that Nuntia is expected to contribute to a software vendor's responsiveness to service performance and usage changes. The vendor has decided to integrate Nuntia with its live software in the near future because of the negligible performance loss this integration entails and because of the valuable information that is provided by the SKU report.

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 improve its responsiveness based on the gathered knowledge. The vendor expects to base 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 the aspect-oriented programming technique) enables easy integration with target software. Since Stabiplan did not actively gather knowledge about its software, and the SKU prototype was successfully integrated with the vendor's service software, Stabiplan indicates that its SKU level has increased. Moreover, the vendor denoted that the prototype contributed to this SKU level increase.

Hence, we consider the hypotheses confirmed on the long run.

5.2 Conclusions

This thesis research shows that the concept of service knowledge utilization is an approach that potentially improves the responsiveness of software vendors. By using this approach, vendors can make informed decisions with respect to their software requirements management and software maintenance processes. Service usage and feedback data that is considered valuable and useful by software vendors, can be gathered real-time using aspect-oriented programming techniques. It is demonstrated that service performance, usability and usage can be expressed by a set of quality of service metrics, of which the values are based on gathered service usage and feedback data. This data can be visualized and summarized in a report that supports a software vendor's management teams in extracting software knowledge and making informed decisions.

Even though both the SKU concept and SKU report are considered useful and the case study results are promising, it can be concluded that gathering software knowledge will not improve the vendor's responsiveness *on its own*: a software vendor will have to integrate the SKU process into its existing processes and infrastructure. Furthermore, some form of change management will be required: employees will have to get used to involve the gathered software knowledge in their daily work.

5.3 Answers to Research Questions

This section answers the research questions that were formulated in the introduction of this thesis.

RQ *How can Continuous Customer Configuration Updating be applied to service-based software and Web services?*

The answer to this question is the concept of service knowledge utilization (SKU). While C-CCU is developed for component-based software — the processes defined in the C-CCU model closely match those of the component-based software life cycle — SKU is applied to service-based software. The concept of SKU is constructed by comparing the life cycles of both software types and has the same goal as the C-CCU model: to increase the responsiveness of a software vendor.

SQ1 *What are differences and similarities between software components and services with respect to C-CCU?*

This question is mainly answered by sections 3.1 and 3.2. In the former section, the definitions of both concepts developed by research are compared. Furthermore, both components and services are compared by a number of criteria. The latter section details a comparison with respect to C-CCU and the phases defined in the C-CCU model: in this section, the life cycles of both component- and service-based software are compared.

SQ2 *What are measurable critical success factors of C-CCU applied to service-based software and Web services?*

The answer to this question is the SKU characterization model presented in section 3.8.2. This model expresses a software vendor's responsiveness in terms of its flexibility (horizontal axis) and service knowledge utilization level (vertical axis). In this thesis research, two factors that determine the location of a software vendor on an axis have been defined for both axes. The critical success factors of the flexibility axis are a vendor's software supply network and its time to market. For the SKU level axis, these factors are service knowledge gathering and service knowledge significance. Because the factors determine a software vendor's location on both axes and consequently determine the responsiveness of a vendor, the factors can be seen as critical success factors of the SKU concept (C-CCU applied to service-based software). The factors are validated by means of the case study performed during the thesis research.

SQ3 *When is C-CCU, applied to service-based software and Web services, implemented successfully?*

The answer to SQ3 is also contained in the SKU characterization model. In section 3.8.2, four software vendor types are defined in terms of their flexibility, SKU level (and consequently, responsiveness): the unwieldy, impulsive, sensitive and adaptive software vendor. The adaptive software vendor can be seen as a software vendor that has implemented the concept of service knowledge utilization (C-CCU for service-based software) successfully.

5.4 Discussion

This thesis research started as an initiative to apply Jansen's C-CCU model to service-based software and web services. While the research questions formulated in this thesis are satisfactory answered and the thesis research results are promising, remarks can be made.

First, a case study was performed in order to validate outputs of this thesis research as part of the design research evaluation phase. Specifically, interviews, expert validation and testing research methods were utilized to gather and validate evidence. On the one hand, the case study is a suitable method to gather evidence from the field: real people are interviewed and updated information is gathered, an actual and realistic image of a company can be formed. On the other hand, one should prevent that too specific and characteristic evidence is gathered, in order to apply research results to comparable companies. This could be done by interviewing experts or gathering documentation from various companies, for example.

Concerning this research, results can be generalized to similar-sized software vendors easily, even though a limited number of experts was interviewed. While more effort will have to be done with respect to process integration and alteration, the concept of SKU may well be of even more importance within larger software vendors. Furthermore, the Nuntia SKU prototype is developed with generic concepts, techniques, goals and parties in mind, and can be implemented at another software vendor without difficulty. Finally, the structure of the SKU report is such that it can be extended effortlessly. For example, newly defined event types and properties, as well as environment information log structures, are automatically included in the SKU report when the report is generated. With respect to future research, more experts will be interviewed and evidence from different software vendors will be gathered.

Secondly, even though both the SKU report as the information it contains are considered useful, the service usability index could be further developed. As described in chapter 3, a lot of work has been done in the usability research area. Therefore, the service usability index could be composed of more typical usability metrics. With respect to service quality and performance, most important metrics are taken into account, as visualized by table 5.1. Concluding, concerning software development and software maintenance, relevant service indices are included in the SKU report. Service indices that represent other aspects of service behavior could also be included. For example, a 'Service Customer Contact Index' could be constructed of metrics that represent the time a software vendor spends on a certain customer. With respect to the events overview that is contained in the SKU report, one should recognize that the quality of the feedback provided influences the quality and usefulness of the overview and the data contained in it. Moreover, since a software developer or software tester probably will provide feedback of a different, more technical type than an end-user of the software, feedback could be categorized in feedback source types. By doing so, feedback of a particular type can be directed to or included in a particular enterprise department, business process or management team.

Thirdly, remarks can be made with respect to the SKU characterization model. While the model is not designed to identify quantifiable SKU maturity levels, the factors that determine a software vendor's SKU type could be described more elaborate and could be validated more thoroughly, in order to justify a vendor's (shift in) SKU type.

Fourthly, as indicated by the experts interviewed and reflected by the SKU characterization model, the success of the research outputs and an SKU implementation in general depends on to which extent a software vendor is prepared to implement and apply these outputs, as well as the gathered service knowledge, in and to its existing infrastructure and processes. For a software vendor, the greatest challenge will not be in the technology area, but in integrating the SKU concept and all its artifacts into the business processes and infrastructure that are already in place and where people are familiarized with. Future research will be done to address possible integration issues (see chapter 6).

This thesis research opened a relatively new research area. In its broadest form, this research area can be identified as research on the concept of 'software knowledge utilization'. The main question of this research area can be defined as 'How can software knowledge be successfully utilized in the life cycle phases of all types of software?' As described in chapter 6, future research will focus on this area.

5.5 Positioning of Research Contribution

This section positions this thesis research contribution in the field of related research. The thesis is positioned for each of the research areas it contributes to: QoS metrics, service management and usage monitoring and aspect oriented programming.

5.5.1 Quality of Service Metrics

In research, QoS metrics are often used or created [62] in the process of web service discovery [90], selection [117] or composition [119]. Furthermore, research is done specifically on the integration of QoS management in service-oriented enterprise architectures [109].

Table 5.1 lists research articles on quality of service metrics and compares them in terms of these metrics. The table visualizes how frequent certain quality of service metrics are used in related research. Comparing related research to this thesis research, two important observations can be made. First, related research frequently includes a 'Cost' (or comparable) metric in the set of quality of service metrics. As a consequence, the quality of a service often is inter alia composed of the purchase and usage cost of a service. Since the main service purchase and usage costs are paid by enterprises and end-users which purchase and use the service software, and since this thesis research is done from a software vendor's point of view, a cost metric is not taken into account in this research. Secondly, as represented by table 5.1, little research is done on the inclusion of a 'Usability' (or comparable) metric as a part of the quality of a service. Contrarily, web usability is evaluated in typical usability research like [112]. Since the case study software vendor of this thesis research develops and maintains service-based software that is controlled by end-users via an user interface, a usability metric is included in the set of service metrics.

While many research initiatives focus on the application of QoS metrics in the processes that take place before the management phase of the service-based software life cycle (see figure 3.3), this thesis research utilizes QoS metrics in the two last phases of this life cycle, management and usage, focusing 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.

5.5.2 Service Management and Usage Monitoring

Huang et al. [52] propose a service management framework, specifically for service-oriented enterprises. While their work is based on a model-driven approach and is designed for service-oriented enterprises, the research and solutions presented in this thesis can also be applied in non-service-oriented enterprises.

Since Microsoft implemented its error reporting mechanism in Windows and discovered that 1% of the bugs caused 50% of all errors [15], a lot of research has been done on this topic. Farrell and Kreger [38] 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. Lazovik et al. [69] propose a framework for planning and monitoring the execution of web service requests against standardized business processes. This planning framework specifically monitors the execution of planned goals against predefined standard business processes and interacts with an end-user to achieve goal satisfaction, while the solution presented in this thesis monitors (traces) services in a broader scope (on performance, usage and feedback criteria) to achieve software quality goals and increase the responsiveness of software vendors to performance and usage changes in their software.

5.5.3 Aspect-Oriented Programming

Since a lot of cross-cutting concerns exist, research on the subject of aspect-oriented programming is done on various areas. Altisen et al. [4] apply the notion of cross-cutting concerns to reactive systems. Tsang et al. [102] utilize AOP to encapsulate real-time cross-cutting concerns such as memory management and thread scheduling. Bruntink [18] investigated the renovation of idiomatic cross-cutting concerns in the embedded control software of ASML's wafer scanners using AOP. Papapetrou and Papadopoulos [84] have researched the more conventional application of aspect-oriented programming: they performed a case study in which an AOP-based logging tool was applied to real-life component-based software. While their application of aspect-oriented programming is similar, our solution is also utilized with service-based software and environments.

Research has also been done with respect to the monitoring, logging and tracing of Web services by using aspect-oriented programming. Bhatti et al. [8] study the execution behavior for various service classes with a formal model they developed for simulating and studying response time of ‘aspectized’ Web services. While their research is limited to service performance measuring (the execution time of Web service classes is measured), our research also studies the usability of and feedback on Web services.

Literature	Metric[a]							
	Reliability	Throughput	Latency	Accuracy	Availability	Usability	Reputation	Cost
Ran [90]: *A Model for Web Services Discovery With QoS*	+	+	+	+	+		+	
Yu et al. [118]: *Efficient algorithms for Web services selection with end-to-end QoS constraints*			+[b]		+			+[c]
Bochmann et al. [108]: *Introducing QoS to Electronic Commerce Applications*		+	+[d]		+			
van Moorsel [106]: *Metrics for the Internet Age: Quality of Experience and Quality of Business*	+	+	+[c]	+[e]	+	+[f]	+	
Zeng et al. [119]: *Quality Driven Web Services Composition*	+		+[g]		+		+	+[h]
Yu and Lin [117]: *Service selection algorithms for Web services with end-to-end QoS constraints*	+		+[c]		+			+
Mani and Nagarajan [73]: *Understanding quality of service for Web services*	+	+[i]	+[h]	+[j]	+			
Kalepu et al. [62]: *Verity: A QoS Metric for Selecting Web Services and Providers*	+	+	+	+	+		+	+[g]

[a] Underlined metrics are part of one or more service index compositions of this thesis research (see chapter 3).
[b] Referred to as ‘Execution time’.
[c] Only the initial cost of a service (price) is taken into account.
[d] Referred to as ‘Response time’.
[e] Referred to as ‘Error rate’.
[f] Referred to as ‘Quality of Experience’.
[g] Referred to as ‘Execution duration’.
[h] Referred to as ‘Price’.
[i] Contained in the metric ‘Performance’.
[j] Referred to as ‘Integrity’.

Table 5.1: *Quality of Service metrics in related literature*

Chapter 6

Future Research

While the results of this research are promising, further research is needed in various areas. This chapter describes future research in each of these areas.

6.1 Integration

Research has to be done to more tightly integrate the process of service knowledge utilization into a software vendor's existing infrastructure and processes. For example, exceptions of which a vendor knows they really should not occur, could be automatically sent to the vendor's help desk support system or development bug database. Within this context, a structured mechanism with which actions can be defined that are performed based on usage and feedback data, should be developed. These actions could be defined in the form of expressions which could be expressed in a structured language like XML. To accomplish this tight SKU integration within an organization, a SKU business process alteration or integration method could be defined that is based on survey or expert interview results. Furthermore, tools that orchestrate this integration and business process alteration could be developed.

6.2 Scalability

The areas of data mining techniques and software tomography [12] have to be researched. Data mining [47] is required to extract sophisticated statistics, whereas software tomography keeps performance loss of the target software caused by the process of software usage and feedback data gathering negligible under heavy use circumstances. Future research should also focus on the data mining of large service knowledge repositories to ensure scalability of the SKU implementation. Structurally increased scalability could be accomplished by an extensive literature study in the area of data mining and software tomography.

6.3 Reporting

While the SKU report generation functionality implemented in the software prototype developed during this thesis research is validated by the case study company and is mature enough to support decision making processes of a software vendor, reporting can be improved. For example, the report may be generated on several levels of detail and complexity, depending on the application of the report and configurable by the software vendor. Furthermore, new service indices (and corresponding metrics) may be included in the SKU report. Finally, the report could more resemble the architecture of the software it is generated for. This could be realized by performing a literature study on automated software structure and architecture analysis and implement techniques used in these areas in the SKU report generation tool.

6.4 Development Assistance

Data from the usage and feedback database can be visualized to developers and software maintainers in a more comprehensive and effective way. This data can be used to visualize heavily-used, critical code, for example. Software developers could be presented with a visualization of code that enables a developer to understand how a piece of code behaves in practice. The visualization must be insightful, yet lay a small claim on the already scarce screen estate of a software developer. Furthermore, the context of the code must be visualized as well, to disambiguate the multiple uses a code fragment has. In this respect, research has to be done in feedback visualization techniques and software maintenance process alteration. A code visualization plugin that provides only the information that developers find useful in specific situations can be developed. Because developers or project managers may resist changes in software development or software maintenance processes, they have to be convinced of the value that a feedback data visualization plug-in may bring to a software vendor's integrated development environment[1].

6.5 Characterization

More research may be done on the area of SKU characterization. As mentioned in chapter 3, the SKU characterization model that is part of this research can be used to identify and recognize software vendors in terms of their service knowledge utilization. However, only few factors that identify a software vendor in terms of SKU and determine a vendor's SKU type have been defined, and new factors may be researched. The agility ('agile-ness') of the software development process of a software vendor might be a new factor, for example. Furthermore, research has to be done on both quantifiability and measurability of the factors already defined, as well as potential additional factors. Both aspects could be established by conducting a survey among software vendor's CEOs and projects managers.

[1] This persuasion is part of the business process alteration described in section 6.1.

Bibliography

[1] Russell Ackoff. From data to wisdom. *Journal of Applied Systems Analysis*, 16:3–9, 1989.

[2] Paul Allen and Stuart Frost. *Component-based development for enterprise systems: applying the SELECT perspective*. Cambridge University Press, New York, NY, USA, 1998.

[3] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services: Concepts, Architecture and Applications*. Springer Verlag, 2004.

[4] K. Altisen, F. Maraninchi, and D. Stauch. Aspect-oriented programming for reactive systems: Larissa, a proposal in the synchronous framework. *Science of Computer Programming*, 63(3):297–320, 2006.

[5] Nate Anderson. Tim Berners-Lee on Web 2.0: "Nobody even knows what it means". `http://arstechnica.com/news.ars/post/20060901-7650.html`. Ars Technica, LLC, September 2006.

[6] Autodesk, Inc. `http://www.autodesk.com`.

[7] Carol M. Barnum. *Usability Testing and Research.* Allyn & Bacon, Inc., Needham Heights, MA, USA, 2001.

[8] Muhammad Usman Bhatti, Samir Youcef, Lynda Mokdad, and Valrie Monfort. Execution Time Analysis of Aspectized Web Services. In *Second International Conference on Internet and Web Applications and Services*, page 30. IEEE Computer Society, 2007.

[9] Barry Boehm. Get Ready for Agile Methods, with Care. *IEEE Computer*, 35(1):64–69, 2002.

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

[11] David Booth, Hugo Haas, et al. Web Services Architecture. `http://www.w3.org/TR/ws-arch/#whatis`. Web Services Architecture Working Group, World Wide Web Consortium.

[12] Jim Bowring, Alessandro Orso, and Mary Jean Harrold. Monitoring Deployed Software Using Software Tomography. *SIGSOFT Software Engineering Notes*, 28(1):2–9, 2003.

[13] Tim Bray, Jean Paoli, et al. Extensible Markup Language (XML) 1.1 (Second Edition) - Origin and Goals. `http://www.w3.org/TR/xml11/#sec-origin-goals`. World Wide Web Consortium.

[14] Tim Bray, Jean Paoli, et al. Extensible Markup Language (XML) 1.1 (Second Edition). `http://www.w3.org/TR/xml11`. World Wide Web Consortium.

[15] Harry Brelsford, Michael S. Toot, Karishma Kiri, and Robin Van Steenburgh. *Connecting to Customers*. Microsoft Press, Redmond, Washington, February 2002.

[16] Sjaak Brinkkemper and Lai Xu. Concepts of product software: Paving the Road for Urgently Needed Research. *European Journal of Information Systems*, 16(5):531–541, 2007.

[17] John Seely Brown, Scott Durchslag, and John Hagel. Loosening up: How process networks unlock the power of specialization. *The McKinsey Quarterly: Risk and Resilience*, 2, September 2002.

[18] Magiel Bruntink. *Renovation of Idiomatic Crosscutting Concerns in Embedded Systems.* PhD thesis, Faculty of Electrical Engineering, Mathematics and Computer Science, March 2008.

[19] Steve Burbeck. The Tao of e-business services. `http://www.ibm.com/developerworks/webservices/library/ws-tao/`. Emerging Technologies, IBM Software Group, 2000.

[20] L.F. Capretz. Y: A New Component-Based Software Life Cycle Model. *Journal of Computer Science*, 1(1):76–82, January 2005.

[21] Stefano Ceri, Piero Fraternali, and Stefano Paraboschi. XML: Current Developments and Future Challenges for the Database Community. In *EDBT '00: Proceedings of the 7th International Conference on Extending Database Technology*, pages 3–17, London, UK, 2000. Springer-Verlag.

[22] Roberto Chinnici, Jean-Jacques Moreau, et al. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. `http://www.w3.org/TR/wsdl20/`. World Wide Web Consortium.

[23] Vidyanand Choudhary. Software as a Service: Implications for Investment in Software Development. In *HICSS '07: Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, page 209a, Washington, DC, USA, 2007. IEEE Computer Society.

[24] Luc Clement, Andrew Hately, Claus von Riegen, and Tony Rogers. UDDI Version 3.0.2 — UDDI Spec Technical Committee Draft. `http://uddi.org/pubs/uddi_v3.htm`. OASIS Consortium.

[25] Harlan Cleveland. Information As a Resource. *The Futurist*, 16:34–49, December 1982.

[26] Component. `http://dictionary.reference.com/browse/component`. Dictionary.com Unabridged (v 1.1), Random House, Inc.

[27] Software component. `http://en.wikipedia.org/wiki/Software_component`. Wikipedia.

[28] Aaron Crane. Does XML Suck? Or: Why XML is Technologically Terrible, but You Have to Use It Anyway. `http://xmlsucks.org/but_you_have_to_use_it_anyway/does-xml-suck.html`. GBdirect Ltd.

[29] Michael C. Daconta, Kevin T. Smith, and Leo J. Obrst. *The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management.* John Wiley & Sons, Inc., New York, NY, USA, 2003.

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

[31] Alan M. Davis and Edward H. Bersoff. Impacts of life cycle models on software configuration management. *Communications of the ACM*, 34(8):104–118, 1991.

[32] John 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.

[33] Joseph S. Dumas and Janice C. Redish. *A Practical Guide to Usability Testing.* Intellect Ltd., October 1999.

[34] Mark Endrei, Jenny Ang, Ali Arsanjani, Sook Chua, Philippe Comte, Pål Krogdahl, Min Luo, and Tony Newling. *Patterns: Service-Oriented Architecture and Web Services.* IBM Redbooks, July 2004.

[35] Thomas Erl. *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[36] Theodoros Evgeniou. Information Integration and Information Strategies for Adaptive Enterprises. *European Management Journal*, 20(5):486–494, 2002.

[37] Amalina Farhi Ahmad Fadzlah and Aziz Deraman. Measuring the Usability of Software Applications: Metrics for Behaviorness. pages 448–454, 2007.

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

[39] Joseph Feller, Brian Fitzgerald, Scott A. Hissam, and Karim R. Lakhani. *Perspectives on Free and Open Source Software.* The MIT Press, 2005.

[40] Fielding et al. Hypertext Transfer Protocol — HTTP/1.1. Network Working Group, 1999. Request For Comments (RFC) 2616.

[41] International Organization for Standardization. *ISO 8879:1986: Information processing — Text and office systems — Standard Generalized Markup Language (SGML).* International Organization for Standardization, Geneva, Switzerland, August 1986.

[42] International Organization for Standardization. ISO/IEC 12207:1995: Information technology — software life cycle processes. 1995. Geneva, Switzerland.

[43] International Organization for Standardization. ISO 92411-11:1998(E): Ergonomic requirements for office work with visual display terminals (VDTs) – Part 11: Guidance on usability. March 1998. Geneva, Switzerland.

[44] Google Inc. `http://www.google.com`.

[45] Martin Gudgin, Marc Hadley, et al. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). `http://www.w3.org/TR/soap12-part1/`. World Wide Web Consortium.

[46] Remo Häcki and Julian Lighton. The future of the networked company. *The McKinsey Quarterly*, (3):26–39, 2001.

[47] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques, 2nd edition — The Morgan Kaufmann series in data management systems.* Kaufman, San Francisco, California, 2006.

[48] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design Science in Information Systems Research. *MIS Quarterly*, 28(1):75–106, 2004.

[49] David M. Hilbert and David F. Redmiles. Extracting Usability Information from User Interface Events. *ACM Computing Surveys*, 32(4):384–421, 2000.

[50] Lorin Hitt and Prasanna Tambe. Broadband adoption and content consumption. *Information Economics and Policy*, 19:362–378, October 2007.

[51] Gregor Hohpe. Web Services: Pathway to a Service-Oriented Architecture? *ThoughtWorks*, 2002.

[52] Ying Huang, Santhosh Kumaran, and Jen-Yao Chung. A Service Management Framework for Service-Oriented Enterprises. In *Proceedings of the IEEE International Conference on E-Commerce Technology*, pages 181–186, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[53] Mamdouh H. Ibrhaim, Kerrie Holley, Nicolai M. Josuttis, Brenda Michelson, Dave Thomas, and John 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.

[54] IntelliCAD Technology Consortium. `http://www.intellicad.org/`.

[55] Ivar Jacobson. Object Oriented Development in an Industrial Environment. *SIGPLAN Not.*, 22(12):183–191, 1987.

[56] Slinger Jansen. *Customer Configuration Updating in a Software Supply Network*. PhD thesis, Universiteit Utrecht, October 2007.

[57] Slinger Jansen and Sjaak Brinkkemper. Definition and Validation of the Key Process Areas of Release, Delivery and Deployment of Product Software Vendors: turning the ugly duckling into a swan. In *proceedings of the International Conference on Software Maintenance (ICSM2006, Research track)*, September 2006.

[58] Slinger Jansen and Sjaak Brinkkemper. Pheme: A Communication Infrastructure for Product Software Knowledge. In *ICSM 2007: Proceedings of the IEEE International Conference on Software Maintenance 2007*, pages 527–528, October 2007.

[59] Slinger Jansen and Sjaak Brinkkemper. Rapport Nationaal Benchmarkonderzoek Productuitlevering 2007. 2007.

[60] Slinger Jansen, Sjaak Brinkkemper, and Tijs van der Storm. Living on the Cutting Edge: Automating Continuous Customer Configuration Updating. In *Proceedings of the Third International ERCIM Symposium on Software Evolution (Software Evolution 2007)*, 2007.

[61] Alan Jones. ISO 12207 Software life cycle processes - fit for purpose? *Software Quality Journal*, 5:243–253, 1996.

[62] Sravanthi Kalepu, Shonali Krishnaswamy, and Seng Wai 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.

[63] Anish Karmarkar and Mike Edwards. *Assembly of Business Systems Using Service Component Architecture*, volume 4294/2006, pages 529–539. Springer Berlin / Heidelberg, 2006.

[64] Alexander Keller and Heiko Ludwig. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *Journal of Network and Systems Management*, 11(1):57–81, 2003.

[65] Alexander Keller, Heiko Ludwig, Asit Dan, Richard King, and Richard Franck. A Service Level Agreement Language for Dynamic Electronic Services. 3:43–59.

[66] Dirk Krafzig, Karl Banke, and Dirk Slama. *Enterprise SOA: Service-Oriented Architecture Best Practices*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[67] Scott Laningham. IBM developerWorks Interviews: Tim Berners-Lee — Orginator of the Web and director of the W3C talks about where we've come, and about the challenges and opportunities ahead. `http://www.ibm.com/developerworks/podcast/dwi/cm-int082206txt.html`. IBM developerWorks podcast interview, August 2006.

[68] Kung-Kiu Lau and Mario Ornaghi. OOD Frameworks in Component-Based Software - Development in Computational Logic. In *LOPSTR '98: Proceedings of the 8th International Workshop on Logic Programming Synthesis and Transformation*, pages 101–123, London, UK, 1990. Springer-Verlag.

[69] Alexander Lazovik, Marco Aiello, and Mike Papazoglou. Planning and monitoring the execution of web service requests. *Int. J. Digit. Libr.*, 6(3):235–246, 2006.

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

[71] Xumin Liu and Athman Bouguettaya. Managing Top-down Changes in Service-Oriented Enterprises. In *IEEE International Conference on Web Services (ICWS 2007)*, pages 1072–1079, Salt Lake City, UT, USA, July 2007. IEEE Computer Society.

[72] C. Lovelock, S. Vandermerwe, and B. Lewis. *Services marketing.* Prentice Hall Europe, 1996.

[73] Anbazhagan Mani and Arun Nagarajan. Understanding quality of service for Web services. http://www.ibm.com/developerworks/java/library/ws-quality.html. IBM developerWorks, January 2002.

[74] Daniel A. Menascé, Daniel Barbará, and Ronald 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.

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

[76] Nikola Milanovic. *Contract-based Web Service Composition.* PhD thesis, Humboldt-Universität Berlin, June 2006.

[77] Ernest-Jan Mutsaers, Han van der Zee, and Henrik Giertz. The evolution of Information Technology. *Information Management & Computer Security*, 6:115–126, 1998.

[78] Felix Naumann, Ulf Leser, and Johann Christoph 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.

[79] Eric Newcomer and Greg Lomow. *Understanding SOA with Web Services (Independent Technology Guides).* Addison Wesley Professional, 2004.

[80] Jakob Nielsen. *Usability Engineering.* Academic Press, Boston, MA, USA, 1993.

[81] Frank Niessink, Viktor Clerc, Ton Tijdink, and Hans van Vliet. The IT Service Capability Maturity Model. Technical report, Department of Computer Science, Faculty of Sciences, Vrije Universiteit, Amsterdam, The Netherlands, January 2005.

[82] Richard L. Nolan. Managing the computer resource: a stage hypothesis. *Commun. ACM*, 16(7):399–405, 1973.

[83] Tim O'Reilly. The Open Source Paradigm Shift. http://tim.oreilly.com/articles/paradigmshift_0504.html. O'Reilly Media, June 2004.

[84] Odysseas Papapetrou and George 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.

[85] M. P. Papazoglou and D. Georgakopoulos. Introduction: Service-oriented computing. *Communications of the ACM*, 46(10):24–28, October 2003.

[86] Michael P. Papazoglou. Extending the Service-Oriented Architecture. *Business Integration Journal*, pages 18–21, February 2005.

[87] Michael P. Papazoglou and Willem-Jan van den Heuvel. Web Services Management: A Survey. *IEEE Internet Computing*, 9(6):58–64, 2005.

[88] Wolfgang Pree and Hermann Sikora. Design Patterns for Object-Oriented Software Development. In *ICSE '97: Proceedings of the 19th international conference on Software engineering*, pages 663–664, New York, NY, USA, 1997. ACM.

[89] Václav T. Rajlich and Keith H. Bennett. A Staged Model for the Software Life Cycle. *Computer*, 33(7):66–71, 2000.

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

[91] Lauge Baungaard Rasmussen and Arne Wangel. Work in the virtual enterprise—creating identities, building trust, and sharing knowledge. *AI & Society*, 21:184–199, January 2007.

[92] Susannah Ravden and Graham Johnson. *Evaluating usability of human-computer interfaces: a practical method*. Halsted Press, New York, NY, USA, 1989.

[93] Akhil Sahai, Jinsong Ouyang, Vijay Machiraju, and Klaus 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`. HP Laboratories, June 2001.

[94] Burak Sari, Tayyar Sen, and S. Engin Kilic. Formation of dynamic virtual enterprises and enterprise networks. *The International Journal of Advanced Manufacturing Technology*, 34:1246–1262, August 2006.

[95] Service Component Architecture Assembly Model Specification v1.00. `http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications`. Open Service Oriented Architecture collaboration, March 2007.

[96] Ahmed Seffah, Mohammad Donyaee, Rex B. Kline, and Harkirat K. Padda. Usability measurement and metrics: A consolidated model. *Software Quality Journal*, 14(2):159–178, June 2006.

[97] Brian Shackel and Simon J. Richardson. *Human factors for informatics usability*, chapter Usability — context, framework, definition, design and evaluation, pages 21–37. Cambridge University Press, New York, NY, USA, 1991.

[98] Stabiplan B.V. `http://www.stabiplan.nl`.

[99] Fareena Sultan. Consumer response to the Internet: an exploratory tracking study of on-line home users. *Journal of Business Research*, 55:655–663, August 2002.

[100] Clemens Szyperski. *Component software: beyond object-oriented programming*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1998.

[101] CMMI Product Team. CMMI for Development, version 1.2. Software Engineering Institute, August 2006. CMU/SEI-2006-TR-008.

[102] Shiu Lun Tsang, Siobhan Clarke, and Elisa Baniassad. An Evaluation of Aspect-Oriented Programming for Java-Based Real-Time Systems Development. *ISORC*, 00:291–300, 2004.

[103] V. Vaishnavi and W. Kuechler. Design Research in Information Systems. http://www.isworld.org/Researchdesign/drisISworld.htm. Association for Information Systems, January 2004, 2005.

[104] I. van de Weerd and S. Brinkkemper. Meta-modeling for situational analysis and design methods. To appear in the Handbook of Research on Modern Systems Analysis and Design Technologies and Applications, Idea Group Publishing, Hershey, PA, USA.

[105] Wil M.P. van der Aalst, Arthur H.M. ter Hofstede, and Mathias Weske. Business Process Management: A Survey. In *Business Process Management*, pages 1–12, 2003.

[106] Aad van Moorsel. Metrics for the Internet Age: Quality of Experience and Quality of Business. Technical report, E-Services Software Research Department, Hewlett-Packard Laboratories, Palo Alto, California, USA, 2001.

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

[108] Gregor von Bochmann, Brigitte Kerhervé, Hanan Lutfiyya, Mohamed-Vall M. Salem, and Haiwei 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.

[109] Guijun Wang, Alice Chen, Changzhou Wang, Casey Fung, and Stephen Uczekaj. Integrated Quality of Service (QoS) Management in Service-Oriented Enterprise Architectures. In *Proceedings of the 8th IEEE International Enterprise Distributed Object Computing Conference (EDOC)*, pages 21–32, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[110] John Ward and Joe Peppard. *Strategic Planning for Information Systems.* John Wiley & Sons, Inc., New York, NY, USA, 2002.

[111] What is Web services? - A Word Definition From the Webopedia Computer Dictionary. http://www.webopedia.com/TERM/W/Web_services.html. Webopedia Computer Dictionary.

[112] Christopher C. Whitehead. Evaluating web page and web site usability. In *ACM-SE 44: Proceedings of the 44th annual Southeast regional conference*, pages 788–789, New York, NY, USA, 2006. ACM.

[113] Thomas Woodley and Stephane Gagnon. BPM and SOA: Synergies and Challenges. In *Web Information Systems Engineering (WISE) 2005*, pages 679–688, 2005.

[114] Naiqi Wu and Ping Su. Selection of partners in virtual enterprise paradigm. *Robotics and Computer-Integrated Manufacturing*, 21:119–131, April 2005.

[115] Jian Yang and Mike P. Papazoglou. Service Components for Managing the Life-Cycle of Service Compositions. *Information Systems*, 29(2):97–125, 2004.

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

[117] Tao Yu and Kwei-Jay 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.

[118] Tao Yu, Yue Zhang, and Kwei-Jay Lin. Efficient algorithms for Web services selection with end-to-end QoS constraints. *ACM Transactions on the Web (TWEB)*, 1(1):6, 2007.

[119] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Quan 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.

[120] Ronggang Zhou. How to Quantify User Experience: Fuzzy Comprehensive Evaluation Model Based on Summative Usability Testing. pages 564–573, 2007.

Appendix A

Nuntia Database Diagram

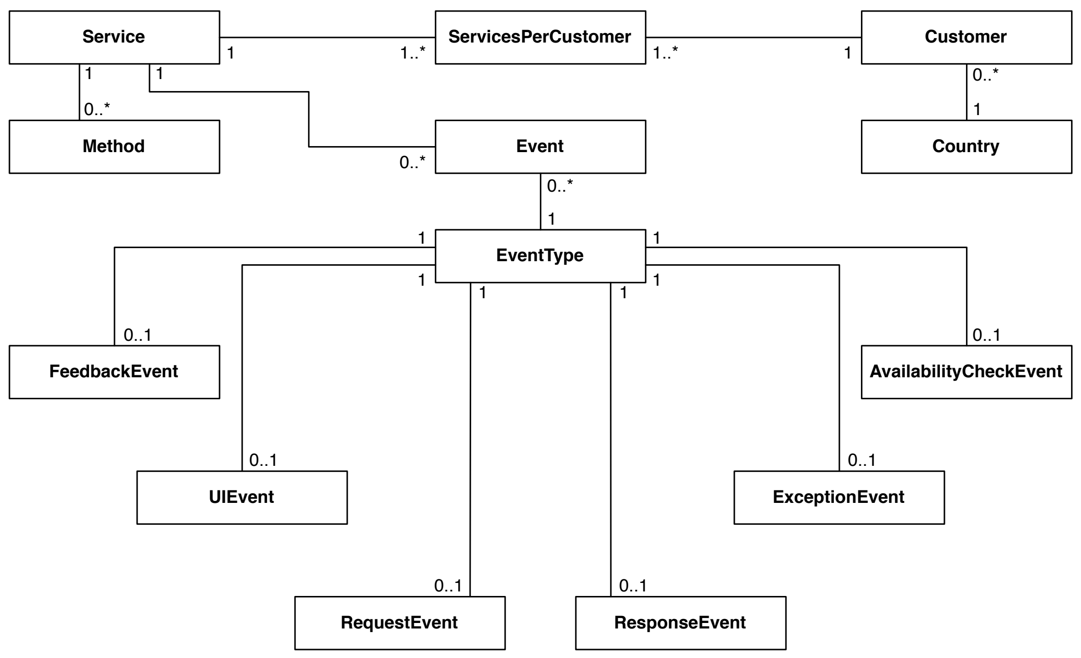


Figure A.1: *Database diagram of the Nuntia database, used to store logged usage and feedback data and generate SKU reports*

Appendix B

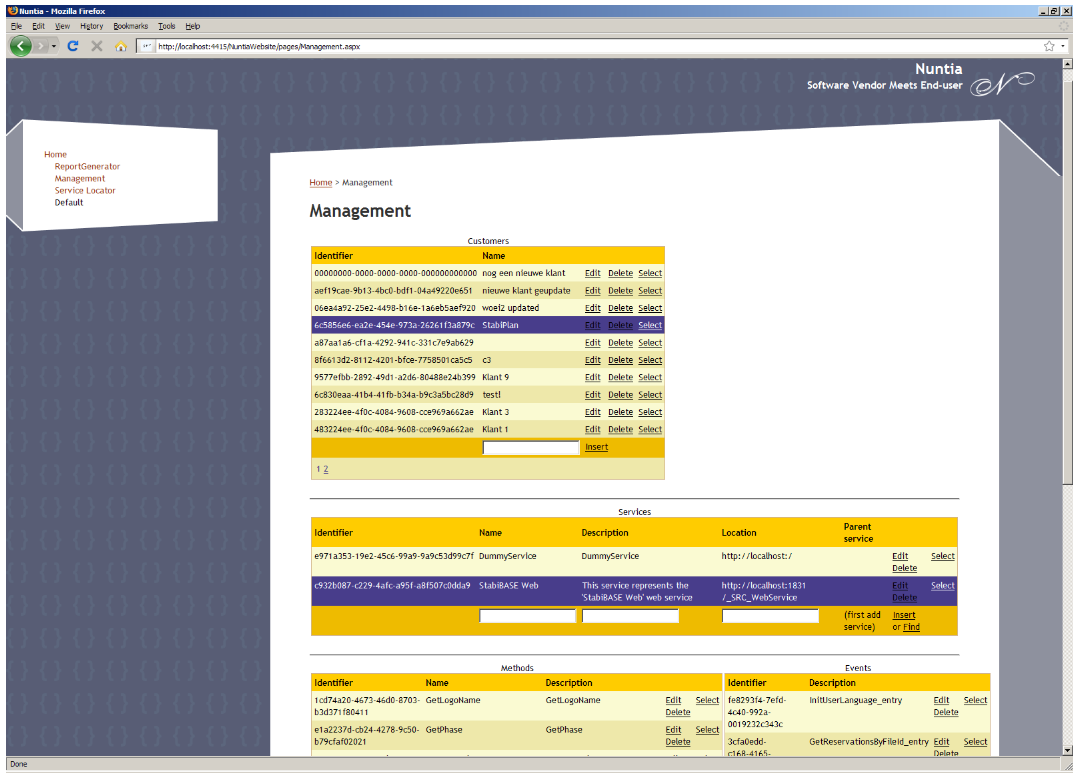
Environment Information Data Structure

```
<environmentinfo>
  <browser>
      <id><![CDATA[{0}]]></id>
      <support>
          <cookies>{1}</cookies>
          <java>{2}</java>
          <javascript>{3}</javascript>
          <vbscript>{4}</vbscript>
          <frames>{5}</frames>
          <css>{6}</css>
          <callbacks>{7}</callbacks>
      </support>
      <versions>
          <browser>{8}</browser>
          <javascript>{9}</javascript>
          <msdom>{10}</msdom>
          <w3cdom>{11}</w3cdom>
      </versions>
      <characteristics>
          <ismobile>{12}</ismobile>
          <platform>{13}</platform>
          <screendimensions>{14},{15}</screendimensions>
          <screenbitdepth>{16}</screenbitdepth>
      </characteristics>
  </browser>
  <other>
      <headers><![CDATA[{17}]]></headers>
      <url>{18}</url>
      <requesttype>{19}</requesttype>
      <secureconnection>{20}</secureconnection>
      <currentpage><![CDATA[{21}]]></currentpage>
      <currentpagesize>{22}</currentpagesize>
      <requesterip>{23}</requesterip>
  </other>
</environmentinfo>
```

Figure B.1: *Data structure of the environment information logged by Nuntia in case of an exception*

Appendix C

Nuntia Screenshots



Nuntia - Mozilla Firefox

http://localhost:4415/NuntiaWebsite/pages/Management.aspx

Nuntia
Software Vendor Meets End-user

Home
ReportGenerator
Management
Service Locator
Default

Home > Management

Management

Customers

Identifier	Name			
00000000-0000-0000-0000-000000000000	nog een nieuwe klant	Edit	Delete	Select
aef19cae-9b13-4bc0-bdf1-04a49220e651	nieuwe klant geupdate	Edit	Delete	Select
06ea4a92-25e2-4498-b16e-1a6eb5aef920	woei2 updated	Edit	Delete	Select
6c5856e6-ea2e-454e-973a-26261f3a879c	StabiPlan	Edit	Delete	Select
a87aa1a6-cf1a-4292-941c-331c7e9ab629		Edit	Delete	Select
8f6613d2-8112-4201-bfce-7758501ca5c5	c3	Edit	Delete	Select
9577efbb-2892-49d1-a2d6-80488e24b399	Klant 9	Edit	Delete	Select
6c830eaa-41b4-41fb-b34a-b9c3a5bc28d9	test!	Edit	Delete	Select
283224ee-4f0c-4084-9608-cce969a662ae	Klant 3	Edit	Delete	Select
483224ee-4f0c-4084-9608-cce969a662ae	Klant 1	Edit	Delete	Select
		Insert		
1 2				

Services

Identifier	Name	Description	Location	Parent service		
e971a353-19e2-45c6-99a9-9a9c53d99c7f	DummyService	DummyService	http://localhost:/		Edit Delete	Select
c932b087-c229-4afc-a95f-a8f507c0dda9	StabiBASE Web	This service represents the 'StabiBASE Web' web service	http://localhost:1831 /_SRC_WebService		Edit Delete	Select
				(first add service)	Insert or Find	

Methods

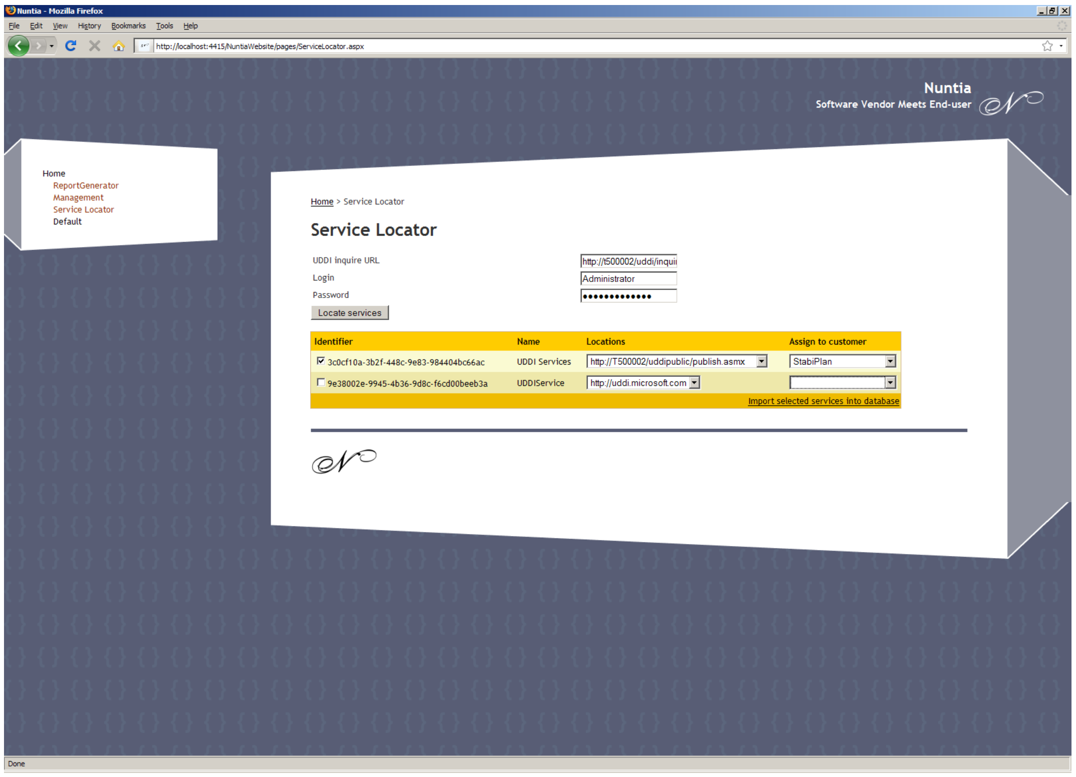
Identifier	Name	Description		
1cd74a20-4673-46d0-8703-b3d371f80411	GetLogoName	GetLogoName	Edit Delete	Select
e1a2237d-cb24-4278-9c50-b79cfaf02021	GetPhase	GetPhase	Edit Delete	Select

Events

Identifier	Description		
fe8293f4-7efd-4c40-992a-0019232c343c	InitUserLanguage_entry	Edit Delete	Select
3cfa0edd-c168-4165-	GetReservationsByFileId_entry	Edit Delete	Select

Done

Figure C.1: *Screenshot of Nuntia's management web interface*



Nuntia - Mozilla Firefox

http://localhost:4415/NuntiaWebsite/pages/ServiceLocator.aspx

Nuntia
Software Vendor Meets End-user

Home
ReportGenerator
Management
Service Locator
Default

Home > Service Locator

Service Locator

UDDI inquire URL	http://t500002/uddi/inqui
Login	Administrator
Password	●●●●●●●●●●●●●

Locate services

Identifier	Name	Locations	Assign to customer
✓ 3c0cf10a-3b2f-448c-9e83-984404bc66ac	UDDI Services	http://T500002/uddipublic/publish.asmx	StabiPlan
☐ 9e38002e-9945-4b36-9d8c-f6cd00beeb3a	UDDIService	http://uddi.microsoft.com	

Import selected services into database

Done

Figure C.2: *Screenshot of Nuntia's Service Locator*

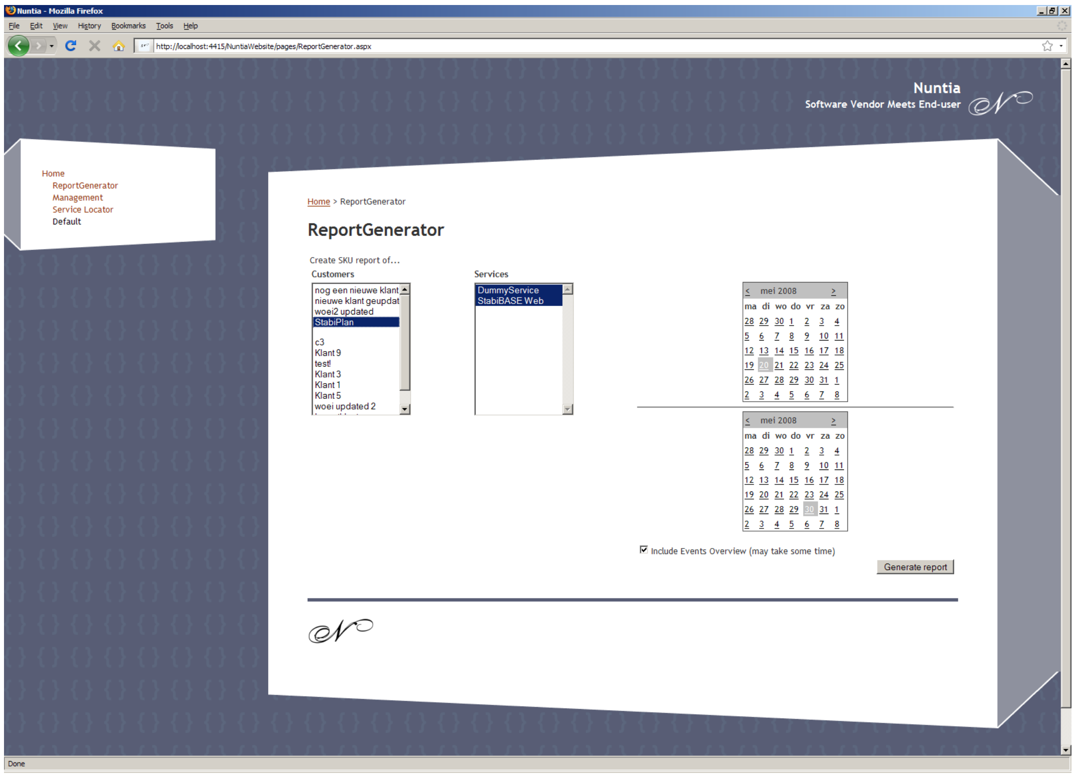
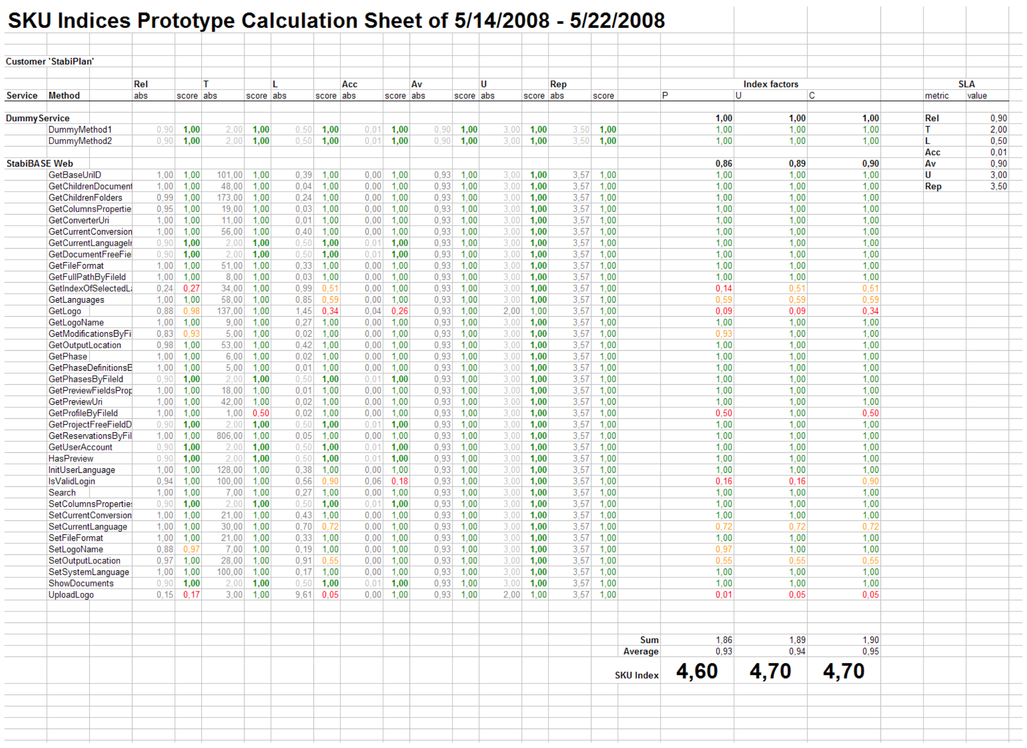


Figure C.3: *Screenshot of Nuntia's Report Generator*

Appendix D

Nuntia SKU Report



SKU Indices Prototype Calculation Sheet of 5/14/2008 - 5/22/2008

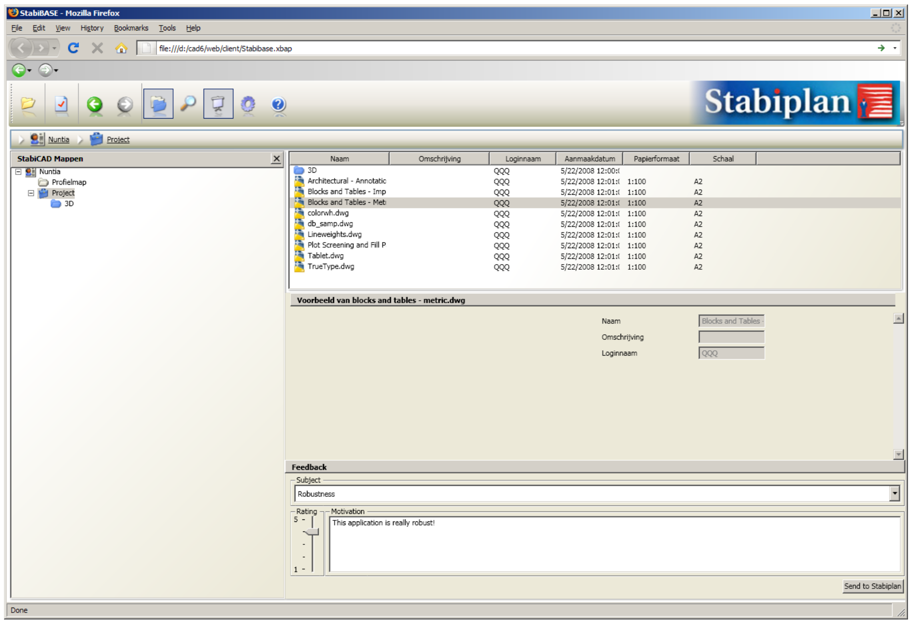
Customer 'StabiPlan'

		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
DummyService																	**1,00**	**1,00**	**1,00**	**Rel**	0,90
	DummyMethod1	0,90	**1,00**	2,00	**1,00**	0,50	**1,00**	0,01	**1,00**	0,90	**1,00**	3,00	**1,00**	3,50	**1,00**		1,00	1,00	1,00	**T**	2,00
	DummyMethod2	0,90	**1,00**	2,00	**1,00**	0,50	**1,00**	0,01	**1,00**	0,90	**1,00**	3,00	**1,00**	3,50	**1,00**		1,00	1,00	1,00	**L**	0,50
																				Acc	0,01
StabiBASE Web																	**0,86**	**0,89**	**0,90**	**Av**	0,90
	GetBaseUrlID	1,00	1,00	101,00	1,00	0,39	1,00	0,00	1,00	0,93	1,00	3,00	**1,00**	3,57	1,00		1,00	1,00	1,00	**U**	3,00
	GetChildrenDocument	1,00	1,00	48,00	1,00	0,04	1,00	0,00	1,00	0,93	1,00	3,00	**1,00**	3,57	1,00		1,00	1,00	1,00	**Rep**	3,50
	GetChildrenFolders	0,99	1,00	173,00	1,00	0,24	1,00	0,00	1,00	0,93	1,00	3,00	**1,00**	3,57	1,00		1,00	1,00	1,00		
	GetColumnsPropertie	0,95	1,00	19,00	1,00	0,03	1,00	0,00	1,00	0,93	1,00	3,00	**1,00**	3,57	1,00		1,00	1,00	1,00		
	GetConverterUri	1,00	1,00	11,00	1,00	0,01	1,00	0,00	1,00	0,93	1,00	3,00	**1,00**	3,57	1,00		1,00	1,00	1,00		
	GetCurrentConversion	1,00	1,00	56,00	1,00	0,40	1,00	0,00	1,00	0,93	1,00	3,00	**1,00**	3,57	1,00		1,00	1,00	1,00		
	GetCurrentLanguageI	0,90	**1,00**	2,00	**1,00**	0,50	**1,00**	0,01	**1,00**	0,93	1,00	3,00	**1,00**	3,57	1,00		1,00	1,00	1,00		
	GetDocumentFreeFie	0,90	**1,00**	2,00	**1,00**	0,50	**1,00**	0,01	**1,00**	0,93	1,00	3,00	**1,00**	3,57	1,00		1,00	1,00	1,00		
	GetFileFormat	1,00	1,00	51,00	1,00	0,33	1,00	0,00	1,00	0,93	1,00	3,00	**1,00**	3,57	1,00		1,00	1,00	1,00		
	GetFullPathByFileId	1,00	1,00	8,00	1,00	0,03	1,00	0,00	1,00	0,93	1,00	3,00	**1,00**	3,57	1,00		1,00	1,00	1,00		
	GetIndexOfSelectedL	0,24	0,27	34,00	1,00	0,99	0,51	0,00	1,00	0,93	1,00	3,00	**1,00**	3,57	1,00		0,14	0,51	0,51		
	GetLanguages	1,00	1,00	58,00	1,00	0,85	0,59	0,00	1,00	0,93	1,00	3,00	**1,00**	3,57	1,00		0,59	0,59	0,59		
	GetLogo	0,88	0,98	137,00	1,00	1,45	0,34	0,04	0,26	0,93	1,00	2,00	1,00	3,57	1,00		0,09	0,09	0,34		
	GetLogoName	1,00	1,00	9,00	1,00	0,27	1,00	0,00	1,00	0,93	1,00	3,00	**1,00**	3,57	1,00		1,00	1,00	1,00		
	GetModificationsByFi	0,83	0,93	5,00	1,00	0,02	1,00	0,00	1,00	0,93	1,00	3,00	**1,00**	3,57	1,00		0,93	1,00	1,00		
	GetOutputLocation	0,98	1,00	53,00	1,00	0,42	1,00	0,00	1,00	0,93	1,00	3,00	**1,00**	3,57	1,00		1,00	1,00	1,00		
	GetPhase	1,00	1,00	6,00	1,00	0,02	1,00	0,00	1,00	0,93	1,00	3,00	**1,00**	3,57	1,00		1,00	1,00	1,00		
	GetPhaseDefinitionsB	1,00	1,00	5,00	1,00	0,01	1,00	0,00	1,00	0,93	1,00	3,00	**1,00**	3,57	1,00		1,00	1,00	1,00		
	GetPhasesByFileId	0,90	**1,00**	2,00	**1,00**	0,50	**1,00**	0,01	**1,00**	0,93	1,00	3,00	**1,00**	3,57	1,00		1,00	1,00	1,00		
	GetPreviewFieldsProp	1,00	1,00	18,00	1,00	0,01	1,00	0,00	1,00	0,93	1,00	3,00	**1,00**	3,57	1,00		1,00	1,00	1,00		
	GetPreviewUri	1,00	1,00	42,00	1,00	0,02	1,00	0,00	1,00	0,93	1,00	3,00	**1,00**	3,57	1,00		1,00	1,00	1,00		
	GetProfileByFileId	1,00	1,00	1,00	0,50	0,02	1,00	0,00	1,00	0,93	1,00	3,00	**1,00**	3,57	1,00		0,50	1,00	0,50		
	GetProjectFreeFieldD	0,90	**1,00**	2,00	**1,00**	0,50	**1,00**	0,01	**1,00**	0,93	1,00	3,00	**1,00**	3,57	1,00		1,00	1,00	1,00		
	GetReservationsByFil	1,00	1,00	806,00	1,00	0,05	1,00	0,00	1,00	0,93	1,00	3,00	**1,00**	3,57	1,00		1,00	1,00	1,00		
	GetUserAccount	0,90	**1,00**	2,00	**1,00**	0,50	**1,00**	0,01	**1,00**	0,93	1,00	3,00	**1,00**	3,57	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**	3,57	1,00		1,00	1,00	1,00		
	InitUserLanguage	1,00	1,00	128,00	1,00	0,38	1,00	0,00	1,00	0,93	1,00	3,00	**1,00**	3,57	1,00		1,00	1,00	1,00		
	IsValidLogin	0,94	1,00	100,00	1,00	0,56	0,90	0,06	0,18	0,93	1,00	3,00	**1,00**	3,57	1,00		0,16	0,16	0,90		
	Search	1,00	1,00	7,00	1,00	0,27	1,00	0,00	1,00	0,93	1,00	3,00	**1,00**	3,57	1,00		1,00	1,00	1,00		
	SetColumnsPropertie	0,90	**1,00**	2,00	**1,00**	0,50	**1,00**	0,01	**1,00**	0,93	1,00	3,00	**1,00**	3,57	1,00		1,00	1,00	1,00		
	SetCurrentConversion	1,00	1,00	21,00	1,00	0,43	1,00	0,00	1,00	0,93	1,00	3,00	**1,00**	3,57	1,00		1,00	1,00	1,00		
	SetCurrentLanguage	1,00	1,00	30,00	1,00	0,70	0,72	0,00	1,00	0,93	1,00	3,00	**1,00**	3,57	1,00		0,72	0,72	0,72		
	SetFileFormat	1,00	1,00	21,00	1,00	0,33	1,00	0,00	1,00	0,93	1,00	3,00	**1,00**	3,57	1,00		1,00	1,00	1,00		
	SetLogoName	0,88	0,97	7,00	1,00	0,19	1,00	0,00	1,00	0,93	1,00	3,00	**1,00**	3,57	1,00		0,97	1,00	1,00		
	SetOutputLocation	0,97	1,00	28,00	1,00	0,91	0,55	0,00	1,00	0,93	1,00	3,00	**1,00**	3,57	1,00		0,55	0,55	0,55		
	SetSystemLanguage	1,00	1,00	100,00	1,00	0,17	1,00	0,00	1,00	0,93	1,00	3,00	**1,00**	3,57	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**	3,57	1,00		1,00	1,00	1,00		
	UploadLogo	0,15	0,17	3,00	1,00	9,61	0,05	0,00	1,00	0,93	1,00	2,00	1,00	3,57	1,00		0,01	0,05	0,05		
																Sum	1,86	1,89	1,90		
																Average	0,93	0,94	0,95		
																SKU Index	**4,60**	**4,70**	**4,70**		

Figure D.1: *Screenshot of a SKU report generated by Nuntia*

Appendix E

StabiBASE Web Integration


StabiBASE - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///d:/cad6/web/client/Stabibase.xbap
Stabiplan
Nuntia
Project
StabiCAD Mappen
Nuntia
Profielmap
Project
3D
Done

Naam	Omschrijving	Loginnaam	Aanmaakdatum	Papierformaat	Schaal
3D		QQQ	5/22/2008 12:00:(		
Architectural - Annotatic		QQQ	5/22/2008 12:01:(	1:100	A2
Blocks and Tables - Imp		QQQ	5/22/2008 12:01:(	1:100	A2
Blocks and Tables - Met		QQQ	5/22/2008 12:01:(	1:100	A2
colorwh.dwg		QQQ	5/22/2008 12:01:(	1:100	A2
db_samp.dwg		QQQ	5/22/2008 12:01:(	1:100	A2
Lineweights.dwg		QQQ	5/22/2008 12:01:(	1:100	A2
Plot Screening and Fill P		QQQ	5/22/2008 12:01:(	1:100	A2
Tablet.dwg		QQQ	5/22/2008 12:01:(	1:100	A2
TrueType.dwg		QQQ	5/22/2008 12:01:(	1:100	A2

Voorbeeld van blocks and tables - metric.dwg

Naam: Blocks and Tables -
Omschrijving:
Loginnaam: QQQ

Feedback

Subject: Robustness

Rating: 5 – 1

Motivation: This application is really robust!

Send to Stabiplan

Figure E.1: *Adjusted StabiBASE Web user interface, incorporating feedback submission functionality*

Appendix F

Thesis Research Output Publications

This chapter is considered confidential. Therefore, this chapter is not published. Please contact Henk van der Schuur at hwschuur@cs.uu.nl for more information.