Lappeenranta University of Technology

Faculty of Technology Management

Department of Information Technology, Laboratory of Software Engineering

Master's Thesis

Tommi Kähkönen

THE EFFECT OF SERVICE ORIENTED ARCHITECTURE AND CLOUD COMPUTING ON SOFTWARE TESTING

Examiners and supervisors: Professor Kari Smolander

Master of Science Leah Riungu

Abstract

Lappeenranta University of Technology

Faculty of Technology Management

Department of Information Technology, Laboratory of Software Engineering

Tommi Kähkönen

THE EFFECT OF SERVICE ORIENTED ARCHITECTURE AND CLOUD COMPUTING ON SOFTWARE TESTING

Master's Thesis

2011

114 pages, 16 pictures, 5 tables, 1 appendix

Examiners and supervisors:	Professor Kari Smolander
	Master of Science Leah Riungu

Keywords: service oriented architecture, web services, software testing, cloud computing, SOA testing

Software testing is an important and challenging phase in software development process that verifies the quality of the end product. Technologies including service oriented architecture (SOA) and cloud computing increase the complexity of software testing process. In this thesis, a literature review was made in order to find how the testing process would change because of these technologies. Some viewpoints from practice were gathered by utilizing the MASTO project interviews. The unique features of SOA make the testing more difficult and absolutely require the collaboration between stakeholders as well as the usage of testing tools. Cloud computing can offer a fully capable testing environment with pay per use model for testing organizations. Test strategy must compare different cloud providers and their suitability. Building of SaaS applications or cloud environments introduce certain technical challenges that must be addressed in design and testing phases. Cloud computing was not utilized at large scale during the time when the interviews were made. SaaS as a software delivery model was considered in many companies. Future research is needed to find out more about the exact changes introduced by new technologies on software testing in practice.

Tiivistelmä

Lappeenrannan Teknillinen Yliopisto

Teknistaloudellinen Tiedekunta

Tietotekniikan Osasto, Ohjelmistotekniikan Laboratorio

Tommi Kähkönen

PALVELUKESKEISEN ARKKITEHTUURIN JA ETÄRESURSSIPALVELUN VAIKUTUS OHJELMISTOTESTAUKSEEN

Diplomityö

2011

114 sivua, 16 kuvaa, 5 taulukkoa, 1 liite

Tarkastajat ja ohjaajat: Professori Kari Smolander

Diplomi-insinööri Leah Riungu

Hakusanat: palvelukeskeinen arkkitehtuuri, www-sovelluspalvelut, ohjelmistotestaus, etäresurssipalvelu, SOA testaus

Ohjelmistotestaus on tärkeä ja haastava ohjelmistokehitysprosessin vaihe, jolla voidaan varmentaa lopputuotteen laatu. Palvelukeskeinen arkkitehtuuri (SOA) ja etäresurssipalvelu (Cloud Computing) lisäävät testausprosessin mutkikkuutta. Tässä diplomityössä tehtiin kirjallisuuskatsaus, jotta saataisiin selville näiden teknologioiden vaikutus ohjelmistotestausprosessiin. MASTO projektin haastatteluaineistoa hyödynnettiin, jotta kysymyksiin saataisiin näkökulmia myös käytännöstä. Palvelukeskeisen arkkitehtuurin ominaispiirteet vaikeuttavat testausta merkittävästi, ja vaativat, että testausprosessin eri osakkaiden välillä tehdään riittävästi yhteistyötä, ja että testaustyökaluja käytetään testauksen automatisointia varten. Etäresurssipalvelu voi tarjoa testausorganisaatiolle kokonaisen testausympäristön, josta maksetaan käytön mukaan. Kun etäresurssipalveluja hyödynnetään, täytyy eri vaihtoehtojen sopivuus arvioida yrityksen testausstrategiassa. Ohjelmisto palveluna (SaaS) –sovellusten tai pilviympäristöjen rakentaminen tuo mukanaan teknisiä haasteita, jotka täytyy huomioida suunnittelu- ja testausvaiheissa. Haastatteluaineistosta ilmeni, että etäresurssipalvelua ei hyödynnetty suuressa mittakaavassa. SaaS–mallia sen sijaan on harkittu monissa yrityksissä. Lisää tutkimusta tarvitaan, jotta saataisiin tarkempi kuva niistä muutoksista, joita uudet teknologiat aiheuttavat testausprosessiin käytännössä.

Preface

I would like to thank Kari and Leah for their flexibility, support and for giving me full rein to do this thesis by allowing me to focus at those subjects I’m personally interested in.

I would also like to than Tero Pesonen for his support and for the interesting conversations we had concerning my thesis.

Lappeenranta 23.3.2011

-Tommi Kähkönen

“The aim of Computer Science is to build something that will last at least until we've finished building it.”

-Carl William Brown

“If we knew what we were doing, it wouldn't be research”

-Anonymous

Contents

Symbols and abbreviations

3D	Three-dimensional
AMI	Amazon Machine Image
API	Application Programming Interface
ASP	Application Service Provider
Avv	Avalanche virtual
BPEL	Business Process Execution Language
BPM	Business Process Management
CPU	Central Processing Unit
DB	Data Base
EAI	Enterprise Application Integration
EC2	Amazon Elastic Compute Cloud
ESB	Enterprise Service Bus
GA	Generic Algorithms
GUI	Graphical User Interface
HTTPS	Hypertext Transfer Protocol Secure
HuaaS	Humans as a service
IaaS	Infrastructure as a Service
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IETEF	Internet Engineering Task Force
IO	Input Output
IP	Internet Protocol
Ipsec	Internet Protocol Security
ISO	International Organization for Standardization
ISV	Independent Software Vendor
IT	Information Technology
J2EE	Java 2 Platform, Enterprise Edition
LAN	Local Area Network
MASTO	Adaptive Reference Model for Software Testing
NASA	National Aeronautics and Space Administration
OASIS	Organization for the Advancement of Structured Information Standards
OWL-S	Ontology Web Language for Web Services
PaaS	Platform as a Service
PHP	PHP: Hypertext Preprocessor
PKCS12	Public Key Cryptography Standards 12
POV-Ray	Persistence of Vision Ray tracer

QoS	Quality of Service
QoWS	Quality of Web Service
R&D	Research and Development
RDBMS	Relational Data Base Management System
RDS	Relational Database Service
REST	Representational state Transfer
RFC	Request for Comments
RIA	Rich Internet Application
ROI	Return of Investment
S3	Simple Strorage Service
SaaP	Software as a Product
SaaS	Software as a Service
SAN	Storage Area Network
SAWSDL	Semantic Annotation for Web Service Description Language
SLA	Service Level Agreement
SME	Small and Medium Enterprise
SOA	Service Oriented Architecture
SOAP	Simple Object access Protocol
SOC	Service Oriented Computing
SQL	Structured Query Language
SQS	Simple Queue Service
SSL	Secure Sockets LAyer
StaaS	Software Testing as a Service
STCv	Spirent TestCenter virtual
TBRC	Technology Business Research Center
UDDI	Universal Description, Discovery and Integration)
W3C	World Wide Web Consortium
WAN	Aide Area Network
WCF	Windows Communicarion Foundtation
VM	Virtual Machine
VPC	Virtual Private Cloud
VPN	Virtual Private Network
WS	Web Services Description
WSDL	Web Services Description Language
WS-I	Web Service Interoperability
WSLA	Web Service Level Agreement
WSMS	Web Service Management System
WSS	Web Service Security
XaaS	Everything as a Service
XML	eXtended Markup Language

1 Introduction

Software engineering is a rather young discipline with its own methodologies and standards. More and more complex problems are solved with software and boundaries between different systems are becoming fuzzier. Systems are being integrated, not only with the systems inside a company, but also with other companies' systems. New technologies and ways of thinking create challenges for software engineering processes that are constantly shaped by new requirements. Leveraging new technologies may introduce changes to existing processes, methods, techniques and ways of thinking. The volume of these changes may vary from minor adjustments to major changes that are organizational-wide.

Software testing is a phase in a software development cycle that not only affects heavily on the total costs of the software but also verifies that the quality of the software is expected. Testing has received more and more attention since it was separated from debugging decades ago and became a discipline of its own. (Kit 1995) Successful testing requires careful planning and sufficient amount of resources. Because of new technologies and increasing complexity, traditional aspects and ways may not hold true in software testing anymore. Software testing processes need to respond to the new challenges and need to be considered more seriously than ever before.

Recent ideas in doing business suggest organizations to focus at their core competence. However, these specialized companies need to collaborate with their external partners to deliver a complete product or service to the end user. This creates a business network of collaborating companies. Enterprise application integration (EAI) has remained as a challenge in enterprises during last two decades. Service-oriented architecture (SOA) is a way to integrate enterprise's applications in a reasonable way and also to integrate systems with the business partners' systems, and realize business networks for collaboration. By offering operations as business services to their partners, companies can facilitate on-demand collaboration opportunities. By having an enterprise architecture that is manageable and can react rapidly to the customers' needs can increase the competitive edge of a company.

Cloud computing is one of the most discussed topics in today's information technology industry. A short definition for cloud computing is: Computing capability delivered as a utility through internet standards and protocols (Kommalapati 2010). Computing capability can be software, hardware, platforms or even humans. Resources offered by cloud computing can be rented and utilized on demand, and disposed when no longer needed. Cloud computing offers new possibilities especially for small and medium-sized organizations that are now able to rent a great amount of computing power and virtual data storage instead of acquiring new hardware and hiring more persons to manage their own IT infrastructure. Cloud computing can offer a platform to host the applications or it can offer a fully operational testing environment. By renting services and resources from a cloud, a company can focus on its core competence while saving in costs.

One service model of cloud computing is Software as a service (SaaS). It was first introduced in 2001 (SIIA 2001). The first wave of application service provides (ASPs) soon vanished, because they failed to fulfill certain customer needs. SaaS tries to address the needs that were not met with ASP-model. Instead of printing software to medium and installing it on the customers' premises, software can be offered as an online service. This means that the software is hosted in a centralized place, possibly in the cloud, and customers access it via web browser. A customer does not have to care about new versions and updates – the vendor takes care of all the maintenance.

Building SOA solutions and leveraging cloud computing can introduce new challenges and opportunities for software development process. The aim of this master's thesis is to find out how service oriented architecture and cloud computing will change software testing process.

1.1 Background

This master's thesis is done as a part of the MASTO (Adaptive Reference Model for Software Testing) project that is part of TBRC (Technology Business Research Center) institute in Lappeenranta University of Technology. The MASTO studies affecting factors in software testing and relationships between those factors. During MASTO project representatives of 10 Finnish companies that produce software were interviewed. The size of these companies varies from small and medium sized to large organizations. The interview data is utilized in this thesis to gather some viewpoints from the IT industry. No formal research method is used in the analysis of the data. Instead, the data is viewed in the context of interesting topics that are related to the focus area of this thesis. The results from the fourth interview round are discussed in the 7th chapter.

The technologies discussed in this thesis are not brand new in that sense they been around for some time. (Ebert 2008) estimates that the foundations of SOA were established in 1999-2002, it was in limited use in 2003-2006 and its wider usage is in 2007 onwards. Numerous publications about SOA testing were made in 2008 and couple of years before. However, by looking at experts' blog writings and the number of scientific publications, the decreased amount of interest towards SOA can be identified after the year 2008. One possible reason for this would probably be the arrival of cloud computing, which became a hot topic in 2009 and captured the position as number one trend in information technology. Currently, as of writing this thesis in the first quarter of 2011, cloud computing seems to maintain its position as a major hype.

1.2 Purposes and Limitations

It is important to notice, that the technologies investigated in this thesis are slightly different compared to each other although they relate in certain areas. Service oriented architecture is a paradigm that contains a flexible set of design principles and patterns that can be used during the development and integration of systems. It is one way to carry out application integration in a company. The result of a successful SOA is an architecture that is easy to manage and can react to the changes quickly. In a long run, a successful SOA will save in costs. In a smaller scope, SOA principles and practices can be used to build business applications. SOA is an abstract paradigm that is not bound into any specific technology. Recent trends after all point out that Web services are the most popular technology in SOA realization (Erl 2005). This thesis focuses at software testing challenges caused by SOA with Web services.

Cloud computing is a comprehensive subject that can be viewed from different perspectives. Small and medium enterprises (SMEs) can utilize cloud resources, including software and platforms, in software testing. A large company that has invested significant amount of capital in datacenters may build own cloud environment and rent its resources to customers. Companies can also offer their software products in a SaaS model. All these perspectives are discussed in this thesis.

1.2.1 Difference between SOA and cloud computing

Cloud computing can be seen as an extension to SOA – as an alternative to get IT resources that are owned and hosted by third-parties. There is no way to “replace SOA with cloud computing” or “move from SOA to cloud”. SOA makes it possible to leverage IT-resources as services. Most cloud computing solutions have been defined and are going to be defined through SOA. For example, Amazon and force.com have emphasized a significant amount of service planning when designing their cloud services. (Linthicum 2010; Linthicum 2009; Oracle 2008) Yunus (2010) also states that enterprise cloud computing can be established only by first reaching federated SOA that is "A Systematic approach to large-scale, enterprise wide SOA that enables organizations to integrate semi-independent SOA initiatives". According to Yunus, an enterprise is not able to outsource its core infrastructure without having fulfilled certain SOA requirements first. (Yunus 2010a)

1.2.2 Difference between SOA and SaaS

It is notable that the meaning of the word *service* is different in software as a service when compared to service oriented architecture. These two concepts are often mixed up with each other. An application that is designed by utilizing the principles of service orientation and web services is a SOA application. In this case, the term *service* often refers to web services. In SaaS, the meaning of the word is different. By using a SaaS application, the client rents software as a service, meaning that the user does not have to take care of installing, configuring and maintaining the software. The company providing the application takes care of different tasks as a part of the *service*. SaaS is a model of business and SOA is an architectural strategy. Applying SOA is not necessary when delivering applications as services but it may help in construction and optimization of SaaS applications. When requirements for SaaS application increase (for example, when systems need to interact with other applications and data in different environments and platforms) the movement towards a service oriented solution may need to be considered. (Rittinghouse 2010; Oracle 2008; Kommalapati 2007; Melendy 2008)

1.2.3 SaaS as a service model of Cloud Computing

Software as a service is one service type of cloud computing. However, the concept of SaaS has been around for a decade – a long time before cloud computing. A SaaS application may or may not be hosted in a cloud platform. In general SaaS can be viewed more like a software delivery model or a business model.

1.2.4 Research questions

This thesis combines both literature review and case study. The following research questions are set:

- How does software testing processes change when realizing SOA with Web services?
- How does software testing processes change when utilizing cloud computing?

A fair amount of literature about Web-services and SOA testing can be found. Thus, the main focus of testing SOA with Web services relies on the literature. Testing related to cloud computing is a recent area of research that is hardly investigated as of the time which this thesis was written. Because of this, not many research papers addressing it are available. Both literature and the MASTO interview data are used to find out what kind changes and possibilities to software testing process cloud computing would bring.

1.3 Structure

The theory part of this thesis consists of 3 chapters. The 2nd chapter introduces software testing as a phase of software development process. Basic concepts of testing are introduced and problems and challenges of testing are also addressed briefly. The 3rd chapter presents service oriented architecture with web services. Basic ideas and concepts of service orientation are introduced. Core web service standards are discussed briefly but WS-* extensions are left out of this thesis. Quality of service and the business benefits of SOA are discussed in their own sections in chapter 3. Cloud computing is examined in chapter 4 which includes discussion of benefits and risks associated with cloud computing utilization as well as the architecture and technologies of the cloud environment. The cloud service models as well as some of the current cloud services are introduced. The findings how software testing is changed by SOA and cloud computing are represented in chapters 5 and 6. The 7th chapter discusses the observations from the MASTO interviews. The final chapter (8) is the summary of the whole thesis.

2 SOFTWARE TESTING

Software testing is a process that is performed during the development of a software product. Its history starts from 1957, when it was separated from debugging, became a discipline of its own and began to be regarded as "detecting bugs in the software". Back then, software size and degree of complexity were totally different compared to today's software systems. One reason for increased complexity is because the criticality of problems solved by software is increasing all the time. Software testing is to discover software errors and verify the quality of the software product. (Kit 1995) Bugs in software can cause minor annoyance when discovered by a software user or, they can lead to massive catastrophes and economical losses. (Robillard 2003) NASA's spacecraft was destroyed because it went off-course during launch. The reason for this was the bug in software. The Mars Climate Orbiter as well as the Mars Polar Lander was destroyed because of the software errors. (Hurwitz 2007) The need for a well-planned software testing process is obvious to prevent the catastrophes from happening by reducing the likelihood of software failure and to ensure that software satisfies the customer needs and its quality is expected. (Kit 1995; Ribarov et al. 2007)

2.1 What is software testing?

The purpose of testing is to find bugs in software. The quality of the software can be verified by testing but cannot be built by it. Testing can verify the presence of certain quality attributes such as functionality, reliability and performance in software. (Robillard 2003) The planning of testing, documenting a test plan and test cases, building a test environment, executing test cases, observing results and fixing the defects that were found in tests are tasks that are performed in a typical software testing process. (Haikala 2004) Software testing comprises of verification and validation. This means that software testing not limited only to executing software to find defects. Testing of documents and other non-executable forms of a software product is also an essential part of software testing. IEEE/ANSI defines verification and validation as following: *“Verification is the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed of that phase. Validation is the process of evaluating a system or component during or at the end of development process to determine whether it satisfies specified requirements.”* (Kit 1995)

Verification includes evaluating, reviewing, inspecting and doing desk checks of work products. These work products can be requirement specifications, design specifications or pieces of code. Verification is sometimes called “human testing” because it involves looking at documents written on paper and is performed by humans. Validation usually involves the execution of actual software or simulation on a computer. Testing has been historically very validation-oriented and in some companies, it still is. Both verification and validation are needed in order to achieve comprehensive, effective and systematic error detection process. (Kit 1995)

2.2 Testing process and development process

In a typical software development process, no matter what specific development model is used, there are software life cycle phases like requirements specification, analysis, design, implementation, testing, as well as operation and maintenance. Each of these phases produces some kind of product that can be verified and validated. Testing process must be aligned with development process in a way that each development phase has a corresponding testing activity. For example, for the development phase “requirements” there is testing activity “requirements verification”. (Kit 1995) Early adoption of testing reduces the cost of making quality software and aims to find errors as early as possible. If an error is made during the requirements phase of the software development project, it will cost tens of times more to fix it after software has been released. (Kit 1995; Robillard 2003) The table below shows the average costs of fixing defects based on when they are introduced and detected.

	Time detected				
Time introduced	*Requirements*	*Architecture*	*Construction*	*System Test*	*Post-Release*
Requirements	1	3	5-10	10	10-100
Architecture	-		10	15	25-100
Construction	-	-	1	10	10-25

Table 1: Cost of software errors in different development phases (McConnell 2004)

The importance of software testing has been noticed during last decades and testing has even become a choice of career. Still, According to Kit, many companies have immature development and testing processes where testing is seen as an end-of-cycle task or as an “additional work that is done if there is time left”. Instead, testing should exist as a process that is active during the whole development cycle of the software. Kit has defined that a mature testing process follows six essentials of software testing:

1. The quality of the test process determines the success of test effort
2. Prevent defect migration by using early life-cycle testing techniques
3. The time of software testing tools is now
4. A real person must take responsibility for improving the test process
5. Testing is a professional discipline requiring trained, skilled people
6. Cultivate a positive team attitude of creative destruction

The first essential suggests that testing should be seen as an independent process with its own life cycle. It is not an activity that is performed just before releasing the product but instead, in each phase of the development process there should be a testing activity to be performed. The test process should be carried out by a separate testing team if possible and the process should be constantly improved. The second essential suggests using early lifecycle testing techniques because of the known fact that half of the errors in software are usually introduced in the requirements phase. The earlier the errors are discovered, the inexpensive it is to fix them. The errors tend to migrate to the later phases. Because of this, it is a fairly good idea to eradicate them at the early phases with reviews and inspections. Testing should be performed with the help of contemporary testing tools as the third essential promotes. Strategies for testing tool acquisition and selection must exist in a company. The fourth essential emphasizes the management - there should be a dedicated person who is responsible for testing. In order to improve the testing process, someone must plan and manage it. The fifth essential says that testing requires trained and skilled people to perform testing activities. The sixth essential is about tester's attitude that should be "creative destruction". The goal of testing should be to find bugs instead of proving the correctness of software. (Kit 1995)

2.3 Black-box and white-box testing strategies

Test strategies can be divided into black-box and white-box testing. Black-box testing is also known as data-driven or input/output-driven testing where tester sees the program as a black box without knowing the internal structure and behavior of the program. The test data for black-box testing is taken from software specification. In white-box testing (sometimes called logic-driven testing), the test data is derived from an examination of the program's internal logic. Combining the elements from both black-box and white-box must be used in order to perform successful testing. (Myers 2004)

2.4 Static testing (verification)

Static testing (sometimes called human testing) includes methods such as desk checking, reviews, walkthroughs and inspections. Static testing is a least expensive model of testing and has the largest potential to reduce defects in software under development. It is important to prevent defects migrating from one development phase to another. The target of static testing is any work product, for example a document or a piece of code related to the software project. This work product is read and examined by a group of people. (Kit 1995) Some candidates for static testing are documents like software requirements and software project plans. Software developers' documents such as use cases, data flow diagrams, database designs, interfaces and security specifications as well as the code itself can also be tested with static testing methods. Testers' and end users' documentation can be tested. (Everett 2007)

Static testing is sometimes overlooked by saying that the time spent on it should instead be used for developing activities. The fact is that a few well spent hours on static testing of the software product documentation can save hundreds of hours correcting and redesigning the code caused by flaws in documentation. (Everett 2007) Static testing can find some types of errors more efficiently than dynamic (computer-based) testing. Static and dynamic testing are complementary, the overall error detection is not as efficient as it could be if the other one is missing. (Myers 2004)

2.5 Dynamic testing (validation)

Dynamic testing includes validation activities that can be divided into low and high level testing. Low level consists of debugging, unit testing and integration testing. High level testing consists of usability testing, functional testing, system testing and acceptance testing. Alpha and beta testing are often mentioned as their own levels of testing. (Kit 1995)

2.5.1 Low-level testing

Debugging means the tracking and fixing of errors from program's source code. (Haikala 2004) Some sources regard debugging as a part of testing, others see it as a development task. It is closely related to testing in a sense that if testing finds a bug, some amount of debugging might need to be done. Debugging is usually performed by the developer. In debugging, the source code is compiled in order to find implementation defects that can be caused by incorrect implementation of the control flow, misunderstandings about programming language, incorrect naming or spelling of the program statements. (Robillard 2003)

The target of **unit testing** (or module testing) is a piece of code, one product unit that is the smallest testable element of the software. This element can be a class, subprogram, function, routine or procedure. Testing one unit in isolation may require a generation of test beds. A test bed consists of stub and driver modules. A driver module makes it possible to call the services of the module under test and to observe the results. A driver module transmits test cases in the form of input arguments to the unit under test and either prints or interprets the results produced by the unit under test. Stub modules simulate the function of those modules that are needed by the module under test but are not implemented yet. Debugging and unit testing need no prior planning at the project level but all the other levels of testing need to be planned beforehand. (Kit 1995; Robillard 2003; McConnell 2004; Haikala 2004)

Integration testing is performed to ensure that elements, that can be modules, classes, packages, components or subsystems, work properly when combined. The main purpose of integration testing is to find errors in the interfaces between elements. Integration testing can be made by non-incremental "big bang" integration where all the components of the program are combined and the integrated result is tested. Bottom-up (testing the lower level modules first and going up) and top-down (the opposite direction) integration are more reasonable ways to perform integration testing. Low-level testing requires knowledge of program's internal structure. Because of this, low-level testing is usually performed by the developer himself. (Kit 1995; McConnell 2004)

2.5.2 High-level testing

High-level testing includes function, system, usability and acceptance testing and it is often performed by an independent testing group. **Functional testing** takes place after unit and integration testing have been completed. Its purpose is to find differences between the functional specification of software and actual behavior of the software. When a discrepancy is found, either the specification or program can be incorrect. All black-box testing methods can be used for function testing. **System testing** targets the whole system and the test results are compared with software requirements. System testing also targets the non-functional properties of the software with appropriate tests that can include performance testing, usability testing and security testing. The purpose of **usability testing** is to adapt the software so that it is suitable for the user rather than forcing users to change their working styles for the software. Usability testing collects information from the users that are using the software. Usability characteristics that can be tested are accessibility, responsiveness, efficiency and comprehensibility. **Acceptance testing** is made after the system is integrated but before deploying software to the customer. The purpose of this test level is to verify that the software is ready and to be used by end users, and they are able to perform the functions and tasks for which the software was built. **Alpha and beta testing** are specific types of acceptance testing. Alpha testing is usually done in the development environment to verify that the system implements the requirements that were set for it. Beta testing is done in customer's environment. Customers often participate beta testing and report problems to the developers. (Kit 1995; Robillard 2003; Haikala 2004) Moving from lower levels to higher levels, the nature of testing tends to change from white box view to more black box view of testing. (Haikala 2004)

2.5.3 Regression testing

Regression testing is an important activity in the testing process. After changes have been made to the software, it is necessary to check that changes have not caused any more defects. Same tests are repeated over and over again in regression testing and because of this, using automation is necessary due the limited time available for testing. Automation also minimizes the cost of building and maintaining the sets of regression tests. After a test case has been made, consideration has to be made whether to include that test case in regression testing. (Kit 1995; Robillard 2003)

2.6 Non-functional testing

Both system and acceptance testing include testing the non-functional properties of the software. The behavior of the system with respect to certain observable attributes is evaluated by non-functional testing. (Palacios et al. 2010) Non-functional testing includes performance (or load testing), usability, security, performance, recovery, installability and trustability testing. When planning testing, the needed types of non-functional tests need to be considered. Decisions about the types of non-functional testing to be done are determined by the domain and nature of the software. (Myers 2004)

Traditionally, building a testing environment that is capable to reliably simulate the traffic has been a challenging task. Performance testing (or load testing) is an important phase of development process and crucial part of testing to ensure that software is ready for production, reducing the risks of unwanted surprises. In performance testing, the expected usage of the system is monitored by simulating multiple users that are accessing the system simultaneously. To perform this kind of test, sufficient performance testing environment with testing software and hardware is needed. In order to get a realistic simulation, sufficient amount of bandwidth is required. Professional testers must be in place to run the tests, monitor them and analyze the results. Setting up this kind of environment can be very expensive. Different compromises in performance testing can be made but with detriment of the reliability of the results. (Girmonsky 2009)

2.7 Challenges of testing

Testing everything is not reasonable and generally not even possible. In a software development project, a huge amount of tests has to be performed within very limited time and resources. Very often, prioritizing must be done by doing a risk analysis to recognize the most important parts of the software that need to be tested. This creates a need for planning, scheduling, documenting and tracking the progress of testing. At some point, testing has to be stopped. Various statistical analyses can be performed on the test results to determine as accurately as possible the risk associated with ending test activities. (Robillard 2003) One difficulty of testing is that it should be viewed as a destructive process where work done by someone else is tried to be destroyed in order to efficiently find errors. (Myers 2004) Testing activities should be kept objective and impersonal, not as an attack against specific persons. (Everett 2007)

Systems grow out of company boundaries and integrate with other companies' systems. Among the general challenges of software testing, the new emerging technologies such as massively parallel processing, cloud computing, self- tuning and healing systems, cyber security and self-similarity in networks set completely new requirements for software testing and requires testing processes to change and adapt with them. (Collard 2009) E-commerce, service-oriented solutions and software as a service cause new challenges for testing which means that processes, quality thinking and attitudes need to be viewed with different attitude than before. (AppLabs 2008)

3 SERVICE ORIENTED ARCHITECTURE (SOA)

Service oriented computing (SOC) is a new generation platform for distributed computing. It includes many concepts, like its own design paradigm (service orientation) that provide rules and guidelines to realize certain design characteristics in implementation. Service oriented computing has its own design principles, design patterns, an own architectural model that is called service oriented architecture, along with many concepts, technologies and frameworks associated with it. The terms service oriented computing and service oriented architecture are not synonyms, but literature very often if not always treats them as one. (Erl 2008)

Service oriented computing is heavily affected by different paradigms and technologies such as object-orientation, distributed computing, business process management, web services, distributed computing and enterprise application integration (EAI). Service oriented computing is guided by numerous industry standards. Both standardization organizations (including W3C, OASIS and WS-I) and industry organizations (such as Microsoft, IBM, Sun Microsystems, Oracle, Hewlett-Packard, Fujitsu-Siemens and SAP) have actively developed service oriented computing and are constantly developing it further with new standards and specifications. (Erl 2008; Josuttis 2007)

3.1 Definitions of SOA

Service oriented architecture is an architectural model that aims to increase efficiency, agility and productivity of an enterprise by emphasizing services as the primary means to represent the solution logic. The purpose of SOA is to realize the goals of service oriented computing. These goals are discussed later in this chapter. SOA can be seen as a technology architecture that consists of technologies, products, APIs, supporting infrastructure extension among other parts. (Erl 2008) SOA can also be viewed as an architectural style for building software applications that use services in a network. (Haines & Rothenberger 2010) Numerous definitions for SOA exist. (Josuttis 2007) Linithicum's definition of SOA is: *"An SOA is a strategic framework of technology that allows all interested systems, inside and outside an organization, to expose and access well-defined services, and information bound to those services, that may be further abstracted to process layers and composite applications for solution development. In essence, SOA adds the agility aspect to architecture, allowing us to deal with system changes using a configuration layer rather than constantly having to redevelop these systems."* (Linthicum 2010) Oasis (2006) defines SOA as following: *"A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations."* (OASIS 2006) Another definition of SOA is following: *"SOA is an architectural paradigm for dealing with business processes distributed over a large landscape of existing and new heterogenous systems that are under control of different owners."* (Josuttis 2007)

SOA is mainly for building business applications. (Hurwitz 2007) Software processes in SOA are closely modeling business processes and business processes drive the design of SOA. The application logic of a SOA solution is loosely coupled and it is residing on multiple computers that can be inside or outside the organization. Replacing and redesign of components and reusing existing application logic in a SOA solution should be rather an effortless task. In SOA, components communicate through a single interface and systems typically serve multiple remote users. (Barber 2006) According to Josuttis, in order to establish SOA successfully, three components are needed: infrastructure, architecture and processes. A technical platform, such as web services are needed to form the infrastructure. Based on SOA concepts, standards and tools, certain decisions have to be made to form the architecture of the SOA system. Different processes include business process modeling, service lifecycles and SOA governance that is, one of the most important tasks in a successful SOA. (Josuttis 2007)

3.2 Service – a fundamental building block

The fundamental ideology behind SOA is that a large problem that needs to be solved with software is partitioned into smaller pieces - into individual units of logic known as services which are designed with principles of service oriented computing. A service is an independent entity that encapsulates a logic related to a certain business task or other logical grouping and exchanges information with other services in order to achieve the desired goal. A service can be combined with other services in order to create service compositions. Services can be distributed, meaning that they may reside inside or outside the company. Services exist autonomously but are not isolated. Services maintain a degree of commonality and standardization. (Erl 2005; Erl 2008) When utilized, services are not purchased as a software package. (Haines & Rothenberger 2010)

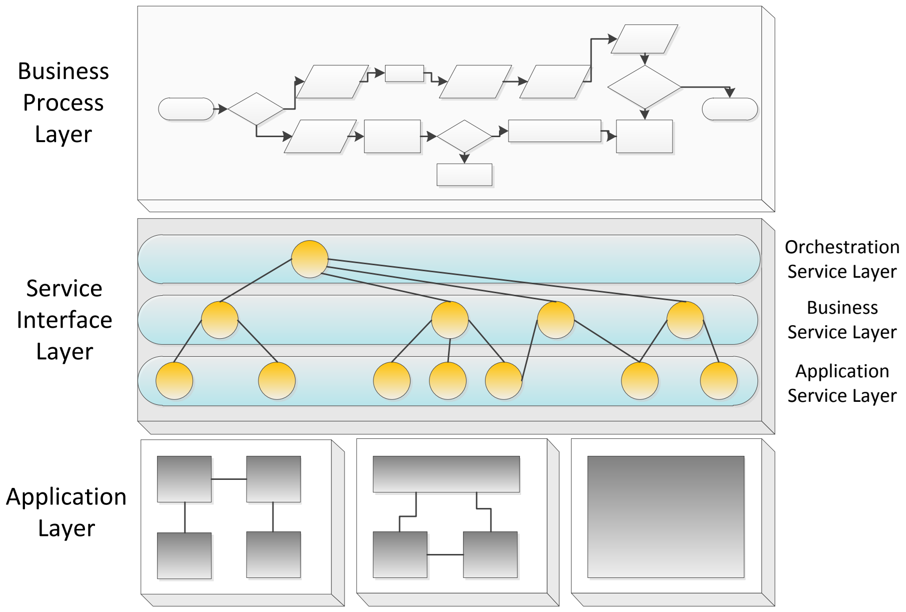
Inter-service communication requires that services are aware of each other. This happens with service descriptions that contains the name and location of the service and inputs and outputs – the data exchange requirements of the service. Services are loosely coupled through service descriptions. A communication framework is needed to establish communication between services. This framework defines the structure of messages, policies and protocols that are used in communication. Like services, messages are also autonomous. They also contain enough intelligence to govern their parts of processing logic. After sending a message, the sender is not able to determine, what will happen to message. (Erl 2005; Erl 2008)

3.3 Design principles

Several design principles can be followed in the design of a service-oriented solution. When designing a service oriented solution and building SOA, some important questions are: how should services be designed, how should relationship between services be defined, how should service descriptions be designed and how messages should be designed. Eight design principles of SOA have been defined. Not all design principles can be realized simultaneously, because principles inter-relate. This means that achieving one principle may require the realization of another principle. For example, in order to reach loose coupling, service contract must be present. Five principles establish concrete service design characteristics and the remaining three act more as regulatory influences. SOA design principles are represented in Appendix A. (Erl 2005; Erl 2008)

3.4 Abstraction of service layers

In order to achieve reusability, it is not possible to build agile services that accommodate both business and applications logic considerations simultaneously. Thus, specialized layers of services need to be built. These layers are the application service layer, the business service layer and the orchestration service layer. Picture 1 shows these layers between business process and application layer. (Erl 2005)



Picture 1: Service layers in SOA (Erl 2005)

Application service layer is to provide reusable functions that are related to processing data in an application environment. Application services are highly technology-specific. Most common types of application services are utility and wrapper services. Utility services are generic services that provide reusable operations for business services to complete business-centric tasks. Wrapper services are typically utilized for integration purposes. They encapsulate the logic of legacy systems. Services containing both application and business logic are called hybrid services. (Erl 2005)

The purpose of business process layer is to introduce services that concentrate on representing the business logic. Business services are always implementations of the business service model. However, business service can also be classified as a controller service or a utility service. It is very likely that business services act as controllers that are composed of several application services to execute their business logic. Business service models include task-centric and entity-centric business models. Task-centric business service encapsulates a business logic that is specific to a certain task or business process. Entity-centric business service encapsulates a specific business entity (that is for example, a timesheet or an invoice) thus having a greater potential of reuse. (Erl 2005)

The orchestration layer consists of one or more process services that compose the services from lower levels (business and applications services) according to the business rules and business logic that is embedded within process definitions. The orchestration layer prevents the need for other services to manage the details related to interaction that are required to ensure the correct execution of service operations. Process services reside in orchestration layer and compose other services that provide specific sets of functions, independent of the business rules and scenario specific logic. Process services can be classified as control services because they compose other services to execute business process logic. Process services can also become utility services if the process they execute can be considered reusable. Business rules and service execution sequence logic is abstracted from other services by orchestration. (Erl 2005)

3.5 Web Services

The evolution of web services can be seen as a driving force in the evolution of service orientation and it has considerably shaped service oriented computing. Even though SOA as a technology architecture and service orientation as a paradigm both are implementation-neutral, web-services are very often associated with them. (Erl 2005; Josuttis 2007) In order to build a SOA-solution, an implementation platform is needed. Web services technology offers this kind of platform. (Erl 2005) Web services are used to implement shared business tasks between enterprises and they are the enabling technology to connect different decoupled systems across various platforms, programming languages and applications. They can also be used for enterprise application integration. (Papazoglou 2008) The definition of Web service is following: "*A Web service is a platform independent, loosely coupled, self-contained, programmable web-enabled application, that can be described, published, discovered, coordinated and configured using XML artifacts (open standards) for the purpose of developing distributed interoperable applications.*" (Papazoglou 2008)

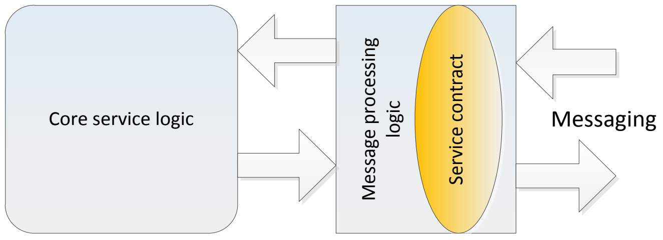
3.5.1 Roles of web services

Web services differ from each other based on the tasks they perform and the scope of the task. A web service can be a self-contained business task (for example, a funds withdrawal of deposit service), a complete business process (such as automated purchasing of office supplies), an application (demand forecast or stock replenishment application) or it can be a resource that is enabled by a web service (for example, an access to a database containing medical records). The functions of web services can vary from simple requests to complete business applications that access and combine information from many different sources. Web services are sometimes mixed with web applications. Four key differences between web services and web applications can be made: (Papazoglou 2008)

1. Web services can act as resources for other applications so that web services can use other services without human intervention.
2. A Web service knows its functional and non-functional properties and it can tell them to users and other web services. Functional properties include functions that the web service can perform; what inputs are required to produce its outputs. Non-functional properties include service utilization cost, security measures, performance characteristic and contact information
3. The state of the web service can be monitored by using external application management and control systems. This way web services are more visible and manageable than web-based applications.
4. Web services can also be brokered or auctioned. A broker can choose a suitable web service for its purposes among several services performing the same task. A choice can be based on, for example, the cost, speed or degree of security of the web service. (Papazoglou 2008)

3.5.2 Interface and implementation

The interface and implementation parts of the Web services are separated. The service interface is visible to the service users. It defines the functionality of the service and how to access it. The interface is also called as a service contract. It consists of WSDL (Web Service Description Language) definition and XML (eXtended Markup Language) Schema definition and can include WS (Web Service)-Policy definition. The implementation details of the service are hidden from the service users. Service implementation is divided into programming logic and message processing logic. Programming logic can be legacy logic that is wrapped by a web service or it can be logic that is specially developed for the web service. In this case, the logic is often called core service logic or core business logic. Parsers, processors and service agents compose the message processing logic of a web service. They can handle the messages sent or received by web services. Some of this logic is provided by the runtime environment but it can also be custom built. Picture 4 represents the components of web service. (Papazoglou 2008; Erl 2008)



Picture 2: Components of a web service (Erl 2008)

The same service interface can be implemented by different service providers by using any programming language. Implementations can be different: one implementation can provide certain functionality while another implementation might use a combination of other services to provide the same functionality. The separation between service interface and implementation can be so radical that organizations, which provide service interfaces are not necessary the same organizations that implement the services. (Papazoglou 2008) Different SOA vendors have developed their platforms to utilize web-services technology. Current industrial technologies and platforms include powerful XML and web services support. These technologies include IBM's Websphere Toolkit, Sun's Open Net Environment and JINITM technology, Microsoft's .NET and Novell's One Net initiatives, HP's e-speak and BEA's WebLogic Integration. (Hull & Su 2005)

3.5.3 Web service standards

The ultimate goal of Web service technology is to enable co-operation between applications over standardized protocols with no need of human intervention. The Web services technology stack consist of numerous industry standards defined by vendor community and it is evolving constantly. Standardization solves many interoperability issues that have been a problem in previous distributed technologies. Moreover, standardization of SOA allows vendors to create tools that automate certain steps in the software development process. (Haines & Rothenberger 2010) Network layer and XML messaging provide the foundations for interaction between Web services. The data that services pass to each other is in XML with data types specified by XML schema. (Hull & Su 2005) At the transport layer, web services commonly utilize the advantages of HTTP (Hypertext Transfer Protocol). (Papazoglou 2008) Web service standards can be classified to eight main groups:

1. Enabling technology standards
2. Core services standards
3. Context and transaction management standards
4. Service composition and collaboration standards
5. Reliability, routing, and attachments
6. Policies and metadata
7. WS-I profiles
8. Security

Core services standards include Web Service Description Language (WSDL), Simple Object Access Protocol (SOAP) and Universal Description, Discovery and Integration (UDDI). In a typical SOA scenario, WSDL descriptions are published in a service registry by service providers. Clients or service consumers then access this registry and dynamically combine services that fulfill the certain needs. SOAP establishes a communication channel between web services. (Palacios et al. 2010; Erl 2005)

Services need information about each other in order to communicate. WSDL offers a standard way to represent service interfaces (service contracts). WSDL defines the XML grammar for describing web services as communicating endpoints that can exchange messages with other services. (Papazoglou 2008) The service description includes the location of a service, how the service is used and what operations and methods the service offers. WSDL provides the metadata for service consumers so that they are able to send valid messages to service providers. WSDL also defines how the service reacts to the message it receives. Both abstract and concrete definitions form a WSDL document. The abstract definition describes the properties of the interface without a reference to the underlying technology. In order to execute its logic, abstract definition must be connected to a concrete definition that includes the transport protocol that is used as well as the physical address of the service. WSDL is extensible and allows description of service endpoints and their messages regardless of the used message formats or network protocols. (SOA Systems Inc. 2009a; Erl 2005; W3C 2001)

Web services communicate by using SOAP protocol that implements a request-response model for communication. SOAP is an XML-based lightweight protocol that is independent of any programming model and other implementation specific issues. SOAP enables the applications running in different operating systems and implemented with different technology platforms and programming languages, to communicate with each other. (SOA Systems Inc. 2009b; W3C 2007) SOAP has two fundamental properties. It can send and receive HTTP (or other) transport protocol packets and process XML messages. One major goal of SOAP is to be extensible with features including reliability, security, routing and message exchange patterns. WS-Extensions can be used to add more features into SOAP messages. (Papazoglou 2008) REST (Representation State Transfer) is a technology that can be used to build web services instead of SOAP. (Josuttis 2007)

Service registration and discovery are important functions in SOA. Web services are useful only if a potential user is able to find the services he needs. As the number of web services grow, it is needed to have a service registry to keep track what services has to offer and what are the characteristics of those services. UDDI was created to address these issues. It is based on industry standards including HTTP, XML, XML Schema, and SOAP. It is a public database of service descriptions and allows web services to be found by service clients. (Papazoglou 2008) UDDI contains services that support description and discovery of businesses, organizations and other web service providers. UDDI offers technical interfaces to access web services. It can be used for publicly available services and services that are used internally within an organization. (OASIS 2004)

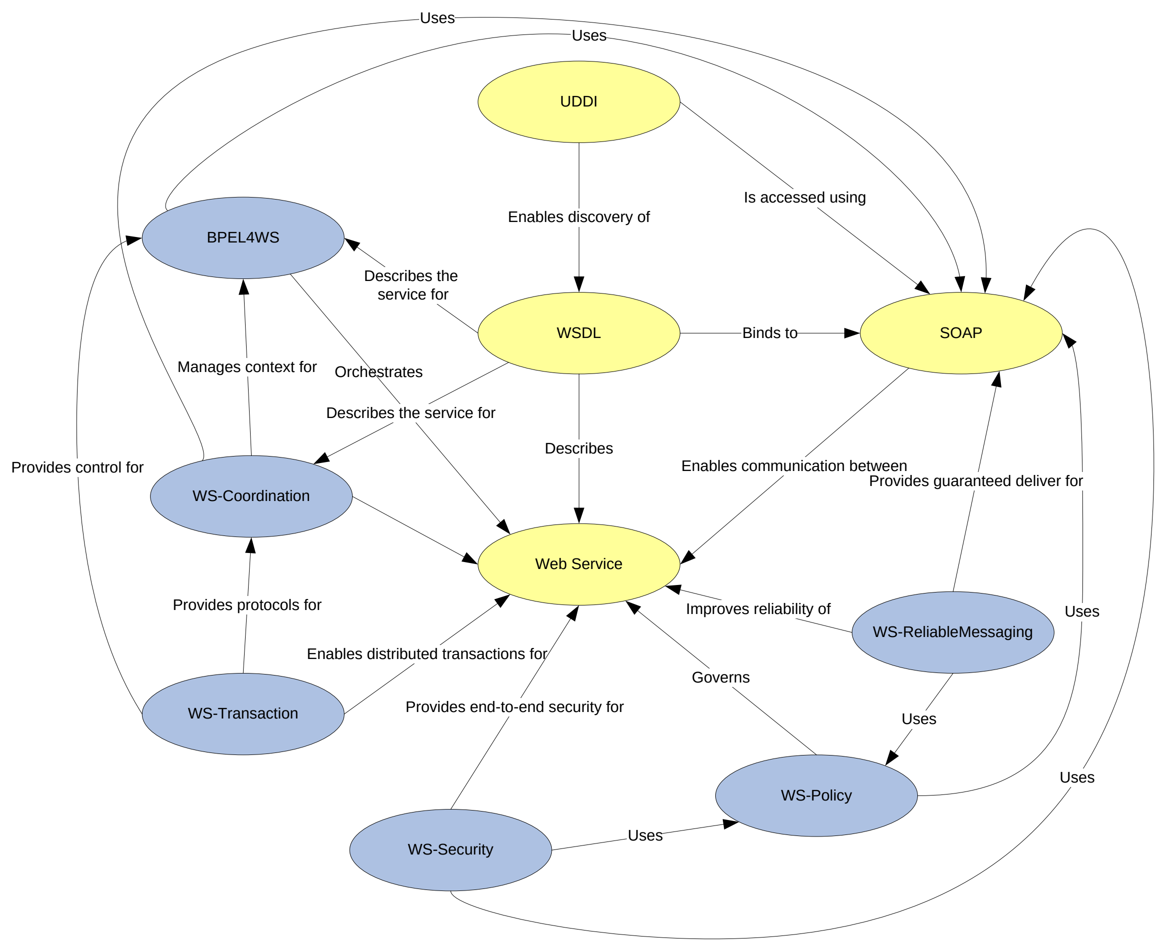
3.6 Contemporary SOA

Thomas Erl separates the concepts of primitive (or first generation) and contemporary (or second generation) SOA. Primitive SOA build upon core service standards but lacks the presence, for example, security issues. This becomes a serious problem when developing mission critical enterprise-level functionality. Issues like message-level security, cross-service transactions, and reliable messaging must be addressed. In the list below, some requirements for capabilities that need to be improved in primitive SOA are listed:

- The ability for tasks to be carried out in a secure manner, protecting the contents of a message, as well as access to individual services
- Allowing tasks to be carried out reliably so that message delivery or notification of failed delivery can be guaranteed
- Performing requirements to ensure that the overhead part of a SOAP message and XML content processing does not inhibit the execution of a task
- Transactional capabilities to protect the integrity of specific business tasks with a guarantee that if the task fails , exception logic is executed

According to Erl, Contemporary SOA is a complex and sophisticated architectural platform that has significant potential to solve many historic and current IT problems. The definition of contemporary SOA is as follows: *"Contemporary SOA represents an open, extensible, federated, composable architecture that promotes service-orientation and is comprised of autonomous, QoS-capable, vendor diverse, interoperable, discoverable, and potentially reusable services, implemented as Web services."*

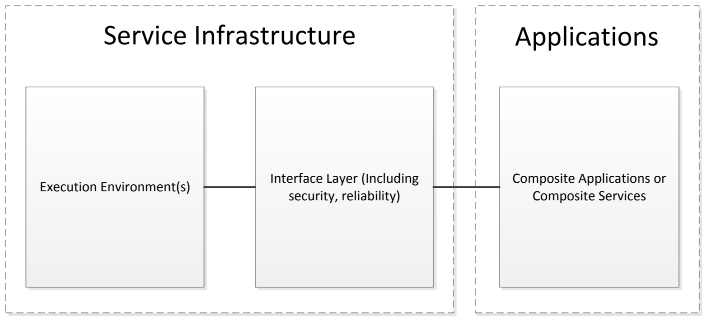
Contemporary SOA builds upon a primitive SOA by extending it with other web services technologies (labeled as WS-* Extensions). The purpose of these extensions is to improve the quality gaps of the primitive SOA. Each WS*- Extension realize certain design principles. It is notable, that like design principles, also extensions overlap and some of them are highly related to each other. Extensions are constantly evolving, which introduces more challenges in SOA design. Also, business partners and clients may have different maturity levels in specifications. This creates new challenges in SOA development. (Erl 2005) In picture 6, the first generation SOA is highlighted with yellow. Some of available WS*-extensions, that are needed in building of contemporary SOA are highlighted as blue.



Picture 3: Web relations between web service standards

3.7 SOA delivery process

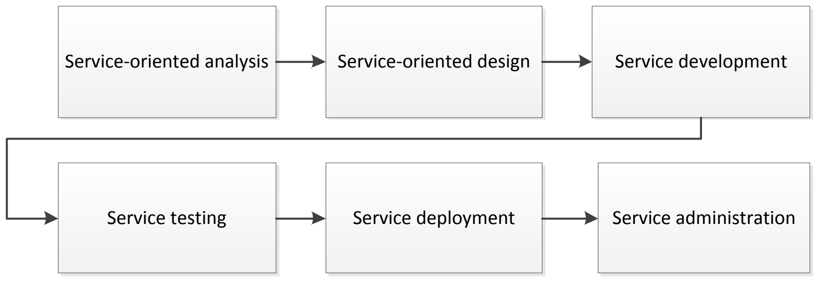
There are two major and separate tasks in the SOA delivery process. The first one is to create the service infrastructure with core services, management and security. The other is to build actual business applications that are based on the infrastructure. Both these tasks have their own development characteristics. (Haines & Rothenberger 2010)


Service Infrastructure
Applications
Execution Environment(s)
Interface Layer (Including security, reliability)
Composite Applications or Composite Services

Picture 4: Two major tasks of SOA

The service infrastructure is also called the Enterprise Service Bus (ESB). The ESB enables the communication of services between heterogenous systems with responsibilities including data transformation, intelligent routing, dealing with security and reliability, service monitoring and management. (Josuttis 2007) The infrastructure consists of interface layer and execution environments, where SOA applications are executed. Building the interface requires a long term global perspective. Because of the dual nature and characteristics of SOA, the development process of SOA differs dramatically from a tradional software development. (Haines & Rothenberger 2010)

Delivering SOA business application or creating enterprise architecture with SOA requires establishing a delivery strategy. The correct balance between long-term migration and short term requirements must be established. Different SOA delivery strategies exist. For example, in a top-down approach, all the phases of SOA delivery process are performed from beginning to end. More agile strategies can be used to build SOA in smaller iterations. Phases of SOA delivery strategy are described in picture 3. It is notable, that these phases are very much the same as the phases in a traditional software development process. (Erl 2005)



Picture 5: SOA delivery process

Traditionally, systems have been developed in a way that the analyst gathers the business requirements that are then handed over to architects and developers who then construct the system. In SOA, a direct relationship between business and technology exists, which means intensive co-operation between analysts and designers. Service-oriented analysis and design answers the questions such as what logic should be represented by services, how services should relate to existing application logic, how can services best represent the business process logic and how services can be built and positioned to promote agility. (Erl 2005)

Service-oriented analysis is a phase where the scope of SOA is determined. Service layers and individual services are mapped out to form a preliminary SOA. **Service oriented design** determines how the service candidates from analysis phase are designed by using standards, web service specifications and SOA design principles. **Service development** includes development platform specific issues such as how services that are produced by previous phases are constructed. Platforms for service construction include .NET and J2EE. **Service testing** is a crucial part in SOA delivery that is done before deploying services. **Service deployment** is determined by the chosen technology platform. This phase includes installing and configuring distributed components, service interfaces, and possibly associated middleware products onto production servers. **Service administration** includes application management issues. Monitoring the non-functional properties of services, version controlling the service descriptions and tracing and management of messages are tasks included in service administration. (Erl 2005) Managing complex service compositions require suitable tools for analysis, testing, and monitoring of the services. (Hull & Su 2005) When realizing

SOA it is also important to identify, which services are core business services that should be implemented internally and which services should be acquired from business partners. (Yunus 2010a)

3.8 Quality of Service

An important requirement for a SOA-based application is the desired level of Quality of Service (QoS) that is sometimes referred to as Quality of Web Service (QoWS). It means the ability of web service to perform its tasks based on the mutual expectations of both the service provider and the consumer. Quality factors need to be considered to keep business competitive and viable. The customer can choose a web service based on the promised QoS attributes. Because of the unpredictable and dynamic nature of the Internet and increased complexity of accessing and managing services, delivering QoS becomes a significant challenge. Applications with different characteristics and requirements compete for network resources. Changes in traffic patterns, hardware failures and reliability issues over the web create a need for securing mission-critical business transactions and managing QoS. (Yu et al. 2006)

Traditionally, QoS is measured by the degree to which applications, systems, networks and all other elements of the IT infrastructure support availability of services at a required level of performance under different access and load conditions. Focusing only on the functional properties of individual services is not enough - the whole environment where services are hosted must be considered. This means that non-functional capabilities of services must be described. In web services' context, QoS can be seen as providing assurance on a set of functional and non-functional service quality properties. The overall behavior of a web service must be understood so that other services can use it as a part of a business process. Run-time negotiable QoS is an advanced functionality made possible by web services. It means dynamically choosing a service provider among various service providers according to QoS criteria. (Yu et al. 2006)

Quality of Web service can be divided into two sub types: runtime quality and business quality. Runtime quality means the service's capability to execute its operations. Business quality allows the assessment of service operation from a business perspective. Both quantitative and qualitative measurements can be used to evaluate the QoWS. (Yu et al. 2006) Appendix 9.1 lists the key elements for supporting QoS in Web services environment.

3.8.1 Monitoring and Management

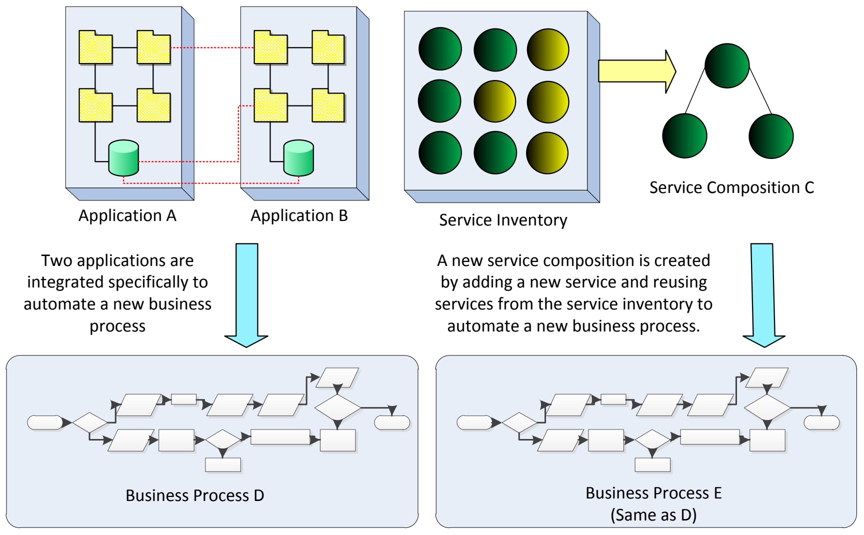
Web service management system (WSMS) must be established in order to monitor and control the quality of web services. Monitoring the behavior of a web service is necessary to assess its QoS parameters and ensuring that the service remains at a promised quality level. This activity is called monitoring management. The service quality can also be improved with a set of control mechanisms. Control management includes web service transaction and coordination; web service optimization and change management. Transactions help to improve Web services' fault-tolerance and reliability. Web service optimization allows clients to identify those services that best fulfill the desired requirements. In change management, certain actions are performed to identify the changes that are made to the service oriented system and adopt the appropriate operations when necessary. (Yu et al. 2006)

3.8.2 Service-level Agreement

A service-level agreement (SLA) is a formal contract between service provider and client. It describes the details of web service including price, contents, delivery process, acceptance and quality criteria and possible penalties of violating the agreement. SLA is used to ensure that the service provider delivers a guaranteed level of service quality. It describes also, how the quality of service is measured. (Baresi & Nitto 2007a) Appendix 9.3 represents the contents of a typical SLA.

3.9 Benefits of SOA

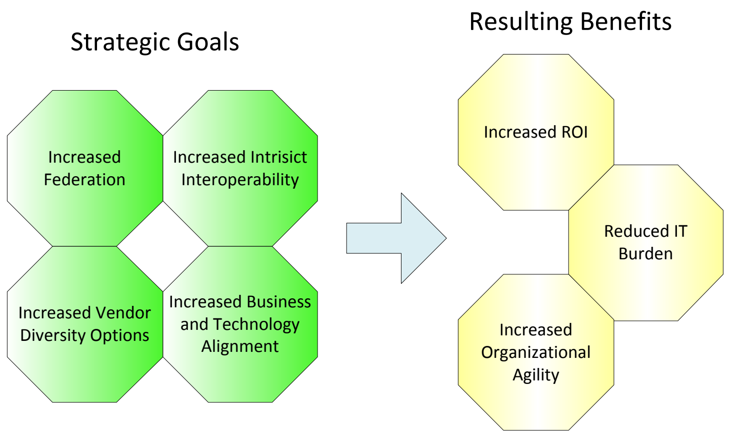
SOA organizes the systems so that they execute business needs in efficient way. If implemented correctly, SOA results an enterprise architecture that is agile and reusable. This means that the same application logic can be used over and over again and business processes can be changed without the changes needed in all systems. (Erl 2005; Erl 2008) In the picture 6, both traditional model and SOA model of automating a business process are presented.


Application A
Application B
Service Inventory
Service Composition C
Two applications are integrated specifically to automate a new business process
A new service composition is created by adding a new service and reusing services from the service inventory to automate a new business process.
Business Process D
Business Process E (Same as D)

Picture 6: Traditional model vs. SOA model (Erl 2008)

In order to traditionally automate the business process D, two applications are integrated with several point-to-point connections. This is troublesome if the business process changes and the need for new functionality appears in the future. As a result, there would be a system that is challenging to manage. In order to automate the business process E (that is the same as D) existing services found in service inventory can be utilized. The amount of new code written is probably less than in traditional model as only a few amount of new functionality is needed to be created. As a result, rapid reaction to changing business is met with the system that is less troublesome to manage. (Erl 2005; Erl 2008)

It is important to realize that SOA is not a technical architecture comprised of web services or simply implementing web services does not lead to all benefits that can be achieved through real SOA. Instead, SOA requires a mind-set change in which business logic is viewed from a service oriented context. (Erl 2005; Erl 2008) The key factors for successfully SOA are the understanding, governance, management and support. (Josuttis 2007) Going through a top-down transformation and approving a cultural change with foresight and commitment is the way to achieve real service oriented architecture with its benefits. The seven goals (represented in picture 7) of SOA can be categorized into two groups, strategic goals and resulting benefits. Strategic goals are increased federation, increased intrinsic interoperability, increased vendor diversity options and increased business and technology alignment. As they are realized, the three resulting benefits can be reached. These benefits are Increased ROI (Return of Investment), reduced IT burden and increased organizational agility (Erl 2005; Erl 2008)


Strategic Goals
Increased Federation
Increased Intrisict Interoperability
Increased Vendor Diversity Options
Increased Business and Technology Alignment
Resulting Benefits
Increased ROI
Reduced IT Burden
Increased Organizational Agility

Picture 7: Benefits of SOA (Erl 2008)

SOA requires a significant change of organizational environment as well as a significant amount of time and effort. SOA is not a silver bullet and it is not suitable for all situations. (Josuttis 2007) Some organizations have faced disappointments with their SOA projects. (Haines & Rothenberger 2010) Also, criticism against SOA can be found by looking at experts' blog writings. Some experts (Manes 2009; Cagle 2009) say that SOA has failed to fulfill the promises and expectations that have been set to it. They still believe that even if SOA would not exist anymore its principles will remain and be adopted by new technologies, like SaaS and Cloud Computing. According to Haines and Rothenberg, the most important question is how SOA can be done right in a current situation or is it possible or reasonable to do it at all.

4 CLOUD COMPUTING

Cloud computing is one of the biggest IT-trends at the moment. It is a result of decades of research and advances in technologies including virtualization, parallel and distributed systems, utility computing as well as recent advances in networking, web services and SOA. The main idea behind cloud computing is to provide computing as a utility. The nature of utility is that the users access it when they need and do not care where it comes from or how it is delivered. (AppLabs 2007; Gotel 2009; Buyya et al. 2009) In 1961, it was suggested that computer timesharing technology might lead to a future where computing power and even specific applications might be sold through a utility-type business model. (Rittinghouse 2010) Timesharing was a reasonable choice because of the high cost of computing and the need for specialized persons to maintain the systems. Because of the evolution of hardware, software, networking and communication protocols; a significant amount of unused computing capacity in organizations' data centers; high speed Internet data communications infrastructure as well as the companies' need to focus at their core competence, timesharing is reborn with a new name that is, cloud computing. (Durkee 2010)

Cloud computing is heavily affected by other paradigms and architectures, and technologies, such as grid computing, parallel processing, cluster computing and server virtualization. Technologies and concepts such as clustered computing, grid computing and utility computing can be considered as building blocks in order to achieve the modern cloud computing model that represents the next natural step in evolution of computing and IT services. Cloud computing is said to have the potential to create a paradigm shift in the way IT resources are used and distributed. It is also causing the delivery model shift from traditional desktop applications to software as a service model. (AppLabs 2007; Gotel 2009; Buyya et al. 2009)

Cloud computing is a new IT provisioning and support model that provides on-demand network access to a shared pool of computing resources. This pool of resources is called a cloud. The cloud can be located in one or more private data centers, in facilities operated by third-party service providers or across the combination of these two. (SPIRENT 2010b) Many different definitions for cloud computing can be found from the literature. According to Kommalapati, cloud computing is “computing capability delivered as a utility through Internet standards and protocols”. (Kommalapati 2010) The definition of cloud computing is predicted to evolve as cloud solutions become more mature and new service models appear. (Riungu et al. 2010)

Vaquero & Al. found a need for clear definition of cloud computing by gathering the over 20 definitions and making their own by identifying the key elements that appeared in most of the existing definitions. They define cloud computing as follows: *“Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized SLAs (service level agreement).”* (Vaquero et al. 2008)

4.1 Properties of cloud computing

Modern IT environment requires ways to increase the capacity and add capabilities to existing infrastructure dynamically, without investing money in purchasing new infrastructure, training new personnel and licensing new software. Cloud computing can offer a solution to these needs with the following set of its characteristics. ***On-demand access*** means rapid fulfillment of demand for computing as required and an ability to fulfill that need instantly. ***Elasticity*** is that computing is provided in the amount required and it is disposed when no longer needed. ***Pay-per-use*** signifies that a cloud service has a pricing model that charges the customer based on the amount of computing actually used. ***Connectivity*** means that all the servers are connected to a high-speed network that allows data flow to the Internet (to and from the cloud) as well as between computing and storage elements that are inside the cloud. ***Resource pooling*** implies that the cloud is shared across many end customers providing economics of scale at the computing and service layers. ***Abstracted infrastructure*** hides the exact location and the type of the hardware on which cloud applications are running from the customer. The cloud provider offers the metrics for performance to guarantee a minimum performance level. ***Little or no commitment*** means that by using a cloud solution, some of the maintenance responsibilities and commitments of resources can be moved to the vendor. (Rittinghouse 2010; Durkee 2010)

4.1.1 Benefits of cloud computing

Cloud computing is also a very controversial term and it has become a real buzzword in the media. For some people, it might appear as too marketing-oriented term invented by salesmen. Technically, cloud computing offers nothing new and similar promises have been heard before by allocated resource management, grid computing, on-demand computing and cluster computing. Cloud computing is also generating a great deal confusion about what it actually means – is it just some old ideas that are sold under a new term. In order to realize the benefits of cloud computing, it is important to understand what it really is, how it has evolved and what kind of opportunities are offered by today's cloud vendors. (Rittinghouse 2010)

The primary motivation for cloud computing utilization for a company is to lower the total costs of ownership of datacenters. (SPIRENT 2010b) Capital expenditures and maintenance costs are reduced as IT resources are centralized to a cloud and the need for own specialized hardware decreases. This can mean huge savings for a company that is now able to utilize major data processing and storing capability without having to invest large amounts of capital. Paying only for the amount of computing and storage that is actually used, companies are not required to have to host an environment that can handle the changing needs. Pay-as-you-go billing model also enables a company to save on costs as lengthy vendor contracts are no longer needed. (Linthicum 2010)

Cloud solutions can offer flexible, scalable and distributed infrastructures that allow applications to scale on demand. Instantly deployable environments for specific needs (such as for software testing) can be deployed in shorter timescales than before, that enables a company a quicker entry to market. Cloud computing is causing a transformation of IT department - from maintenance and implementation centric department to innovation centric. (AppLabs 2007; SPIRENT 2010b; Linthicum 2010) The time that is spent on customizing application frameworks or building and maintaining IT infrastructure could be used to create more value by focusing on core competence and improving the business logic. (Rittinghouse 2010)

Enterprises with large datacenters have realized that they cloud sell their extra capacity as a service and increase their server efficiency and utilization producing greater ROI. (Linthicum 2010; SPIRENT 2010b) As small and medium businesses cannot necessarily invest in their own datacenters, and might not have the expertise to manage them, outsourcing of data centers is becoming more popular in a form of cloud computing. (SPIRENT 2010a) Small and medium enterprises can enter to the same competition with large companies, because they are now able to access vast computing resources with no capital investment and because these resources can now be accessed from anywhere, anytime. (Greengard 2010)

4.1.2 Risks of cloud computing

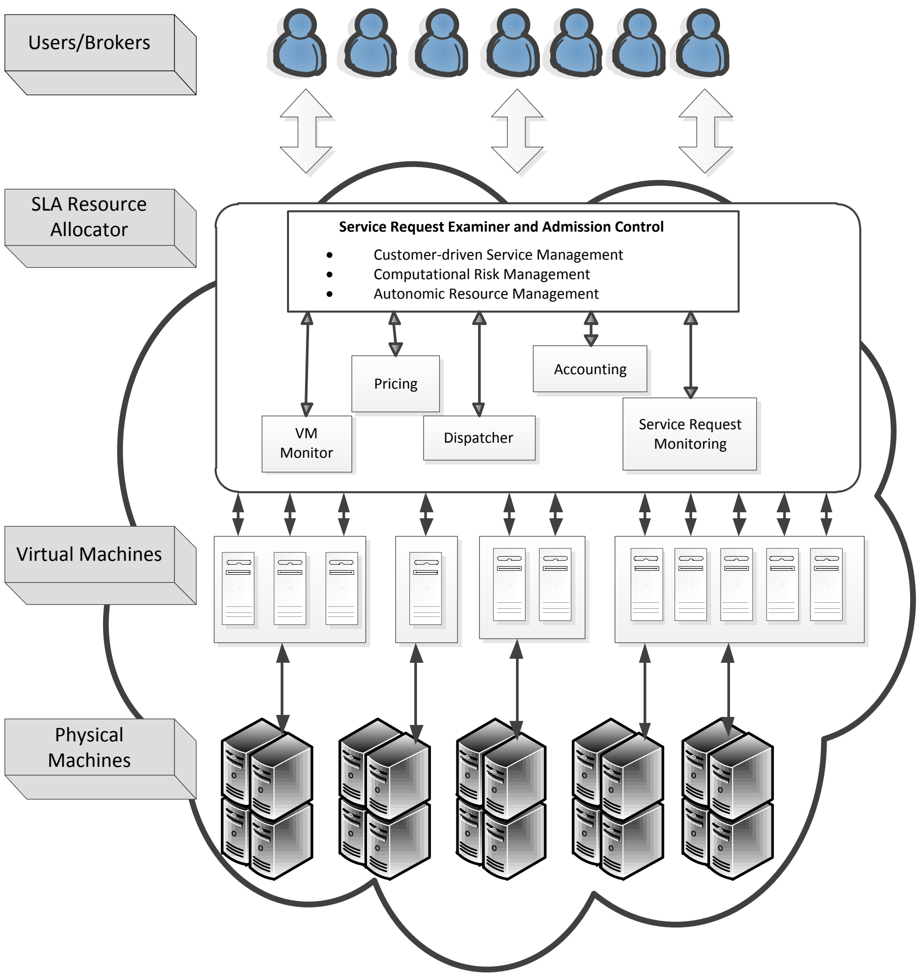
Besides all the benefits and opportunities, cloud computing introduces several risks and challenges that need to be evaluated. These issues include security, lack of control, privacy concerns, data integrity and availability as well as business acceptability. Outsourcing any part of own operation to third parties causes lack of control. It must be considered, how will the business retain and maintain the control of data and handle the impact of possible downtime as well as the possible technology changes or decisions made by the cloud vendor. There is always a possibility that sensitive information will end up in wrong hands. Data privacy and integrity are also notable concerns - does a business maintain the privacy of their users and does the valuable data remain intact in a cloud environment as well. Business critical cloud solutions must be available all the time and service provider must somehow guarantee the availability. The suitability of the third party solution must also be estimated. (AppLabs 2007) If thinking about a larger scale of suitability, in regions that do not have much IT-resources and bandwidth, and that are coping with limited technical expertise, the utilization of cloud computing may be difficult. (Greengard 2010)

4.2 Architecture of the cloud

Virtualization is very often related with cloud computing. However it is technically possible to build a cloud without virtualization technology, especially in cases of SaaS. The flexibility and automation granted by virtualization are the key enablers that create highly scalable and dynamic cloud environments that are able to need enterprise-class needs and can be customized by end-user. The elasticity of cloud computing is also made possible by virtualization technology. Typical components of the cloud include virtualization software, load balancers and networking gear. Load balancers can be used to dynamically direct user application requests to the servers that can best handle the requests at the moment. Data center networking gear of high capacity is needed to provide the constant realignment of network resources that are accessed across LANs (local area network), SANs (storage area network), and WANs (wide area network). Cloud computing sets significant demands on the networking infrastructure and upgrades may be needed in order to support increased amount of elastic and real-time network traffic. (SPIRENT 2010b)

The idea of virtualization is simple: a server with CPU (Central Processing Unit), disks and network interface spend significant amount of time idling. Thus it is reasonable to use free resources to offer more services with the same hardware. (SPIRENT 2010a) With virtualization, the operating system and application software are decoupled from the underlying hardware platform and from other virtual machines. (SPIRENT 2010b) The resources of a physical server can be used to create different virtual machines thus making it possible to create many user domains on a machine level. (SPIRENT 2010a) Virtualized software can be migrated in one cloud between servers with minimal interruption or it can even be moved between different clouds – wherever computing cycles are available on the physical infrastructure at the moment. This allows servers and data centers to be consolidated and utilization rate of physical servers to be increased. (SPIRENT 2010b) Virtualization software resides between the physical machine and the operating system and allows a single physical server or other piece of hardware such as hard disk or network device appear as many separate devices. In a modern data center it is typical to use virtualized networking devices. (SPIRENT 2009)

End user can install desired applications to the cloud environment. Providers can offer the access to applications running in virtual machines or they can provide access to virtual machines themselves. Providers have to manage the virtualized environment so that intelligent allocation of physical resources based on competing demands for resources is handled correctly. Cloud service consumers require fast response times. Distributing cloud services into various locations and dividing the workload of request to these locations may result better response time. (Buyya et al. 2009)


Users/Brokers
SLA Resource Allocator
Virtual Machines
Physical Machines
Service Request Examiner and Admission Control
Customer-driven Service Management
Computational Risk Management
Autonomic Resource Management
Pricing
Accounting
VM Monitor
Dispatcher
Service Request Monitoring

Picture 8: A High-level architecture of a market-oriented cloud

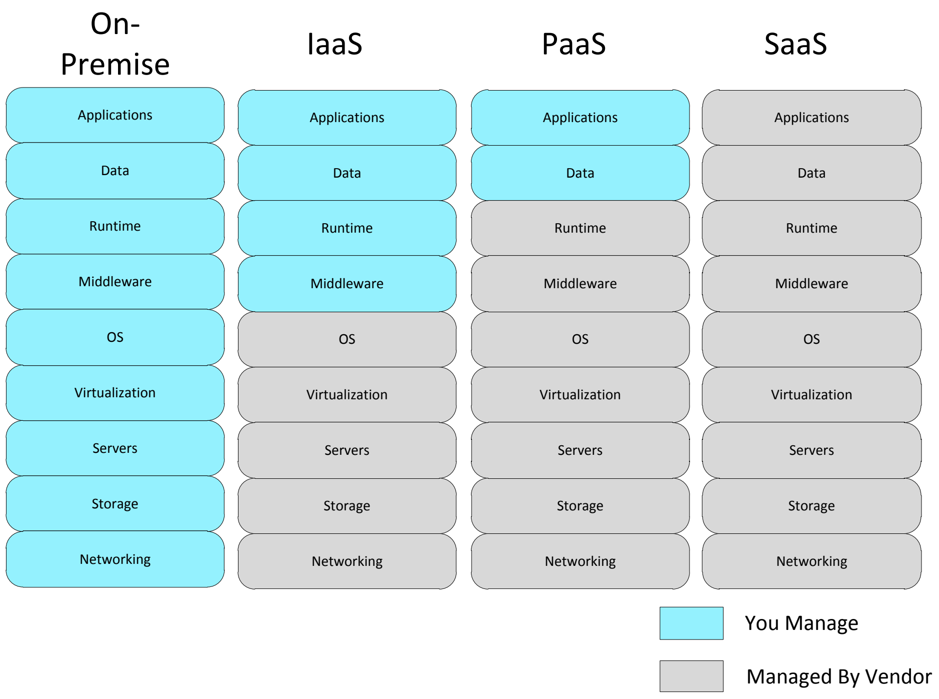
In the picture 8, users and brokers access the cloud via the Internet. The SLA resource allocator is the interface between the data center and its users. It requires the support of several mechanisms to enable SLA-oriented resource management. When a service request arrives, the **service request examiner and admission control** mechanism examine the QoS requirements (such as time, cost, reliability, trust, security) of the request before determining whether to accept or reject the request**. VM monitoring** mechanism gives the latest status information about the availability of resources. Information about workload processing is also needed from the **service request monitor** mechanism. Then, the request is assigned to VMs. The pricing mechanism determines, how service requests are charged. **Pricing** can be based on submission time (peak/off-peak), pricing rates (fixed/changing) or availability of resources (supply/demand). The accounting mechanism keeps track of the actual usage of resources so that the total cost can be calculated and charged to users. **VM monitor** keeps track of the availability of VMs and their resource entitlement. **Dispatcher** starts the execution of service request that is accepted on allocated VMs. Service request monitor keeps track of the progress of the service request being executed. Any number of **virtual machines** that reside on a single physical machine can be started and stopped on demand to fulfill the accepted service requests. VMs provide flexibility to configure resources on the same physical machine to different requirements of service requests. Because VMs are isolated from each other, they can run applications based on different operating systems concurrently on a single physical machine. The actual computing of service requests happens on **physical machines**. (Buyya et al. 2009)

4.3 Cloud types

There are different types of clouds: private, public, community/federated and hybrid cloud. **Private clouds** are meant for exclusive use of a business or a consortium of businesses. (Kommalapati 2010) In this case, the cloud can be managed by the organization itself or by a third party. According to Kommalapati, “any private datacenter that is run by a large enterprise can be called a private cloud if it utilizes the unified resource model enabled by broader virtualization that treats compute, storage and networking as a homogenous resource pool and takes advantage of highly automated processes for operating the system.“ **Public Clouds** are available for anyone. The infrastructure of the cloud is usually owned by a large organization such as Amazon, Google, IBM or Microsoft. The Windows Azure platform, Amazon Web Services, Google App Engine and Force.com are examples of public clouds. (Kommalapati 2010) **Hybrid Cloud** is a combination of private and public cloud services. A company that is operating its own private cloud may get some resources from public clouds, for example, in a situation when the computing power needed exceeds the own capacities. This is called cloudbursting. This mix of cloud models can change dynamically as customer requirements change. **Community/federated** clouds are shared by several organizations that have similar business needs and interests. For example, public safety personnel could share emergency-response clouds for disaster relief efforts. (Louridas 2010)

4.4 Cloud service models

Various service models of cloud computing exist, depending on the type of the capability provided by the cloud. Infrastructure as a service (IaaS), platform as a service (PaaS) and software as a service (SaaS) are the main service models of cloud computing introduced in most literature sources. On-Demand computing is an enterprise model in which computing resources are made available to the users as needed. Traditionally, computing resources are maintained on customer's site but in on-demand model, resources are mostly maintained by the service provider. (Rittinghouse 2010) As resources are obtained from a cloud, the responsibility to manage the resources shifts from the customer to the vendor. (Chou 2010) Picture 8 shows the management responsibilities of certain IT assets in different cloud service models.



Picture 9: Management responsibilities of cloud service types (Chou 2010)

The term XaaS (everything-as-a-service) can be seen very often in cloud computing literature. These are not considered in current definitions of cloud computing since cloud computing is an evolving trend. Some researchers believe that the cloud computing in the future will cover additional service models, for example for example HuaaS (Human as a service) and STaaS (Software Testing as a Service). (Riungu et al. 2010)

4.4.1 Infrastructure as a Service (IaaS)

In IaaS model, the predefined and standardized computer infrastructure, which is typically a platform virtualization environment, is delivered as a service. The provider of IaaS gives a business customer a generic hardware foundation on which a customer can install and run his/her own operating systems, applications and storage data. (SPIRENT 2010b) This hardware foundation consists of computer hardware (that is usually a scalable grid), computer network (routers, firewalls, load balancing), Internet connectivity, platform virtualization environment running for virtual machines that are specified by the client and SLAs and billing based on the usage (pay-per-use). (Buyya et al. 2009) IaaS is generally offered as an unmanaged service. A business rents physical and virtual servers as needed, and customer has a full control of the software configuration, which means both ownership and management of the application. IaaS providers house, run and maintain the platform and manage the transition of the customer's applications on the infrastructure. Providers are also responsible for infrastructure management and off-loading hosting operations. (SPIRENT 2010b)

IaaS saves customers from making investments in the underlying foundation of servers, storage devices and network connections, as customers' business grow and requirements change. (SPIRENT 2010b) Instead, customers rent those components as an outsourced service and customer is charged only for resources consumed, for example, once in a month. Customers are able to build, test and deploy applications with full awareness of the hardware and software configurations of the servers hosted by IaaS providers. IaaS allows customers to transfer the legacy applications to the cloud. (Kommalapati 2010) Different infrastructures provide different abstractions and mechanisms for describing and accessing VMs and storage. Compatibility between different IaaS vendors might become an issue. Developers using IaaS also have to consider some platform specific details, for example, support for parallelism and I/O details offered by an IaaS vendor before moving applications to the cloud. (Louridas 2010)

4.4.2 Platform as a Service (PaaS)

Applications need a platform on which they run. Platform provides services for a developer to create applications and store data. (Chappel 2010) PaaS (Platform as a Service) is another type of unmanaged cloud service. PaaS offers all the necessary building blocks, including the tools, libraries, application programming interfaces (APIs), protocols, operating systems and storage that are required to support the complete lifecycle of building and delivering web applications from the Internet. (Kommalapati 2010; SPIRENT 2010b)

IaaS developers are able to create a specific operating system instance running applications that they have developed in their own environment. This is the major difference between IaaS and PaaS where developers can only focus on web-based development without caring which operating system is used. Developers can access massive computing power and applications can deployed globally reaching the large segment of users. (Kommalapati 2010) PaaS developers can use the cloud to store their code and data and also use the cloud as a channel for distributing the software to consumers. (SPIRENT 2010b) Consumers of PaaS are billed with pay-per-usage model and prevents them paying for massive upfront capitals and software licensing investments, that can sometimes be a barrier to innovation. Developers need to be aware of programming languages available for specific platforms as well as the version numbers of frameworks. Also, a support for relational databases may differ between PaaS vendors. Developers need to get themselves familiar with programming abstractions that may be specific to the platform. PaaS framework can provide specific abstractions for system elements such as data storage and access, data querying and message queues. These abstractions may not be portable across PaaS providers. (Louridas 2010)

4.4.3 Software as a Service

Software as a service (SaaS, or sometimes called the online delivery of software) is an alternative for traditional software delivery model called Software as a product (SaaP) where customer purchases software licenses, installs application on his premises and manages configurations, patches and version updates. (Dubey & Wagle 2007; CIOL 2008)

In the SaaS model, applications become available in the form of network-based service and users access them via Internet by a web browser. The customer does not buy a software license, instead, he signs up to use the application that is developed, hosted, managed and sold by a SaaS vendor. (SPIRENT 2010b) The customer does not have to install or maintain any piece of software. (Reese 2009)

The user of SaaS application does not care where the application is hosted, what is the underlying operating system it runs on, or is the application written in .Net, Java or PHP. SaaS is usually considered as a fully managed service. This means that the service provider maintains both the software and hardware environment on behalf of the customer that is now relieved from having to do any software customization, version tracking, patching and updating. (SPIRENT 2010b) Term SaaS is often referenced as software in a cloud and it is also considered as one service model of Cloud Computing. However, not all SaaS solutions are cloud solutions**.** (Reese 2009) SaaS is an inexpensive way for a customer to get access to the needed software by using on-demand licensing without the associated complexity and possibly high initial costs compared to traditional delivery model. (SPIRENT 2010b) Main characteristics of SaaS are on-demand availability via a web browser, payment terms based on usage, and minimal IT demands. (Reese 2009; Dubey & Wagle 2007; Salesforce 2011b)

Some benefits of SaaS model can be identified from both customer and vendor point of view. By acquiring software as an online service, the customer can focus on its core competence functions. Software can be taken into usage earlier, because no installation on customer's environment is needed. SaaS releases the customer from making long time contracts with the service provider. By paying monthly fees, customer can quit using the application at any time and switch to another vendor. No capital is required for purchasing licenses and possibly new hardware to run the software. This can be a real opportunity for small companies that are not necessarily able to purchase SaaP because of starting costs.

Independent Software Vendors (ISVs) are able to access new markets of smaller customers, who previously were not able to purchase the software. (Dubey & Wagle 2007; Oracle 2008)

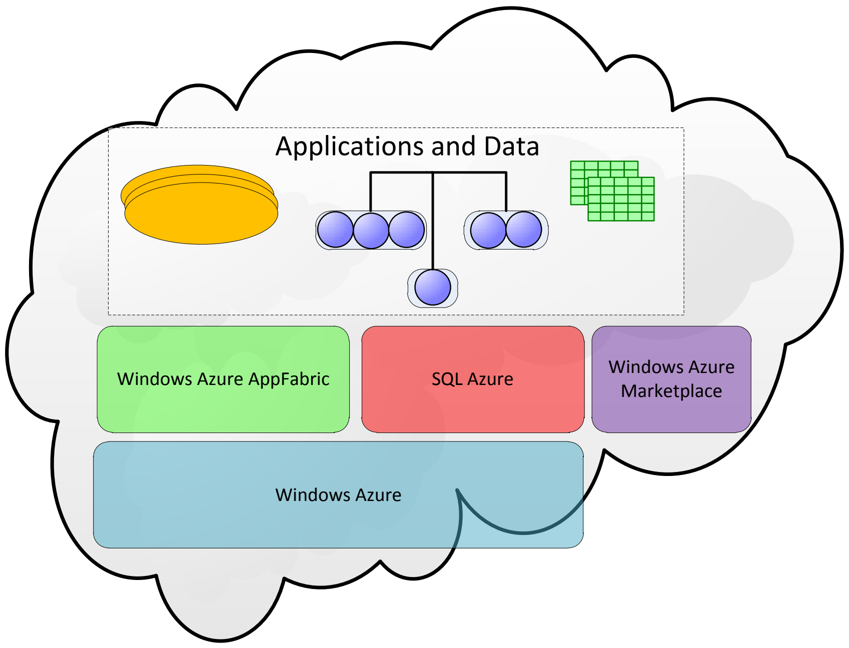
SaaS creates new challenges for independent software vendors, hosting services provides and customers. Technical challenges include support for multi-tenancy, the degree of customization and service-level management. (Oracle 2008) SaaS model does not come without risks involved. Not all types of applications are suitable to be offered as a service. Especially, business critical systems may not be very optimal to be acquired as SaaS. Security becomes a major concern as sensitive data is being stored in vendor's premises. Reliability of a SaaS provider needs to be ensured. Technical solutions of a vendor may cause problems in availability, reliability, security and performance. (Vainikka 2010) Customer's ability to customize SaaS applications vary. If a fully customizable application is needed, the limitations of SaaS model may be a problem and either IaaS or PaaS may need to be considered. (Kommalapati 2010) Some SaaS providers allow certain customization options, for example, Salesforce.com which offers CRM (Customer Relationship Management) and other business software in SaaS model. (Salesforce 2011) Another example of SaaS is Google Apps which provides certain office applications to the user. (Google 2011a)

4.5 Current Cloud services

Vendors, such as Microsoft, Google and Amazon have played a significant role in the evolution of cloud computing. (Rittinghouse 2010) Numerous cloud services from increasing number of service providers exist. In following chapters, Windows Azure Platform, Amazon EC2 and Google App Engine are examined briefly.

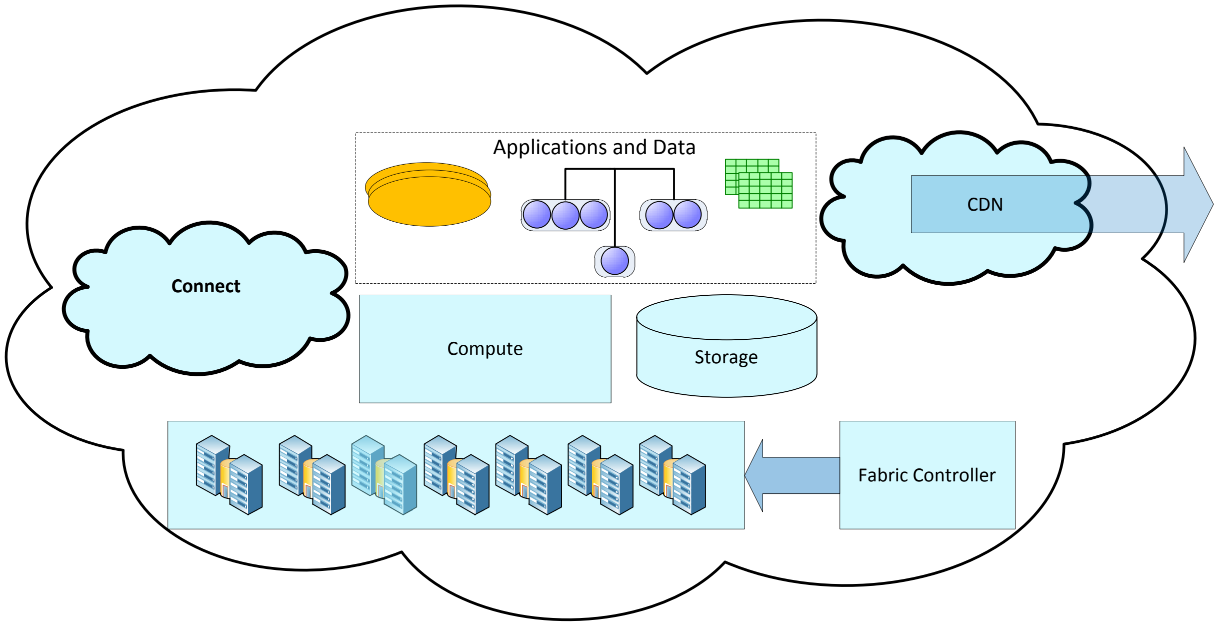
4.5.1 Windows Azure Platform

Windows Azure Platform is a group of technologies that provides a specific set of services to application developers. (Chappel 2010) Windows Azure Platform offers a platform where ISVs (Independent Software Vendor) and enterprises can host their applications and data. The applications can provide services for businesses, consumers or both. Windows Azure applications run in Microsoft data centers and are accessed via the Internet. (Chappel 2009) Applications that are written for Windows azure can scale better, be more reliable and require less administration than applications written using the traditional Windows Server programming model. Applications can be created, configured and monitored via a browser-accessible portal. Azure provides an environment that is interoperable and is based on standards and protocols such as HTTP/HTTPS, REST (Representational State Transfer), SOAP, and XML. REST and SOAP allow developers to interface between Microsoft and non-Microsoft tools and technologies. (Kommalapati 2010) Customers are billed by how much computing time, storage and bandwidth they use. (Chappel 2010)



Picture 10: Main Components of Azure Platform

Picture 10 describes the main components of Azure Platform. Windows Azure is a Windows environment for running applications and storing data on computers that reside in Microsoft data centers. **SQL Azure** is based on SQL server and offers relational data services in the cloud. **Windows Azure AppFabric** is a set of infrastructure services that can be utilized by applications running in the cloud or on-premises. AppFabric provides applications with means to name, discover, expose, secure and orchestrate web services, allowing developers to focus on their application logic rather than building and deploying their own cloud-based infrastructure services. AppFabric includes three components: Service Bus, Access Control and Caching. **Azure Marketplace** is an online service from where cloud-based applications can be purchased. All the components run in Microsoft data centers around the world. Developers are able to control, which data center is used for running applications and storing data. (Chappel 2010)



Picture 11: Main components of Windows Azure

Windows Azure and its components are presented in picture 11. **Compute** runs applications in the cloud in a Windows Server environment. Applications can be created with .NET languages or with C++ or Java. Developers can use Visual Studio or other development tools and they are able to choose whether to use ASP.NET, WCF (Windows Communication

Foundation) or PHP. **Storage** stores both binary and structured data in the cloud. Data elements in Windows Azure storing are blobs, queues and tables. SQL Azure can be used if traditional relational storage is needed. Storage service can be used by both Azure applications and on-premises applications using REST (Representational State Transfer). The purpose of **fabric controller** is to deploy, manage and monitor applications, and also handle updates to system software throughout the platform. **Content Delivery Network** exists for speeding up the global access to binary data stored in Windows Azure storage by maintaining cached copies of that data. This can be done for blobs. **Connect** module allows organizations to interact with cloud applications as if they were inside the organizations own firewall. It allows IP-level connections between Azure applications and on-premises machines. For example, an Azure application can access data bases located on-premises. (Chappel 2009; Chappel 2010)

4.5.2 Amazon Elastic Compute Cloud (EC2)

Amazon Elastic Compute cloud provides its users resizable computing capability in the cloud. Its goal is to make web-scale computing easier for developers. A user has control to scale capacity based on his needs and the user pays only for the capacity that is used. According to Amazon, "the developers are provided with tools to build failure resilient applications and isolate them from common failure scenarios". EC2 is a true virtual computing environment. By using web service interfaces, a user can launch instances of variety operating systems, load them with custom application environment and manage the access permissions of network. A user can select a pre-configured image from templates or the user can create an Amazon Machine Image (AMI) that contains desired applications, libraries, data and associated configuration settings. Security and network access to EC2 instance can be configured. A user can start any number of instances and by using the web service APIs and management tools provided, instances can be monitored. Following list contains the benefits of Amazon EC2: (Amazon 2010)

- **Elastic:** The used capacity can be easily adjusted. One or thousands of server instances can be utilized and controlled with web service APIs. An application can scale itself up and down depending on current needs.
- **Completely controlled:** Server instances can be controlled. User interacts with service instances as he would with any machine. Instances can be stopped and rebooted remotely.
- **Flexible:** The user is allowed to choose from many types of instances, operating systems (Linux distributions, Microsoft Windows Server, Open Solaris among others) and software packages. The configuration of memory, CPU, storage and boot partition size can be selected.
- **Designed to use with other Amazon Web Services:** These services include Amazon Simple Storage Service (Amazon S3), Amazon Relational Database Service (Amazon RDS), Amazon SimpleDB and Amazon Simple Queue Service (Amazon SQS)
- **Reliable:** EC2 runs within proven network infrastructure and datacenters and the promised SLA for availability is 99.95%.
- **Secure:** Numerous security mechanisms are provided. Firewall settings can be configured to control network accesses. Amazon Virtual Private Cloud (Amazon VPC) can be used to isolate instances by specifying the desired IP range. Also, a connection to existing IT infrastructure is allowed by using Ipsec (Internet Protocol security) VPN (Virtual Private Network).
- **Inexpensive:** Financial benefits are promised. User pays only low rates for the computing capacity actually consumed. (Amazon 2010)

4.5.3 Google App Engine

Google App Engine allows users to run web applications on Google's infrastructure. According to Google, "App Engine applications are easy to build, easy to maintain and easy to scale as your traffic and data storage needs to grow". There is no need to maintain servers or to worry about hardware, patches or backups when using Google App Engine. The user only has to upload applications on the platform. Applications can be shared with the world, or access can be limited to specific groups of users. Google App Engine has both Java and Python runtime environment and applications can build with several programming languages. Runtime environments are built to ensure that applications run quickly, securely and without interference from other apps on the system. Environments provide standard protocols and common technologies for web application development. The pricing method is pay per use without no set-up costs or recurring fees. The storage and bandwidth are measured by the gigabyte, and maximum amount of resources that applications consume can be controlled. All applications are allowed to use 500 MB of storage and CPU and bandwidth that enables 5 million page views in a month for free. When more storage of CPU is needed, the user pays only the amount exceeding the free levels. A web based Administration Console is provided so that the user is able to manage his running applications. Google App Engine includes following features: (Google 2011b)

- Dynamic web serving with full support for common web technologies
- Persistent storage with queries, sorting and transactions
- Automatic scaling and load balancing
- APIs for authenticating users, sending email using Google Accounts and using other services such as data store, URL fetch and image manipulation
- A fully featured local development environment that simulates Google App Engine on the user’s computer
- Task queues for performing work outside the scope of a web request
- Scheduled tasks for triggering events at specified times and regular intervals (Google 2011b)

5 Testing Service-Oriented Solutions

The role of SOA and web services is increasing as e-commerce, business-to-business, business-to-consumer and other information based services are becoming more important in the domain of IT. (Baresi & Nitto 2007a) As systems grow out from own enterprise and connect with other enterprises' systems, effective approaches to ensure high level security and reliability are needed, especially in mission critical systems. A successful testing effort determines whether or not the deployment of SOA is successful. The quality of deployed services must be secured, tested and managed throughout the SOA project. SOA testing must cover all the levels of testing from individual services to service compositions and include both functional and non-functional testing. (Lucia & Ferrucci 2009) Web services and SOA introduce a new level of complexity and certain unique features that challenge the traditional software development and testing methods. (Baresi 2007; Haines & Rothenberger 2010; Lucia & Ferrucci 2009) SOA testing can be seen as a complex computing problem that needs to be broken down into smaller components. (Harris 2007)

According to the study made by Haines and Rothenberger (2010) there are significant differences between a software development process of SOA with web services and a traditional software development process. Changes to all development phases were identified and testing was seen as a phase that will face potentially substantial changes. Unique features of SOA make the whole software development environment different. Because of the two sides of SOA development (building the infrastructure and building applications), possibly two testing environments are needed. As applications may comprise of many services, any of them may became a single point of failure in a composition application. Because of this, more integration testing must to be done in a SOA project. Some testing activities are specific to web services that bring many new technologies and specifications with them. An example of this kind of activity is compliance testing that is, checking if web services comply with the web service standards or WS-I Profiles. Also, it was identified that the test plans need to consider different application types (web application, mobile application, desktop application) that might use the service. Testing tools are needed to automate certain testing tasks and they must address both functional and non-functional issues, for example, security and performance. The testing process needs to be coordinated all the stakeholders and it must exist throughout the entire life cycle of the SOA project.

5.1 Testing Challenges in SOA

SOA brings out unique features that make software testing more complex and challenge traditional testing approaches. Existing testing techniques, methods and tools may not be directly adaptable in testing SOA solutions. Some assumptions used in testing of traditional systems do not hold true in SOA testing. For example, one cannot always identify the actual piece of code that is invoked at a given call-site or that all the possible bindings of the polymorphic components are known beforehand. (Lucia & Ferrucci 2009) Old approaches need to be adapted and also new methodologies are needed for SOA testing. (Dumke et al. 2008)

Unique features of SOA include run-time discovery of services, ultra-late (dynamic) binding, QoS aware composition and automated SLA negotiation. Because of run-time discovery and ultra-late binding, the actual configuration of system is known only during the execution, not beforehand. This challenges existing integration testing techniques. QoS aware composition and automated SLA negotiation mean that the service offers different performances in different contexts, causing challenges for performance testing. (Lucia & Ferrucci 2009) According to Palacios et al (2010), one of the biggest challenges in SOA testing is the dynamic behavior of services.

A single service in a SOA solution may be reused in order to compose other applications. (Harris 2007) The applications and contexts where the services will be reused are not known at the time when individual services are developed and tested. (IBM 2007) Reusability of SOA suggests more testing effort to be done at service level, because reusable services are used in many solutions. Services that have bugs and quality issues cannot be reused. (Harris 2007) Moreover, if a single service will be reused in multiple compositions and is not tested well enough, the impact of failure can be catastrophic as multiple organizations and business solutions are affected at the same time. (IBM 2007) This will definitely emphasize developers to put even more effort to service-level testing but also emphasizes integration testing. Functionality, performance and reliability of both individual services and service compositions must be verified. (Xiaoying Bai 2005) According to Baresi et. al. (2007) testing service compositions while considering security issues is one of the key challenges in SOA testing.

Lack of the observability of the service source code and structure has been identified as one key issue that reduces the testability of service-oriented solutions. (Lucia & Ferrucci 2009) Availability of the source code and control of services are not always guaranteed in SOA solutions. Some services are located outside the organization and utilizing services that are not deployed in tester's infrastructure may cause extra costs. Total costs derived from testing can be hard to predict. (Palacios et al. 2010) The lack of graphical user interfaces (GUI) of services under test may make testing more difficult. (Dumke et al. 2008) This forces the testers to deal with messages and protocols based on web services standards instead of GUIs. On the other hand, with traditional applications, the user interface keeps changing frequently during the development of software. This means that the automated test cases need to be updated. In SOA, the test cases are based on the service layer and are independent of the changes of the user interface. This increases the maintainability and will save costs and time. (IBM 2007)

For users and system integrators, services appear as interfaces that hide all the details of the underlying business logic. Third-party services can only be viewed as black boxes, preventing white-box testing approaches from being used. If deeper information (for example, information about dependencies between service operations or a behavior model of the service) is needed, either the service provider must provide this information or the tester has to obtain it by observing the service from outside. (Lucia & Ferrucci 2009)

In traditional systems, components and libraries are integrated physically in a software system. With services, the case is different. Remote services are not physically integrated into a provider's infrastructure. System integrators cannot decide a strategy to migrate to a new version of a service and regression testing of the system. Any changes made to a service may not be informed to integrators and users that are accessing it. This lack of control in regression testing is an issue that heavily reduces testability of SOA systems. (Lucia & Ferrucci 2009) The lack of well-defined boundaries and control over the whole IT solution that is under test creates challenges in overall testing. (IBM 2007)

The service provider provides certain information about the service. This information can include service descriptions and QoS information. Integrators and service users can utilize this information when determining which services to use. Integrators need to decide, whether or not to trust the information retrieved from service providers. This kind of lack of trust is another issue that is associated in testing of SOA systems. In some cases, invoking services may introduce extra costs, if services are charged on a pay-per-use basis. Moreover, provider may experience decreased performance or denial-of-service phenomena if massive testing is done against the service. These kinds of side effects also complicate SOA testing. (Lucia & Ferrucci 2009)

5.2 SOA testing process in general

Like a traditional testing process, a SOA testing process needs to be a well-planned and systematic activity. Testing must not be seen as an end-of-the-life-cycle activity. Instead, SOA project must have a test methodology, which emphasize testing throughout the project life cycle, for example the V-Model. Because of the increased complexity, SOA requires even bigger portion of the development time spent on testing and quality assurance. (Haines & Rothenberger 2010) Testing as well as other development activities in a SOA project should be business-driven. (IBM 2007)

All the levels, including unit, integration and regression testing, need to be considered in SOA testing. The importance of non-functional testing increases dramatically in SOA testing process.s The presence of quality attributes (security, reliability, performance, interoperability, vulnerability) need to be verified by performing non-functional testing and to ensure that SOA solution is interoperable, robust and scalable. When integrating critical business systems, the importance of security is essential. The planning of security testing needs to be started early enough. (Baresi & Nitto 2007b) Different perspectives and stakeholders (developer, provider, broker, user) participating in a SOA testing process must be identified. (Harris 2007) Monitoring approaches have been identified as the most important testing technique to improve the dynamic binding of web services as they are being executed. In monitoring, the functional and non-functional properties of the system are examined and appropriate actions are performed based on the results. (Palacios et al. 2010) The importance of static testing techniques at the early stages of the project must be recognized as well.

SOA with Web services introduces numerous standards and specifications that the developers and testers of SOA solutions need to be aware of. Service descriptions made with

WSDL can be used to generate test cases. In order to increase the testability of SOA, it has been proposed to provide additional information with WSDL specification by adding additional WSDL schema types to the service description. WSDL can be extended with QoS attributes. Also, the information about dependencies between service operation inputs and outputs, invocation sequences when a service delegates its responsibilities to other services and functional description of the service can be described in WSDL. OWL-S (Ontology Web Language for Web Services) and SAWSDL (Semantic Annotation for WSDL) are semantic technologies that can be used to describe the service behavior. BPEL (Business Process Execution Language) can be used to describe web service compositions and to turn workflow of business process models into executable code. (Palacios et al. 2010; IBM 2007) A very common testing scenario in testing web services is to test services against the service level agreements, whether or not they conform or violate the SLA that has been established between the provider and the client. Currently there is no standard language to specification of SLA. For this purpose, WSLA (Web Service Level Agreement) or WS-Agreement can be used. Policy assertions about the QoS can be specified with WS-policy. (Palacios et al. 2010)

The standardized nature of SOA makes it possible to utilize SOA testing tools that can significantly reduce the total costs of testing. (Haines & Rothenberger 2010) Many tools for SOA testing exist. Without automating the creation and execution of most of the tests, testing becomes inefficient and expensive. (IBM 2007) Functional, performance and security regression suites are needed in testing. (Harris 2007)

5.3 Roles in SOA testing

Different roles participating in SOA testing can be identified. The roles include developer, provider, integrator, certifier (or broker), and end-user (or client). Each stakeholder involved in the SOA testing process has different needs and challenges. (Lucia & Ferrucci 2009)

Depending on the role of a stakeholder that is testing the SOA application, the goal of testing may be different. It may be detecting errors in the application or it may be making the decision about which service will be invoked based on the test results. (Palacios et al. 2010) Some literature sources consider service developer and provider as one stakeholder while other sources separate them. The role of service client is also viewed differently - others see it as an active entity in testing while others see it just as a user who does not participate in SOA testing at all. Some sources treat the service registry as an active participant in SOA testing.

5.3.1 Service developer

The service developer owns the implementation and is able to perform white-box testing. The developer aims to create a highly reliable service and tries to detect as much failures as possible. The developer should assess the non-functional properties of the service and its possibility to handle the exceptions. The developer does not have to pay for testing his own service. Non-functional testing done by the developer may not be realistic because it is not done in the same infrastructure where the service will eventually be running. Moreover, the test cases may not reflect a real usage scenario. (Lucia & Ferrucci 2009)

5.3.2 Service provider

The service provider has to test the service to make sure that it satisfies the SLA requirements with the customer. The usage of white box testing techniques may not be possible at this point, because the provider is not necessarily the owner of the service. Non-functional testing done by provider does not necessarily reflect the consumer infrastructure, network configuration and actual load. (Lucia & Ferrucci 2009)

5.3.3 Service integrator

The service integrator needs to test the composition of services to ensure that any new service bound to the composition satisfies the functional and non-functional requirements that were set at the design time. Integrator faces testing challenges because of runtime discovery and ultra-late binding of services. Bound service may be one of many possible candidates. Moreover, integrator cannot control the service in use, and the changes may be made to the service. Service invocations are required for testing, and they may result in costs for the integrator and wasted resources for the provider. (Lucia & Ferrucci 2009)

5.3.4 Certifier

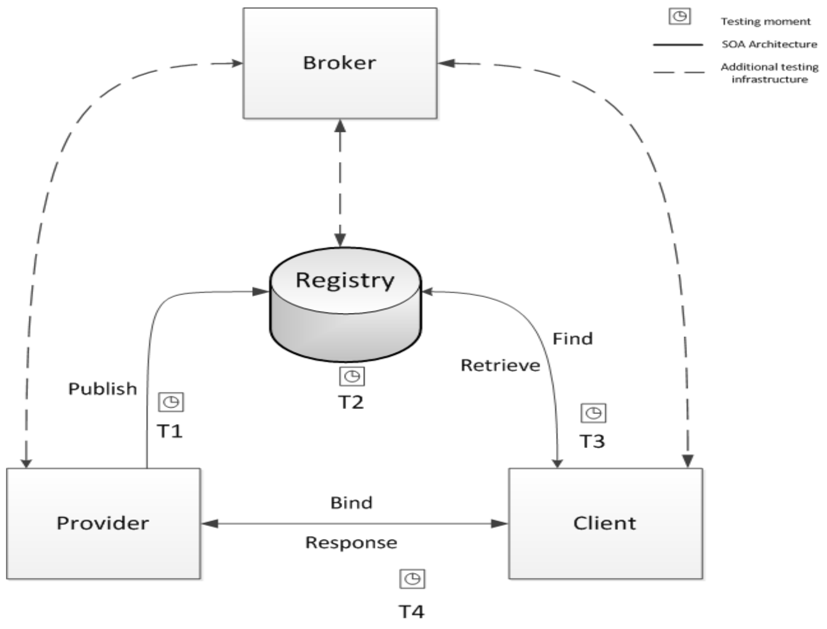
The third-party certifier (broker) is an independent entity that can test the service on behalf of someone else. The Service provider can rely on the certifier in order to guarantee the service reliability and performance to service users. The certifier cannot test a service within any specific composition as the integrator does. Neither can certifier perform testing by using the same network configuration as the service integrator. A provider can save in testing costs by using a third-party certifier in testing. (Lucia & Ferrucci 2009)

5.3.5 End user

The role of the end user varies in different sources. In some cases, a user might also want to periodically run automatic tests to ensure that service he is using, works as desired. (Baresi & Nitto 2007b) At some cases, the end user does not perform any service testing but only determines if the SOA application works as desired. (Lucia & Ferrucci 2009)

5.3.6 Example scenario

Palacios et al. (2010) identify four key entities that are involved in SOA testing. These roles are service provider, service registry, broker and client. The service provider is responsible for providing the service. The registry stores all the services and allows clients to browse and find services. The broker is an independent and objective entity that participates in testing and can increase the reliability of services. The client is responsible for the utilization of services. The authors also define four periods of time when SOA testing happens. Different entities participate in testing in different periods of time. Picture 12 represents this scenario. (Palacios et al. 2010)



Picture 12: Roles and testing periods of time in SOA testing (Palacios et al. 2010)

The first phase is the registration of the service as provider publishes it to the registry (T1). The provider, registry and broker may participate in testing, but at this point of time, the client cannot participate. When services are in the registry, the registry and the broker are responsible for testing (T2). While in the registry, services are still deployed in the provider's infrastructure and some changes to services may occur. The registry and the client may not be aware of these changes. The UDDI standard does not include any support for testing, for example, storing QoS information to the registry. This deficiency has been identified and various extensions to include more testing functionality to the service registry have been suggested. Some of the literature does not consider the service registry as an active entity in testing. The reason for this is the fact that the registry is not necessary needed if the amount of services is reasonably low. QoS information stored in registry can help the client in decision making when the client is running possible tests just before binding (T3). At this point of time, the provider does not participate in testing because it does not know who is going to use the services. The client may run some tests to retrieve the information that allows him to select the most suitable service for his needs. When services are being executed, the client is responsible of monitoring the execution of services (T4). (Palacios et al. 2010)

5.3.7 Requirements for testers

In SOA, application logic is operating within numerous technologies, residing outside the department or even outside the company. Composite SOA applications are no longer monolithic. Testing an application developed by single unified group, as a single project, for a single application server and delivered through a standardized browser interface is no longer a reality. A SOA testing team must be cross-functional and it must consist of both technology and business domain experts. Business services can be developed by different groups of people in different countries and different companies. These groups can have completely different quality thinking and groups may have little interaction between each other. Different groups of people have their own development practices and standards. They may also have different levels of maturity in SOA standards' utilizations, which will generate additional challenges for testing. A framework of collaboration and communication need to be established between the organizations and stakeholders participating in SOA testing. (IBM 2007; Harris 2007)

The testing team, that can consists of testers, performance engineers and developers, needs to understand SOA concepts. The team is also required to understand business tasks and processes to be tested as well as the web service technologies and tools. Understanding of network security is an implicit requirement. Business analysts need to give their input for creating test strategies and plans. An effective testing approach prioritized by business risks is needed at the business process level. (IBM 2007; Harris 2007)

5.4 Testing levels in SOA

Testing single web services only is not sufficient. All the levels of testing need to be considered with both functional and non-functional testing (including performance, security, interoperability, backward compatibility and compliance testing). In functional testing, usually a black-box approach is taken to ensure that a single component or a composition is operating according to specifications. High-level technical design definitions and business requirements can be the main inputs to design functional test cases. (Harris 2007) Harris (2007) suggests dividing the service oriented architecture into separate domains such as service, security and governance. Each domain has its own testing methods and approaches and each domain should be managed separately. SOA testing can be divided into the following levels:

- SOA governance
- Unit testing
    - Service component-level testing
    - Service-level testing
- Integration testing (process and orchestration-level testing)
- Regression testing
- Acceptance testing
- Non-functional testing
    - Security
    - Performance
    - Other non-functional testing types

5.4.1 SOA Governance

SOA governance is a complex set of rules, policies, practices and responsibilities by which a SOA system is controlled and administrated. (Bartolini et al. 2009) The governance includes standards and policies that govern the design, build and implementation of SOA solution and also the policies that need to be realized during runtime. The policies include QoS policies on performance, security and transactions and also regulatory, business, audit and infrastructure policies. The objective of SOA governance is to ensure that all the services conform to the standards, policies and objectives of an organization during the entire life cycle of the services. The governance supports the development, implementation and management of SOA. It is needed to ensure that SOA achieves its goals. (Harris 2007)

In order to establish a quality assurance framework and to improve the testability to SOA composite applications, SOA governance needs to be established among all the stakeholders in a SOA project. (Bartolini et al. 2009) The presence of SOA governance needs to be tested as well. Quality controls and reviews must be implemented during the entire life cycle of SOA implementation. SOA governance testing is not a separate phase in testing process. Instead, it must be constantly executed by reviews, monitoring and different test scenarios. (Harris 2007)

5.4.2 Unit testing

Unit (or service) testing can be divided into service component-level testing and service-level testing. A service can consist of many components that should be tested separately before integrating them into a service. Unit testing is usually done by developers in order to ensure the correct functionality and behavior of a service. Formal peer reviews can be used at this point to ensure that service code complies with organizational standards and policies.

Potential performance and security issues may also be detected. (Harris 2007) Because of different service abstractions of service layers, unit testing may likewise include testing of service compositions. This may include the testing of processes defined by using WS-BPEL. A developer can use white-box test strategies if he has access to the WS-BPEL process. If the process is tested from outside, only black-box methods may be used. Frameworks and approaches for WS-BPEL unit testing have been developed. Observability, test data generation, specification of complex IO (input output) types and the costs and side effects of testing are the major differences and challenges in SOA unit testing if compared to traditional unit testing. Various approaches to generate test cases from WSDL documents exist and unit testing is supported by many SOA testing tools. (Lucia & Ferrucci 2009; Ribarov et al. 2007)

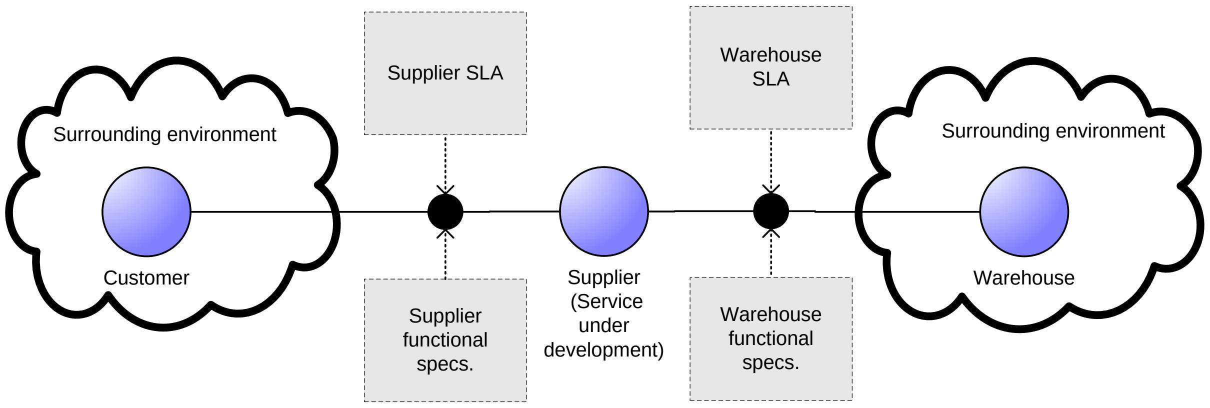
Service-level unit testing is a critical level in SOA testing. Service reusability requires that before services are integrated, they must have a sufficient quality. It is essential to develop and test the interface of a web service extensively, because composition application using that service may break because of the changes made to the service interface. The testing and quality activities recommended by this level include reviews as well as functional, performance and security regression suites to be executed against the service. Testing tools are needed in these activities. One challenge for the testing team is the selection of proper testing tool for unit testing. (Ribarov et al. 2007) An end-to-end approach is required for testing web services. This includes service integration points as well as the connected systems that can be services, web-applications, security gateways, legacy systems and back-end systems. (Suzuki et al. 2008)

5.4.3 Integration testing

Integration testing needs to be performed, because SOA applications comprise of dynamic compositions of autonomous services that are distributed over the network and developed and hosted by different companies. (Lucia & Ferrucci 2009) Even if each service is tested well individually, the quality of the service composition needs to be verified with integration testing. (Ribarov et al. 2007) Integration level testing focuses on service interfaces and its goal is to ensure the interface behavior and information sharing between services. All the services in integration testing must comply with interface definitions regulated by standards. (Harris 2007) WS-Interoperability is a specification that suggests best practices to increase service interoperability. Dynamic binding is the key challenge in service integration testing in SOA: a service may suddenly become unavailable or a new service with better QoS may appear. (Lucia & Ferrucci 2009) Automated monitoring mechanisms can be used at run-time to detect integration problems. The integration testing strategy should be developed in the beginning of a SOA project. (Ribarov et al. 2007)

When developing a new service to the composition, the service’s interaction with other services must be tested validating that it obeys the functional contracts and also to evaluate the level of quality that is produced. The quality of the service that is under development is affected by the quality levels of other services that are invoked in a composition. Not all the services that are part of the composition are under the full control of a single organization or a single stakeholder. Other services in a composition may not be implemented or even if they were, they could not be accessed for testing because their usage may cause unwanted side effects, like utilization fees and database modifications. (Lucia & Ferrucci 2009) Service clients are interested about service quality, functionality and performance. Service provider must cope with client requirements and limited control over services that need to be considered in testing. (Suzuki et al. 2008) Integration testing can be also regarded as process and orchestration-level testing, where complete service orchestrations are tested against the business requirements. (Harris 2007)

One challenge in integration testing is possible missing of services in the moment of testing. In this case, the missing parts may need to be simulated by generating stubs with testing tools. (Ribarov et al. 2007) The lack of information about the integrated components can make the generation of stubs more difficult. (Lucia & Ferrucci 2009) PUPPET is a tool that generates stubs for the external services, so that the service under development can be tested. In order to this, the functional contract of the service that is represented as a state machine is needed. And extra-functional contract that the quality of service contract represented as an SLA document is needed as well. (Suzuki et al. 2008)


Supplier SLA
Warehouse SLA
Surrounding environment
Surrounding environment
Customer
Supplier (Service under development)
Warehouse
Supplier functional specs.
Warehouse functional specs.

Picture 13: Service testing

In the picture 13, three services are owned and implemented by different stakeholders. A test bed is required for testing the compliance of the supplier service with the functional contracts of the warehouse service. In addition, reliable values for supplier service QoS must be derived by taking into account the QoS of the warehouse service. Both functional and non-functional behavior of the supplier depends on the behavior of the warehouse service that is going to be accessed at run-time. The real warehouse service cannot be used in testing, which means that stubs need to be generated for a warehouse service. Stubs for the warehouse service reproduce the functional behavior of the warehouse service and perform according to the QoS parameters that can be for example throughput, latency and reliability defined in warehouse services SLA. SLA describes the agreements that the service commits to accomplish when it is processing a request from a client. (Suzuki et al. 2008)

The stub is generated in three phases. First, the abstract definition of WSDL document is converted into a collection of Java classes. The stub is then extended with the code that is simulating non-functional behavior that is described in WS-Agreement document and mapped to parametric portions of Java code. Finally, the code that emulates the intended functionality is inserted to the stub. This requires that the functional specification of the service to be simulated in the test is available as a state machine. Functional specification defines the correct values that service accepts, the legal order of events and how the service answers to requests. (Suzuki et al. 2008)

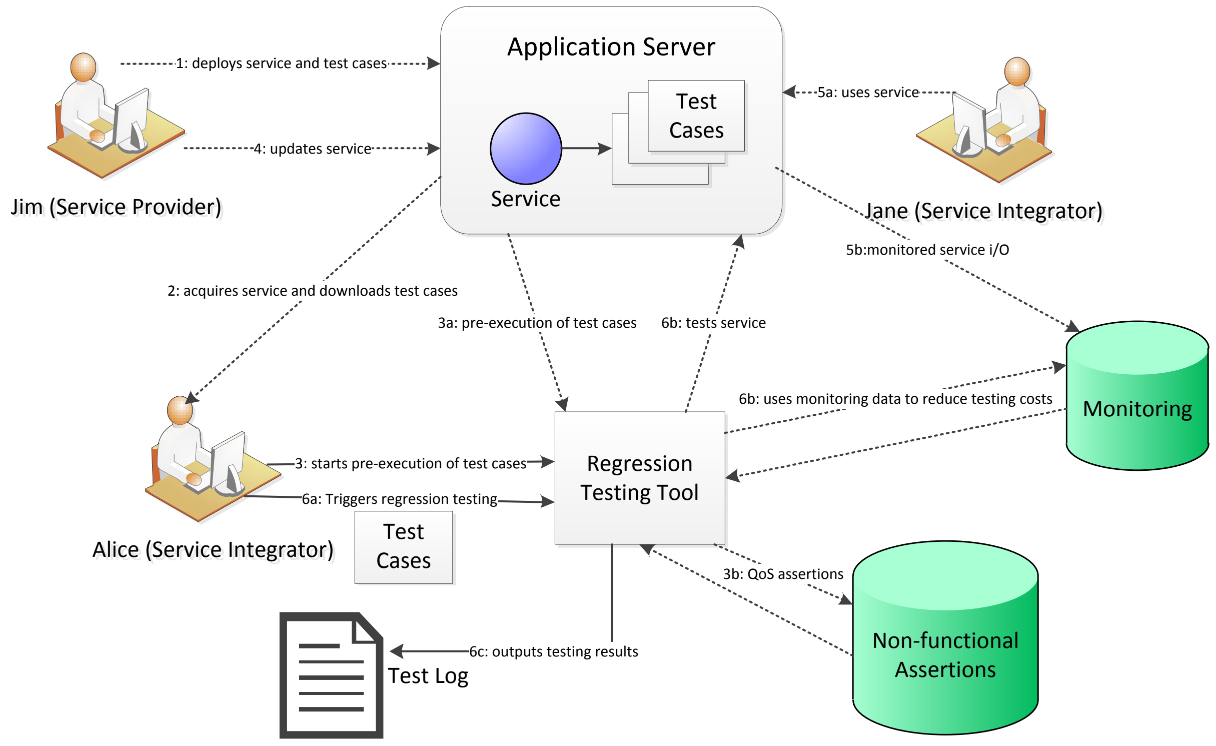
In order to facilitate the integration testing, Ribarov et. al. (2007) suggest an approach that aims to build so called test-enabled enterprise service bus that contains components acting as intermediates between the communicating services by reading and manipulating the messages. (Ribarov et al. 2007)

5.4.4 Regression testing

Regression testing is usually performed by service integrator. Based on certain functional and non-functional requirements, the service integrator chooses a service that is to be integrated to the service oriented system. The service needs to be tested periodically in order to ensure that it fulfills the desired functionality and QoS requirements. (Baresi & Nitto 2007b) Regression test suites must be built in order to automate the functional testing through the lifecycle of services. (Harris 2007)

When the service integrator integrates a service with the system, the integrator assumes that functional and non-functional characteristics are maintained when service is being used. However, this is not true. The service integrator is unable to control evolution and maintenance activities that are made to external services during their lifecycle. If a change does not require modification of service contract, it may be thought as a minor update and it is not communicated to integrators. If a change is made to a service, not only can the functional behavior change, but also QoS parameters might get altered. Eventually, the whole system using altered service may be affected. Some changes that do not affect to the behavior of the service may still have an effect on QoS. Modification to QoS of a single service may also affect to the QoS of the composition. This is a most notable difference between services and components based systems, where system can continue using older version of component in a case that changes are made to it. Stakeholders including provider/developer, system integrator, third party/certifier and even the user may participate in SOA regression testing. (Lucia & Ferrucci 2009; Baresi & Nitto 2007b)

Di Penta et. al. (2006) have developed an approach for SOA regression testing. They propose that service providers should publish test cases along with the service. The main idea of the approach is to reuse test suites that are made during early stages in development of a service, also in later testing because eventually, the service fulfills a part of the functionality of the whole system itself. Because test suites access a service's internal resources that are not visible from outside, it is necessary to transform test cases to proper form. Integrators could then use these test cases as complementary resources in testing. The authors have developed a tool called Testing Facet Generator that generates test cases from test suites made with JUnit and produces XML-encoded testing facet. The other tool, The Test Suite Runner allows a tester to download testing facet of the service, pre-execute the test suite and generate QoS assertions, execute test suite and produce the test log. It supports replay and capture operations. (Baresi & Nitto 2007b)


Application Server
Test Cases
Service
1: deploys service and test cases
4: updates service
Jim (Service Provider)
5a: uses service
Jane (Service Integrator)
5b:monitored service i/O
2: acquires service and downloads test cases
3a: pre-execution of test cases
6b: tests service
6b: uses monitoring data to reduce testing costs
Monitoring
3: starts pre-execution of test cases
6a: Triggers regression testing
Regression Testing Tool
Alice (Service Integrator)
Test Cases
3b: QoS assertions
Non-functional Assertions
6c: outputs testing results
Test Log

Picture 14: An SOA regression testing scenario (Baresi & Nitto 2007b)

In a picture 14, a service provider (Jim) deploys a service and a test suite for it. A service integrator (Alice) discovers the service, negotiates the SLA, downloads the test suite and complements it with additional test cases. Then she pre-executes the test suite and measures its non-functional behavior. An SLA is agreed with the provider and the test suite and QoS assertions are stored in a database. Alice continues using the service until Jim updates it. As a result, the response time of the service increases, from one to three seconds. A service integrator (Jane) uses the new service without any problems, because she may have different quality requirements than Alice. Jane's interactions with the service are monitored and stored into a database. Because of the changes made to the service, Alice wants to test the service. She can utilize monitoring data in order to reduce the number of service invocations that may harm Jim and cause extra costs. The system creates a test log that contains success and failures of functional test cases and QoS assertions. For example, the non-functional assertion that expects the response time to be less than two seconds will now fail. (Baresi & Nitto 2007b)

5.4.5 Non-functional testing

Quality of service needs to be monitored throughout the SOA project. Quality has to be understood in a way that sees how all the elements fit together and form a coherent business process. The quality of SOA implementation of an enterprise must be verified through test approach that takes into account the nature of SOA. Non-functional testing, that has a significant role in SOA testing process, can ensure the presence of quality attributes including performance, interoperability and vulnerability. (Lucia & Ferrucci 2009; Baresi & Nitto 2007b; Haines & Rothenberger 2010)

The importance of non-functional testing in SOA increases dramatically, because the important role of Service Level Agreement (SLA). With the SLA, service provider promises to customers that service realizes a certain level of QoS. Because of unexpected inputs from user or because unpredicted service load, promised QoS may not be fulfilled. Thus, services need to be robust and able to perform recovery actions. Because services are accessed over the Internet, they can be the target of attack against security. These are some of the reasons why appropriate plans for non-functional testing of services need to be established. (Lucia & Ferrucci 2009) Testing web services based SOA means that SOAP, XML and REST based messaging must be tested against the service endpoint to ensure the reliability, robustness and resilience of the service. Testing the service endpoint not only involves the functional testing, but also performance, interoperability and security testing. Numerous different SOA testing tools for testing of non-functional properties exist. (Macy 2009)

5.4.5.1 Performance testing

Performance testing becomes more important as systems integrate with other organizations' systems and as the services are highly reused. It is important to consider performance, load and stress test before the system integration phase. Many SOA testing tools support the performance testing of services and components. (Harris 2007) Performance testing means establishing a framework that can monitor the throughput and capacity statistics of the service when addressed with different inputs and different load variances. With performance testing, SLA rates are validated and potential architectural weaknesses, bottlenecks and performance dependencies are identified. (Macy 2009)

5.4.5.2 Interoperability testing

The design characteristics and the runtime adherence to standards and best practices of services are measured with interoperability testing. Finding potential interoperability issues as early as possible eases the integration level efforts significantly as the services are combined with trading partners' and clients' services that are possibly built with different technologies. Interoperability testing includes both design-time analysis of service characteristics (WSDL and WSDL-schema) and run-time testing of service's robustness in handling of message patterns that may include structures that are not expected. (Macy 2009; Macy 2010) In robustness testing, it is ensured that the service reacts properly when an unexpected condition happens. The error recovery code might very often remain untested. An approach for testing the exception code of the services that is, the code in the catch-block has been developed. (Lucia & Ferrucci 2009) Run-time interoperability behavior testing verifies that services can interoperate and are independent to platform, operating system and programming language. (Harris 2007)

5.4.5.3 Security testing

Security testing must be planned at the beginning of a SOA project and it must be considered throughout the project with an end-to-end approach. (Baresi 2007) End-to-end security includes mutual authentication, authorization to access resources, data integrity and confidentiality, integrity and confidentiality of messages, integrity of transactions and communications, audit records and mechanisms and distributed enforcement of security policy. (Suzuki et al. 2008)

The security standards for web services address to different security aspects, like message-level security and identity management. There are many different security threats against web services, like message alteration, man in the middle, forced claims, replay and denial of service. (Baresi 2007) SOA with web services introduces similar security challenges to web applications, including cross site scripting, buffer overflows and SQL injection. (Ribarov et al. 2007) SOAP protocol does not specify how to deal with security issues. It can be extended with WS-extensions, such as WSS (Web Service-Security, which secures the SOAP foundation layer with technologies such as XML signature, XML Encryption, XML Canonicalization and SSL (Secure Sockets Layer). WS-Security specifies how the header part of a SOAP message can carry security information. It is important to understand all the vulnerabilities and choose the right testing tools to detect faults in web services. (Baresi 2007) Security standards should be adopted at early phases in SOA development. (Ribarov et al. 2007)

There are different perspectives in security testing. From a threat perspective, security testing involves integrity and structure of messages with injection attacks at the parameter and data structure levels in order to assess the behavior and resilience of the service endpoint when faced with data values and message structures outside the expected format. From trust perspective, security testing involves PKI with encryption, signatures and identity tokens. Testing frameworks need to support the various emerging standards from W3C and OASIS in order to support the wide range of security message formats. The means are required to retrieve and utility X509 and Private Keys from a variety of sources including Windows Keystore, Java Keystore and SmartCards and PKCS12 (Public Key Cryptography Standards) files. (Baresi 2007) Security testing assesses the service with regard to vulnerability, data-leakage, data privacy and data integrity. WSDL schema can be used as a source to build security tests based on boundary conditions. Security specifications defined by W3C and OASIS can be utilized to create a security testing framework to test the level of data integrity, access control and data privacy on the service transactions and endpoint itself. (Macy 2009)

5.4.5.4 SLA testing

In a service centric system, there are different factors that can violate the factors specified in SLA. The factors that can cause violations include inputs, bindings between abstract and concrete service descriptions, network configurations and server load. In SLA testing, the goal is to find a situation in which the service cannot provide its functionality with a desired level of SLA. Before providing the SLA to the consumer, the provider must ensure that the SLA is not violated during service usage. A method has been proposed for SLA testing of atomic services and service compositions by using Genetic Algorithms (GA). Combinations of different inputs and bindings generated by GAs can be used to cause SLA violations. Service's QoS attributes including response time, throughput and reliability are monitored during test invocations. (Lucia & Ferrucci 2009)

5.4.5.5 Dependability Testing

In a business critical scenario, the generation of service inputs that cause security problems by allowing unauthorized access to sensible data, service crashes, unexpected and unhandled exceptions or behavior that cannot be observed by only looking at the service response. SOAP message perturbation is one approach that can be used to test dependability. It allows alternation of application's internal state without modifying the source code. Typical testing process involves the modification of request message, resending them to the web service and analyzing the possible changes in response message compared to the original message. (Lucia & Ferrucci 2009)

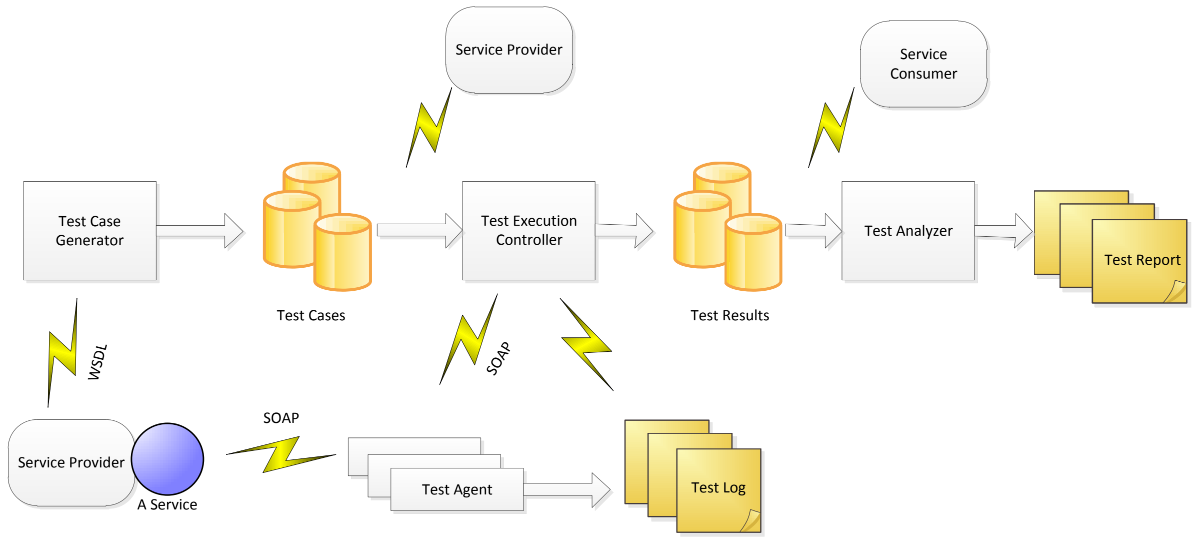
5.4.6 User acceptance testing

In user acceptance testing the SOA solution is tested so that it delivers the defined business requirements and has met the defined business acceptance criteria. This phase of testing targets only the key business scenarios of the solution. The needed quality and test coverage must be reached in earlier levels of testing. (Harris 2007)

5.5 SOA testing automation

Because of the dynamic features of web services and the need for periodically re-test the service compositions it is obvious that SOA testing needs to be automated. Test case generation, test execution and the collection and analysis of test results are tasks that can be automated. Test tools can analyze the specification (WSDL) of a web service and extract information needed for testing from it. This information includes service operation, data structures and operation semantics. Test cases should be documented by using XML in order to reuse test cases through evolution of services. (Xiaoying Bai 2005)

Bai et. al. (2005) represent a framework for distributed service testing. The framework can be used for testing at different levels, including individual operations of a single service, combination of operations of individual services, individual operations of composite services or combination of operations of composite services. The framework is presented in picture 15.



Picture 15: Overall architecture of the distributed service testing framework

First, the Test Case Generator accepts a WSDL of the service provided by the service provider. The generator can be extended to support other description languages, like WS-BPEL and OWL-S. The generator automatically generates the test cases and stores them into a central database. Testing is controlled by Test Execution Controller, which retrieves test cases from the database, allocates them to distributed test agents, monitors test runs and collects results from tests. Test Agents are proxies that perform remote testing on target series with specific usage profiles and test data. Agents are located in LAN or WAN area. The analyzer goes through the test results, evaluates the quality of services and produces test reports. All the stakeholders participating testing can access the databases with different privileges. (Xiaoying Bai 2005)

Reusability of test cases increases the efficiency of test automation. By creating a library of executable test steps that can be combined to build more complex test cases is a good practice. Additionally, a data-driven test approach, where the same test scenario is reused many times with different data sets, may increase the efficiency of test automation. (IBM 2007)

Testing tools are needed for test automation. Tools ease the creation and management of functional and non-functional tests. Test automation must be considered as a separate task with a dedicated approach including a detailed plan, with requirements, objectives and resource allocation. (IBM 2007) The test tool strategy in SOA project must consider the whole life cycle of SOA solution. (Macy 2010) Because the nature of SOA is so standards driven, it is suitable for testing tools. Many vendors have realized the need of SOA testing tools as the development and testing teams require more efficient and centralized management of web services. SOA testing tools must be natively integrated with existing quality assurance and testing infrastructure. (Harris 2007) Some notable SOA testing tools are Networks SOAPSonar, Parasoft SOATest, GreenHat GH Tester, Mindreef SOAPScope Server, Crosscheck iTKO Lisa, Matador Tech Tester and SOAP UI.

6 CLOUD COMPUTING AND TESTING

Different viewpoints exist about how cloud computing can be associated with software testing. No approved definitions for “cloud testing” or “testing in the cloud” exist. The cloud may offer a testing tool in SaaS model that can be utilized by testers. Also, any kind of testing that targets an application residing on third-party computing platform can be called cloud testing. (Riungu et al. 2010) Some companies may be interested in building their own cloud in order to collaborate with its partners. This cloud environment needs to be tested as well. Cloud testing can thus mean three things (listed below). This thesis focuses mainly at point 2, but also discusses about point 3.

1. Testing an online application
2. Utilizing cloud services in testing
3. Testing own cloud environment

As cloud computing is evolving towards everything as a service model (XaaS), Software testing is becoming a utility that can be offered as a service by third party testing companies. Software testing as a service (STaaS) is “a model of software testing used to test an application as a service provided to customers across the Internet”. This aspect of testing is not discussed further in this thesis. (Riungu et al. 2010)

6.1 Utilizing the cloud in testing

Due to limited resources, small and medium sized enterprises may be unable to setup a secure and scalable IT infrastructure rapidly. By migrating to cloud computing, these enterprises can focus on their core business rather than worrying about the investment and maintenance of their IT infrastructure. Cloud computing has its challenges that must be taken into account. Enterprises should consider issues like security, reliability and manageability when creating a test strategy when considering cloud computing. The traditional testing that is performed with on premises applications that are possibly single-tenant applications may not be enough when testing a multi-tenant application in cloud environment. Before moving applications to the cloud, understanding of cloud environment and all the risks and challenges associated with it is needed. (AppLabs 2009)

6.1.1 Test strategy for application in the cloud

Business requirements and inherited risks of cloud computing need to be identified before moving applications to a cloud environment. More skills are required from testers including the knowledge about the nature of cloud computing. A tester should have a general idea about cloud application architecture. Also the knowledge about testing tools is required: what kinds of tools are available, what are their strengths and weaknesses, and which of them are suitable for testing different types of cloud applications. Tester must also cope with constantly changing nature of cloud computing. (AppLabs 2007)

A test strategy that involves identification of various types of testing to be performed as well as selection of cloud test environment needs to be documented. Product risks should be based on the business characteristics of cloud computing and identification of the product risks may give some inputs to determine what type of testing needs to be performed. Cloud computing introduces some quality risks, that are reliability, flexibility, multi-tenancy, self-healing, SLA based pricing and location independence. The inherited risks of cloud computing present potential threats to applications. These threats are data governance, data security, virtualization security, reliability, monitoring and manageability. Understanding of cloud models (SaaS, PaaS, IaaS) is also required when deciding testing types to be adopted in cloud testing. A test strategy should also include the choosing of a test environment whether or not a cloud or on-premises environment will be utilized in testing. Some issues that are specific to cloud environment need to be considered in testing. Support for multiple browsers, user session management and restricting the data access in multi-tenant environment are examples of these issues. (AppLabs 2009)

Testing Type	**Purpose**
System Integration and user acceptance testing	Ensuring that the cloud solution satisfies the functional requirements
Interoperability and compatibility testing	Ensuring that migration to alternative cloud service provider works with no problems
Performance and load testing	Ensuring that the solution meets the business requirements specific to cloud computing
Stress and recovery testing	Ensuring data recovery from application crashes, and hardware failures in cloud environment
Security testing	Ensuring that application and data security requirements are met

Table 2: Testing types for cloud application

When considering using cloud solutions, a company must evaluate and test that solution to ensure that it fulfills the requirements. Table 2 lists the types of testing that can be done to verify and validate the cloud solution. Static testing activities should be conducted to verify and validate the business requirements by establishing reviews and workshops. Dedicated testing tools and methods are needed for load, stress and performance testing of the cloud solution to ensure that promised scalability is met with variable workloads. Security testing is required to ensure that critical data is stored and transported safely. Penetration testing techniques can be done to verify the security mechanisms of the cloud solution. System and integration testing are to be performed before deploying application to the cloud. Acceptance testing that uses business requirements will prove that a cloud solution meets the needs. (AppLabs 2007)

Several major technology vendors such as HP, Intel and Yahoo are collaborating to create cloud “test beds” that allow users to test their cloud deployments at Internet scale. Test beds help the users to understand how their cloud deployment actually behaves within the cloud. HP and IBM are offering test tools for non-functional and automated testing in cloud environment. HP's Quick Test Pro and IBM's Rational Robot can be used to perform automated testing tasks like regression tests. Tools such as Load Runner and Rational Performance are also well suited for testing the cloud solutions. (AppLabs 2007)

6.1.2 Example: comparing different cloud providers

There are many cloud service providers currently in the market and the number of them is increasing. From the service consumer point of view, it becomes crucial to compare different offerings to select the right provider among the alternatives that best satisfy the needs. (SPIRENT 2010a)

In a scenario of comparing different cloud providers, infrastructures of six different cloud providers were utilized. Three-tier architecture, consisting of front end web server accepting HTTPS (Hypertext Transfer Protocol Secure) connections, the application server and an SQL database were set in each cloud environment. Three web pages were made: one that simply returned a string, another with a search field which queried the database and the third one that triggered a CPU intensive task of rendering 3D (Three Dimensional) graphics with POV-Ray (Persistence of Vision Ray tracer) software. HTTPS scalability of each provider was measured. 410 transactions per each page were simulated using Spirent Test Center running Avalanche software connected to the Internet. The percentage of successful HTTPS transactions varied between different cloud providers some of them resulting in a significant amount of failed transactions. Without detailed knowledge about the cloud resources, a user may encounter problems if he plans to host a web service in the cloud for hundreds of users. Scalability test pointed out how many users the cloud can scale up to with HTTPS sessions. In order to measure the CPU performance of the clouds, a lower amount of transactions were simulated to call a CPU intensive task. The scalability test was run with and without the POV-Ray tasks in parallel. HTTPS response times were measured to identify, how long does it take for the server to return the SQL query while processing the POV-Ray task in parallel. The results showed that the most scalable clouds produced the longest response time while processing CPU-intensive task. The results indicated that heavy CPU load affects the response times of all users. Results also indicated that public clouds each have their strengths and weaknesses and cloud may have different configurations and prioritizations that the user may be unaware of. These are the challenges when choosing a cloud provider among many alternatives. (SPIRENT 2010a)

6.1.3 The cloud as a testing environment

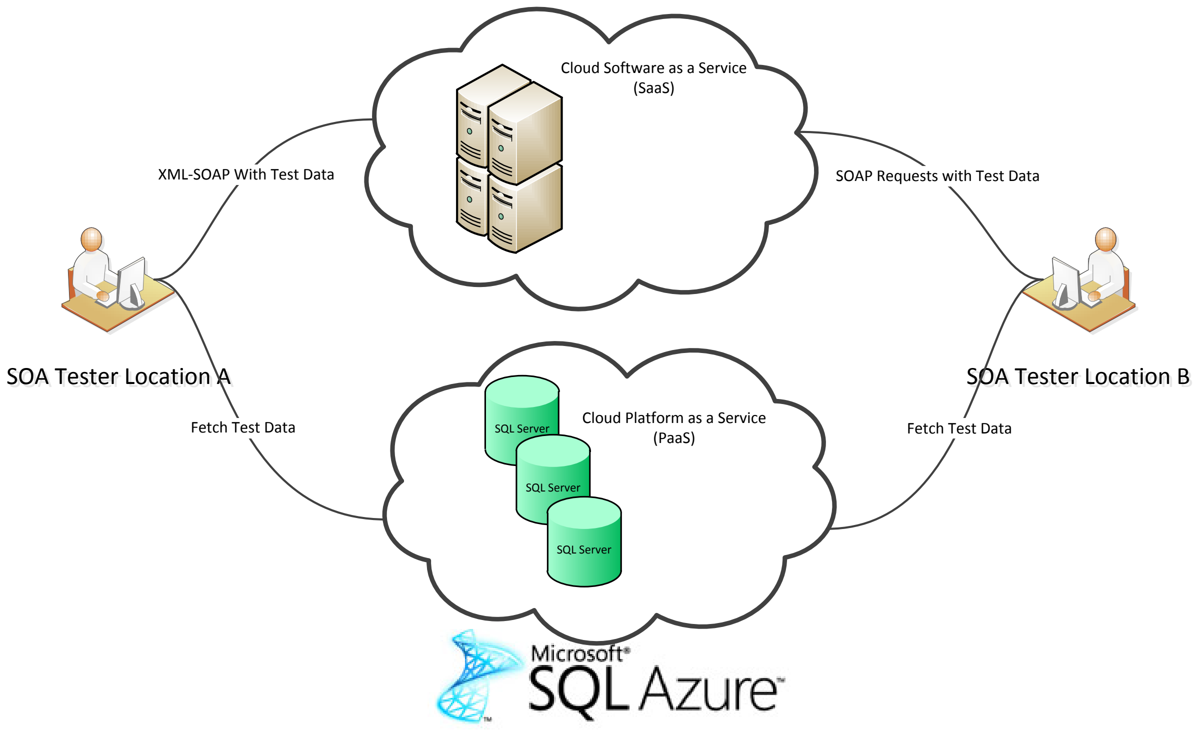
A testing environment must replicate the environment where software will eventually be deployed on. Building this kind of environment may require enormous investments. An organization that has its own test environment may have several projects going on, all requiring a time slice of the environment. This may cause some projects to remain in queue thus making projects longer and decreasing the time to the market. Building a bigger environment is expensive, and after peak times, the usage of the test environment may diminish significantly. (Torode 2009)

The cloud can be utilized as a virtual test lab for application development and quality assurance. In a recent study, software buyers of different corporations were asked about most tempting uses of public clouds. Application testing was listed as one of the most appealing areas of cloud computing utilization. As projects no longer need to wait for their turn to be tested in on-premises environment, they can get to market faster. However, cloud as a testing environment may not be suitable for all companies. The configurations, servers, and storage supported by the cloud vendor may not be able to simulate the customer's real production environment. If applications run on specific physical servers, specific virtualization technology, specific networking and bandwidth, or the used technology is not supported at all, utilizing cloud computing as a test environment may not be possible. (Torode 2009) With cloud computing, certain compromises are no longer needed in performance testing. Testers can access unlimited resources and need to pay only for those they actually consume.

Applications should be tested in environments that are as close to real product environments as possible. When deploying applications to the platforms offered as a service, such as Windows Azure, on premise environment can be used for testing, but final testing should be done in a cloud environment in order to get realistic results. This can be done by having two different subscriptions of Windows Azure, one completely for testing and the other for real production. These environments are identical environments and the application will run in the same way on both of them. Also the application deployment will work similarly in both environments, what is not guaranteed in on-premises environments. Access to these environments can be set so that only authorized persons can access the environments. Because two environments are billed separately, it is also easy to monitor the costs spend on testing and development, which means that product budged planning easier. (MSDN 2011)

6.1.4 Example: Using Azure and SOAPSonar in testing

Cloud computing enables developers and testers to use cloud based relational database management systems (RDBMS). MS SQL Azure is an example of this. It enables better collaboration among testers and developers by sharing test data across enterprise boundaries and geographies. Downloading, installing, patching and managing the database that is used to store test data, becomes simpler as cloud-based RDBMS is being used. New instance of database server can be launched within minutes, which can reduce resource conflicts and scheduling challenges. By eliminating the need for on-premise hardware and software, costs can be saved. (Yunus 2010b)



Picture 16: SQL Azure Setup for collaborative testing (Using SQL Azure for SOA Quality Testing)

Picture 16 presents an example scenario of collaborative testing. Example scenario includes three components: SQL Azure, target web service and SOAPSonar testing tool. SQL Azure is used to maintain and share test data that is consumed by testers that are located around the globe. First, the SQL Azure account was acquired and database instance created by using Azure Web Interface. Then, the firewall rules were configured to allow remote access to create tables and insert data to the database. SQL Server Management Studio was used for creation of the database. SOAPSonar testing tool was then installed, and the WSDL of the service to be tested was loaded in SOAPSonar. A connection to SQL Azure was then created from SOAPSonar by adding a connection string of the database. Now, the data from the database can be used for test automation by adding an automated data source in SOAPSonar. For all entries in the database table, SOAPSonar sends a SOAP request to the web service under test. Both requests and responses can be viewed in editor. (Yunus 2010b)

6.2 Testing the cloud itself

Increased bandwidths and advances in network technologies are causing paradigm shift in computer and network architectures. Datacenters are able to provide different types of services than before. Companies which have invested heavily on their datacenters are becoming more interested to realize the benefits of cloud computing by building private clouds. (SPIRENT 2010a) Datacenters are not only larger and faster than ever, but also more complex. Validation and performance assessment of the data centers comprising of new technologies are in important role. (SPIRENT 2009)

The cloud sets significant requirements for the network, which now have to deliver better and faster services. Companies that are building private clouds need to assess the performance, availability, security and scalability of the cloud. Cloud-optimized test and measurement solutions are needed to assess these attributes. (SPIRENT 2010b) Below is the list for the tasks that cloud test solutions need to perform:

- Holistically validating the performance of all elements of the data center and the cloud computing environment
- Benchmarking the performance of new virtual switches, firewalls, load balancers and other virtual devices
- Quickly determining which physical or virtual components are impacting performance
- Measuring the impact of application reliability on dynamic resource scheduling with live migration of virtual servers
- Testing the vulnerability of virtualized security devices by emulating real world attacks and threats (SPIRENT 2010b)

Several RFCs (Request for Comments) has been published by IETEF (Internet Engineering Task Force) for data center device testing, including RFCs 1242 and 2544 for router testing, RFCs 2285 and 2889 for Ethernet switch testing, RFCs 2432 and 3918 for IP multicast testing, RFCs 2647 and 3511 for firewall performance measurement testing. (SPIRENT 2009) When moving towards a hybrid model of cloud computing where public clouds are integrated with private datacenter, several concerns are introduced:

- Data center benchmarking: how performance of interconnection devices of hybrid datacenter can be measured?
- Firewall performance and scalability: as the number of user sessions increase, how does firewall affect the performance?
- Virtual security: how it can be verified that virtual machines are secure and block connectivity to other virtual machines?
- Availability: Is the provider honoring the SLA? Do advanced features such as VM migration affect the availability to customers of the data center?
- WAN optimization: Do WAN optimizers really increase the performance and provide ROI?

Virtualization is the key technology of private clouds. It causes challenges to network design by introducing concepts of virtual switches and links. Architecting, securing and testing virtual networks must be addressed. Spirent has developed test equipment for virtual environments consisting pieces of Spirent TestCenter virtual (STCv) and Avalanche virtual (AVv) that can address the testing challenges of virtual environments. Challenges for testing virtualized network are: how can a test instrument attach to a virtual network device, can test instruments on virtual machines be trusted, do virtual and physical switches offer comparable performance and does a virtual switch support the same protocols and functions as a physical switch. The unpredictable nature of virtualized resources remains as a challenge and are of future research. For example, how does one user's virtual machine affect the performance of other VMs residing on the same physical machine. (SPIRENT 2010a)

6.3 The SaaS model and testing

There are different ways to develop a SaaS application. Many different types of software components and application frameworks can be utilized in SaaS application development. Several issues have to be taken into account, for example, when making architectural decisions. A separation between first and second generation of SaaS applications can be made. The first generation SaaS providers have transferred existing on-premise applications to web-based application. The next generation of SaaS applications will evolve to be more service oriented. It is also predicted that more technical features such as embedded business processes, web 2.0 style user interaction with Rich Internet Applications (RIA) and embedded in-process real time analytics will be implemented to SaaS applications in future. (Oracle 2008)

A set of technical requirements need to be taken into account when developing SaaS applications. Service level management tools are needed to have the ability to model real world services from end-user's perspective. Tools must be able to monitor key service indicators for availability, performance, usage, and service level compliance and provide centralized reporting and viewing of service levels from real time and historical perspective. Customers of SaaS solutions are concerned about security and privacy issues. Sensitive customer data must be kept safe in multitenant environment. Powerful enough monitoring tools must be available to track access and usage. A Service oriented architecture based on standards enables a provider to composite SaaS applications from existing reusable services rather than building monolithic applications from scratch. SOA also allows greater interoperability within existing services located within and outside the firewall. Because customers have different processes and business requirements, the on-demand applications must allow easy customization and extension of user interfaces, data and business processes. The underlying infrastructure and software architecture must allow patches and upgrades so that there is a minimal possible downtime in service. This is a major challenge because of the SaaS application can be used by multiple customers across different countries, continents and time zones. The underlying platform must be scalable so that more tenants and more users per tenant can be added without causing problems in service levels. Traditionally, for time and mission critical systems customers have required high scalability, availability and reliability. The importance of these quality attributes became even more relevant in SaaS applications, even the applications are not mission critical. Downtime of SaaS application can affect hundreds or even thousands of customers. (Oracle 2008)

Multitenant architecture and shared services increases the complexity of tasks like restoring data, for example, how to restore data for one single customer without the impact on other customers. The principle of multi-tenancy is not universally accepted within the software industry and not all the literature agrees that it is a requirement for SaaS. According to Bezmer et. al (2010) “*A multi-tenant application lets customers (tenants) share the same hardware resources, by offering them one shared application and database instance, while allowing them to configure the application to fit their needsas if it runs on a dedicated environment.”* (Bezemer & Zaidman 2010) Multi-tenancy can offer several benefits for a SaaS vendor. More clients can be supported with fever hardware, application updates and security patches can be done quicker and simpler, and software architecture is in general more robust and manageable. Multi-tenancy is not the only architecture of SaaS application. Instead, architectures vary from hosted applications with multiple instances to multi-tenant shared services model. Customers’ needs concerning security, availability and customizability have impact on the choice of proper tenancy model. (Oracle 2008)

With SaaS model, an ISV can get a better customer insight; they can monitor how the application is actually used and get valuable information about usage patterns and usability issues, which is not always possible with on-premise applications. The usage information can later be used to improve the software. New applications can be brought to market more quickly because of the shorter product lifecycle. Upgrades and fixes are also easier to make as the solution can be tested with real-life users. SaaS may cut down R&D (Research and Development) costs dramatically, because vendors are no longer obliged to develop and maintain multiple different versions of software product or develop software for different platforms. Because the customer is interested only in the service, SaaS providers can determine the implementation details and technologies to be used and also optimize the software for single environment. This may reduce the costs caused by porting and testing applications on multiple different platforms. Customer support costs as well as the product distribution costs may also decrease. However, some vendors' consulting services may no longer be needed with new delivery model. (Dubey & Wagle 2007; Oracle 2008)

A traditional software company is only responsible for developing and testing the software. After these phases, the product is shipped to the customer, who installs it in his premises and takes care of updates and maintenance. In SaaS model, the ISV is additionally responsible for everyday operations and constant maintenance that includes capacity planning, patch management and upgrades. The provider must also meet its SLAs for uptime and performance. Hosting and managing the application environment for SaaS applications may introduce new costs and challenges for vendors. (Oracle 2008)

7 MASTO INTERVIEW RESULTS

In fourth interview round of MASTO-project, representatives of ten IT-companies were interviewed. The data from these interviews were analyzed with Atlas.ti software without any formal research method being used. The main emphasis of the interviews was on the testing process and ISO/IEC (International Organization for Standardization, International Electro-technical Commission) 29119 testing standard. From this thesis point of view, the most interesting part of the interview was the section, which discussed about "testing process affected by new technologies". In this section, participants were asked about how concepts such as software architecture, SaaS, cloud computing, open source testing tools and crowd sourcing would affect software testing. It was queried how organization units' current software architecture or delivery model affects software testing, what problems their current architecture or delivery model may cause on software testing and how the problems (if there are any) would be solved. It was also questioned, if the software architecture or software delivery model has changed during the last years. The plans for the utilization of cloud computing and how it would possibly affect software testing was queried as well. It was also asked, if the open source tools are used and what benefits crowdsourcing would offer to a company. The roles of the participants were following:

Organization unit	**Roles of the participants**
Case 1	Head of unit of software engineering and hardware design
Case 2	Director of testing and methodologies
Case 3	Software manager
Case 4	Unit leader for tools performance testing unit
Case 5	Leader of the mainline testing
Case 6	Head of the software development
Case 7	Representatives of 2 different departments: testing manager and responsible of testing and release unit
Case 8	Responsible for development and software production
Case 9	Project manager
Case 10	CIO

Table 3: The roles of the participants

7.1 Software Architecture

Organizations had many different types of architectures. There appeared to be great amount of misunderstanding regarding what a software architecture is and what a delivery model is. These terms mixed with cloud computing caused lot of confusion. For most people, software architecture means an architectural style, like client-server, object oriented, distributed, etc. Some participants viewed SaaS as software architecture (which is definitely not). A SaaS application can have many different architectural styles. The cloud has its own architecture, but cloud computing is not a software architecture, but a way to get or deliver the IT-resources. The table below consists of architectures that the participants mentioned best to represent their current solutions.

Organization unit	**Software architecture**	**Changes in architecture**
Case 1	Consultation company, customers have many different	
Case 2	Several architectures because many different products	
Case 3	Web-based: web-databases and web-servers. Client (javascript) with some application logic. Has not changed	Has not changed
Case 4	Consultation company, customers have many different	
Case 5	Very data base oriented. Has not changed	Has not changed
Case 6	SOA, that is extending towards cloud computing	Constantly changing
Case 7	Database centric client-server	Has not changed. The other is changing because of SaaS.
Case 8	Quite a monolithic, desktop application	Has not changed.
Case 9	Database oriented, old style (process oriented, no object orientation) does not follow modern trends of SW development and planning.	Has not changed. At the end of life cycle.
Case 10	Depends on customer	

Table 4: Participants' description of their current software architecture and the amount of change addressed to it

Two of the organizations were consultation companies that do not produce any software by themselves. These consultation companies had many different clients that each had own architectures and delivery models. An organization unit like Case 2 has many different products with different architectures. Case 10 also pointed out that the software architecture depends on the customer.

Two extremities were identified concerning architectural maturity. Case 6 said its architecture to be service-oriented, and the organization unit has already considered cloud computing as a way to get certain IT-resources. The organization is utilizing Microsoft tools which in this case might lower the threshold of moving into cloud computing with Azure platform. Case 9 has an application that is at the very end of its life cycle and it does not follow any modern paradigms of software design, like object- or service-orientation. Because of the architecture, the application is very difficult to test with testing tools. The interviewee said that if they are going to build a new application, they must learn from mistakes and put a great emphasis on testability when designing the new software architecture.

> *"Well, software architecture has, or it is changing constantly. We are developing it more or less, it has been first for a service oriented model, if you talk about the architecture right now, now we are going to, more or less to cloud computing and that (so the) software platform and the architecture is changing. The model, software delivery model hasn't been changing that much, at least for now, but cloud computing obviously brings something to that as well." -Head of the software development, -Case 6*

> *"The architecture from one perspective is database oriented. But on the other perspective, it's good old process oriented software. No simultaneous or no object oriented programming … Because we have a variety of problems, because of architecture. It doesn't fit into modern server technologies so well. It doesn't take in use all the capabilities that modern servers and databases offer. So it's just a wiser choice to sell this so long as we can, and try to build a new software from scratch,*

using the lessons learnt from this, but still from scratch. Taking into account every possible architecture choice." –Case 9

Software architecture itself has a high impact on software testing. Case 5 said that because of their data base-oriented software architecture, the huge amount of database knowledge is required from testers. Testing would change if there would be major changes in the software architecture. Case 3 pointed out the nature of their software product that requires testing in different environments. Because of the client-server architectural pattern, software must be tested in wide range of browsers. This creates some challenges for testing.

"I would say, well, there are tons of different environments you have to fulfil, tens of browsers, different versions, different operating systems. And when one tier is running environment, you can't test or be quite certain that it runs. It's quite, well, I think only thing you can do is build the software on good and reliable building blocks, so you just believe that it runs all the environments customers use. So, this browser tier is one thing that really affects testing. –case 3

Case 9 pointed out that testing in general is very difficult because of the software architecture:

"And I think it affects so that the most, usually the most recent advances in testing are quite irrelevant, because if you take for example some software as service, so that we could use some Internet service for testing our program, that could be almost impossible, just because our software architecture doesn't follow the modern guidelines in software development or software design. And also, our software is database oriented, very heavily database oriented, and the database architecture that is used, is, to say kindly, not the, any good practice architecture.

I: I see. And how does it affect testing, anyway?

R: ..It affects so that, testing is hard…"

Changes in software architecture are laborious to make. If testability of the architecture is not agile and manageable, the architectural changes are very likely avoided to make. Sometimes the changes to the architecture might mean that the whole software must be rebuilt from scratch. Participants were asked if their software architecture has changed lately. Nearly all answered that architecture has remained the same for a long time. Only one participant reported that their architecture is changing constantly.

> *"We have plans to change, and we have been changing it. And the plans that we have to change the architecture, it also, there's one important part of the changing the architecture is to make it easier to test, I mean as automated testing. Those are the changes we have been doing and we are continuing to do, to be able to automate as much as possible." –Case 6*

Other representative of Case 7 sees that their architecture has been good and there are no needs to change it. Another representative of Case 7 said that there will be changes in architecture in future because of the SaaS model. Testing ought to be changed as well.

> *"We have at the moment, or we are going through our software architecture and there is some changes planned there. And we also have changed our, technically our licensing system during couple of last years, and that has affected somehow our customers, and also, we get requi-, or they have started to ask kind of.. well, I dunno if that is pure SAS type of solution for that licensing system, but some kind of hosting of that licensing server here and giving them as a ser-, that as a service there. So there is some changes going on in our side in that sense. And definitely that affects our testing work somehow. And we need to adapt to those changes also in testing side. –Case 7"*

Case 5 also gave some hints about architecture being changed because of e-services being taken into use:

"Yes. It will be changed. Yes, and really we are taking some new services, so some new e-services to use. So that will effect to us. But it will be in future." –Case 5

Both cases 6 and 7 emphasized the testability as an important goal of architectural design. The architecture should be testable, so that software testing can be automated.

7.2 Cloud computing

It appeared that cloud computing is extremely new concept among organizations and is not utilized very largely at the moment. One participant reported that they are using data storage services from a third party and another stated that the company has made some experiments by building their own cloud lab that could be possibly used as a testing environment. The rest of the participants reported that either some considerations have been made to map out the possibilities offered by cloud computing or no plans for cloud computing have been made so far. One participant deemed to extend the organization's service oriented architecture with cloud computing in near future.

Cloud computing could offer solutions for computing and data storage needs. Interviewees were asked about the suitability of cloud computing to fulfill the needs for heavy computing power and virtual data storage. The needs for these issues are represented in the table below.

Organization unit	**The need for computing power and virtual storage**
Case 1	Sometimes a need for huge computing power
Case 2	Neither is required.
Case 3	Not computing power but large amounts of data storage needed.
Case 4	Both are needed
Case 5	Sometimes the amount of data and storage needed is huge.
Case 6	No need for computing power, but need for storage
Case 7	No need for computing power, but need for storage
Case 8	Neither is required.

Case 9	Not massive computing power needed, significant amount of virtual storage needed.
Case 10	Rarely required.

Table 5: The need for massive storage capacity and computing power

The representative of case 3 saw the potential of cloud computing when the huge amount of computing power is needed:

> *”But if I were a manager in some kind of, for example Lippupalvelu, they sell tickets to concerts and stuff, if I were a manager there, I would absolutely buy software as a service from a cloud company, because their loads are so evenly distributed, so that if there’s a normal day, they might get away with one server. But if Bon Jovi comes to Helsinki, then they have 100 000 users for one hour [laughs]. So it’s, and they want to have a good reputation that the system didn’t crash when the big load came, so it would be really useful to buy computing power for one day, to get 100 machines now.”*

The representative also stated that they already have invested in their own data center that satisfies the capacity need at the moment. Because of this cloud computing is not a vital option for them at the current situation.

Data hosted somewhere by third party creates the issue of trust and data security. Security issues of cloud computing came out clearly. In some cases, the customer owns the test data and legal issues might forbid the data being transferred outside the company. The representative of case 1 says the following:

> *“…most customers don't trust virtualization or cloud computing, I think because of data security. They need to keep all those databases in their own premises. Or hosted premises, actually, very often. But they want to create a relationship of trust, really, to some provider. I guess it's, it will of course happen some time that one of those*

providers says that "OK, we are hosting your huge database, but it won't be in our vault here under this city, but it will be somewhere, and we don't say where". I'm sure this will happen and they will see what the customers will say. I don't know, I guess some people will say "OK, we don't care as long as you say that it will be secure". And people will say "OK, no matter what you say, it still needs to be in Finland…"

Case 3 also pointed out the legal issues of data. For example, Amazon's data services cannot be used because they are located outside the country where this company is operating. Utilizing cloud computing causes more third parties to be included in software development process. This means that communication framework between different parties must be established and the existing processes need to be adjusted. Two participants believed that if a third party (for example a cloud testing service provider) is involved in a project, more planning and documentation needs to be done. It is also believed that cloud computing (and SaaS) would lengthen the projects because of the increased complexity they will introduce.

7.3 Software as a service

As a delivery method for software, software as a service (SaaS) has been seriously considered in many companies and some participants mentioned that it was currently their delivery model. Interviewees were asked how this delivery model affects their testing (if they don't have SaaS, they evaluated how it would affect testing). A full spectrum of answers could be identified.

The representative of Case 1 guessed that SaaS would increase the complexity and would cause more testing needed to be done. Case 3 (who's delivery model is SaaS) thought that SaaS made it easier to use customers in testing if the software under test is available online. Case 4 also believed that customers can test the software easier when it is available online.

Another representative of Case 7 predicted that the company is moving towards SaaS. The software architecture as well as testing would both change with SaaS model:

> *"If and when we start changing our architecture, then I think our testing should be considered there as well, in early phase, so we can take it into consideration when we plan the new architecture, how it affects our testing and how our testing should be developed against that new architecture. We have some software as a service customers already, and that has not yet affected our testing at all. But if it comes more common, then I think it will affect. We need to consider that more than today."*

Case 9 stated that a SaaS model would require huge changes from them. Not only testing, but also the whole business model would change dramatically with a new delivery model:

> *"…we sell the product, and if the customer is silent, the product is running fine. If we would be somehow, or if we would sell the product as SaaS, so that it would be a continuing service from us, I think we would have a different view for the testing, because of course the customer who expect us to make continual changes and updates for the program."*

Case 8 had used SaaS model in some of the smaller projects. In larger projects, a new version of the application is installed on customer's premises every year. The representative deemed that testing would not be different in SaaS model than it is currently in traditional model:

> *"Maybe to clarify a bit, it's not that dramatic, the difference if it's a SAS or hosted service, or if we install in the customer's premises. Because it's exact-, technologically it's the same. And we usually have access to the customer's computers, although they are on their site, so.. I think the testing is.. don't differ at all"*

SaaS model promises many benefits for customers while introducing more challenges to service providers. If the domain is very critical, customer may rather want to upkeep the software in own premises. Case 8 mentioned that their customer wants to host the application by himself, because the confidential data must be kept inside the company walls. Case 6 gives its customers an option to choose whether they want the product to be installed on their premises or available as SaaS. The representative stated that software that is installed on customer’s premises requires more testing than the software hosted by developers.

> *“Yes. We also delivers it more as a.. so, we have a, our customers have a few possibilities, they can buy it as a SAS format. Software as a service. Or they can run it on the premises of their own, if they want. But mainly it's the as a service one. And how it affects at testing, I said earlier already, that if it's on the customer's premises, we have to have more testing phases to make sure that the environments are compatible for our product” –Case 6*

7.4 Testing tools

The need of testing tools appeared with almost every participant. The new testing tools are constantly searched, especially for areas of performance, security and automate testing in general. One participant said that open source tools offer opportunities for small companies who are not possibly able to purchase commercial tools that can be very expensive. Two participants were happy with the tools that they have. It was also pointed out that new tools (like any new asset taken into testing) require time to spend on learning. Open source tools are not being used very widely among the participants. Case 10 pointed out that availability of open source tools have increased. One participant preferred commercial tools because they are more stable and better documented with no licensing issues with them.

If software development and software architecture do not follow any good practices, paradigms and standards, the utilization of existing testing tools might become difficult. One example of this kind of situation was with Case 9. The representative said that their software architecture does not follow any modern guidelines of software development and design. Because of this, testing is challenging and the usage of testing tools is difficult or nearly impossible. Participant saw the testability as an important issue when building completely new software from scratch possibly in the future.

7.5 Discussion

Software testing is about to change when utilizing cloud computing or moving into SaaS –model. What exactly these changes are, remain unclear and cannot be predicted based on the data from the fourth interview round of the MASTO project. It seems that even though cloud computing has been in topics for rather a long time it was not largely utilized at the moment when the interview was conducted. SaaS –model is either considered or it is already used in many companies. A spectrum of answers were identified when participants were asked about how SaaS –model would affect software testing. This is probably because of the SaaS application can be developed in many different approaches and it can have many different architectures.

Issues of trust came out clearly when discussing about adoption of cloud computing. Cloud computing will bring data privacy issues to be considered because it may not be possible to store sensitive customer data on virtual disks owned by the cloud company. Cloud computing might require third parties to be involved in organizations' software development processes. This may require more project planning, management and documentation to be done. Test strategies and plan should as well contain the consideration how the information is transferred between different stakeholders.

It was pointed out that the role of software architecture becomes a very important issue when moving to SaaS model. Software architecture affects heavily on testability of the software, determining how easily the software under test can be tested with testing tools (or is it possible at all to test it with tools). Testability needs to be considered more carefully when designing software so that it becomes one quality requirement that remain important during analysis and design. The SaaS model sets its own requirements for software architecture makes architectural planning even more critical. With SaaS, not only the software architecture will change but also the operation model of the whole company will be changed dramatically as the vendor becomes responsible for maintaining of the software and continuous service. SaaS software cannot anymore be thrown at customers without sufficient testing, hoping that no defects will be found.

This study had some drawbacks. First, the fourth interview round was not entirely focused on new technology issues affecting software testing, but instead, it was still made as a part of MASTO project, and the new technology issues were not entirely exhaustive. MASTO project studies the testing standard ISO/IEC 29119, that does not have any reference to the concepts such as SOA, Cloud Computing or SaaS. Another problem of the interview was that the number of participants was low.

More research needs to be done in order to find out actual effects of cloud computing and SaaS on software testing. Candidates for a case study could be those organization units that already had made some investigations and pilot projects on cloud computing. The SaaS model was already present in some of the organization units and it will very likely become even more common. For future research, surveys can be conducted to find out those companies already utilizing cloud computing or offering their software in a pure SaaS. It is also important to reach the software testers to be interviewed in order to get reliable answers.

8 SUMMARY

Software testing is an essential phase in the software development process that can determine the success of the whole software project. Testing requires resources, which means humans, as well as the hardware and software assets. Testing must be well planned, scheduled and performed in a systematic way throughout the entire development process. New technologies are frequently introduced in the area of information technology industry. The new technologies challenge existing processes, methodologies, techniques and tools. The purpose of this master's thesis was to find out how software testing will change because of service oriented architecture and cloud computing.

Web services are the main instrument to build the service oriented architecture. SOA promises appealing benefits, but requires a significant amount of planning and commitment in order to be realized. SOA introduces new complexities and unique features that challenge many existing testing approaches. Late dynamic binding and numerous standards and specifications that can be on different maturity levels among the companies that are involved in the same SOA system, are examples of these issues. Most significant testing challenges are integration testing and non-functional testing. Integration testing is challenging because the services are owned by different companies, a possibility to access the service code is not guaranteed. Because services can be used in compositions and brokered by a client, quality of service becomes important issue and requires security and performance testing to be done. Because the services in a SOA solution are owned by different companies, a need arouses to improve service testability by providing additional information with services that are to be published to the service registry. The role of test automation must be highly emphasized in a SOA testing process. Numerous SOA testing tool vendors exist offering both commercial and open source tools to enhance the SOA testing.

Cloud computing is a major trend in the domain of information technology and many companies have investigated the possibilities of cloud computing. Small and medium-sized enterprises can reduce the costs by acquiring resources from the cloud. These resources can be infrastructures, platforms or software. Larger companies that already have invested heavily on datacenters can build own cloud environments and rent excess resources to customers. Cloud testing does not have an unambiguous definition. Testing own application with a cloud based tool, testing application that is available online or testing own datacenter can be regarded as cloud testing. Companies can reduce testing costs by utilizing cloud based testing tools and environments and paying only for the amount that is actually consumed. Performance testing is a suitable type of testing to be outsourced to the cloud because of the nature of it (sometimes environment is heavily needed by multiple projects and sometimes it is not needed at all). Building own testing laboratory can cost too much for a small company. Cloud can be utilized as an infrastructure or platform for applications that before were hosted on premises. Availability, performance and security issues become a major concern when considering different cloud service providers. A test strategy should estimate different vendors and find the most suitable one. Utilizing cloud platform in testing can ease the collaboration between testers that can be geographically in different locations.

Software as a service is counted as one cloud service type, but it can also viewed as a business model, or, a software delivery model. SaaS providers can get better customer insight by monitoring the application usage and steer the development and testing activities based on that information. SaaS vendors can optimize their systems for a specific platform or environment with no need of testing application with multiple different configurations. This may mean less testing efforts need to be done. SaaS vendors also learn more about usability issues of SaaS applications by monitoring the usage directly. This eases the usability testing significantly. Arranging Alpha, beta or acceptance testing in SaaS model is cheaper because no need of moving the application between customer's and developer's sites. The effect of SaaS model on software testing will be dependent on how the application is built. SaaS application can have many different types of architectures – it can be a legacy application that is moved to cloud environment, it can be designed to obey multitenancy, or it can be a SOA application. It is predicted that next generation SaaS applications are going to be more service oriented. This means that SaaS application development will face the challenges of SOA testing as well. Changes to software testing process will definitely be dependent on the choices and decisions that are made in analysis and design phases of software development and these phases are probably facing more challenges than testing. Viewpoints from practice also support this statement. Multitenancy and SOA make the testing more complex and creates the need for a suitable quality assurance system with monitoring and management tools that help to indicate how the QoS attributes are being delivered to the service users. SaaS will not only affect the several phases of software development, but also the whole business model of a SaaS provider is needed to change as the service provider is going to get more responsibilities in maintaining the application on behalf of the customer. The provider must now offer continuous service to the users.

MASTO projects interview data was utilized to gather some viewpoints from the industry. No formal research method was used in the analysis of the data. Instead, the data was viewed in the context of interesting topics that are related to the focus area of this thesis. The topics include software architecture, cloud computing, software as a service and testing tools. Based on the observations, cloud computing is currently considered in many companies but not utilized at a large scale. It is most likely to be become more common in near future. Results pointed out that SaaS –model and cloud computing will cause changes to existing testing processes. What exactly are these changes remain unclear and to find out these changes, more research needs to be done. Grounded theory or case studies would be a suitable research method to retrieve this information but right organization units, those that are currently leveraging cloud computing with a high degree and those who are offering pure SaaS to their customers. Surveys could be one way to reach these companies. SOA testing in practice is also a topic that seems to be hardly investigated. Companies that build SOA solutions could be reached with similar methods.

BIBLIOGRAPHY

Amazon, 2010. Amazon Elastic Compute Cloud (Amazon EC2). Available at: http://aws.amazon.com/ec2/ [Accessed February 25, 2011].

AppLabs, 2009. Approach to Cloud Testing. Available at: http://www.applabs.com/html/download_Knowledgecenter.html/3238?nid=19.

AppLabs, 2008. Future of Software Testing. Available at: http://www.applabs.com/ap-private/pdf-download/3263%3Fnid%3D569%2526Print%3Dpdf.

AppLabs, 2007. Testing the Cloud. Available at: http://www.qaguild.com/upload/app_whitepaper_testing_the_cloud_1v00.pdf.

Barber, S., 2006. SOA Testing Challenges. Available at: http://www.perftestplus.com/resources/SOA_challenges_ppt.pdf.

Baresi, L., 2007. *Test and analysis of Web services*, Berlin [Germany] ;;New York: Springer.

Baresi, L. & Nitto, E.D. eds., 2007a. *Test and Analysis of Web Services*, Berlin, Heidelberg: Springer Berlin Heidelberg. Available at: http://www.springerlink.com/index/10.1007/978-3-540-72912-9.

Baresi, L. & Nitto, E.D. eds., 2007b. *Test and Analysis of Web Services*, Berlin, Heidelberg: Springer Berlin Heidelberg. Available at: http://www.springerlink.com/index/10.1007/978-3-540-72912-9.

Bartolini, C. et al., 2009. Whitening SOA testing. In *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering on European software engineering conference and foundations of software engineering symposium - ESEC/FSE \uc0\u8217{}09*. the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. Amsterdam, The Netherlands, p. 161. Available at: http://portal.acm.org/citation.cfm?doid=1595696.1595721.

Bezemer, C.-P. & Zaidman, A., 2010. Multi-tenant SaaS applications. In *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE) on - IWPSE-EVOL \uc0\u8217{}10*. the Joint

ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE). Antwerp, Belgium, p. 88. Available at: http://portal.acm.org/citation.cfm?doid=1862372.1862393 [Accessed March 30, 2011].

Buyya, R. et al., 2009. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6), pp.599-616.

Cagle, K., 2009. SOA is Dead? It's About Time! - O'Reilly Broadcast. Available at: http://broadcast.oreilly.com/2009/01/soa-is-dead-its-about-time.html [Accessed February 25, 2011].

Chappel, D., 2009. Introducing Windows Azure. Available at: http://www.davidchappell.com/writing/white_papers/Introducing_Windows_Azure_v1-Chappell.pdf.

Chappel, D., 2010. Introducing Windows Azure Platform. Available at: http://www.davidchappell.com/writing/white_papers/Introducing_the_Windows_Azure_Platform,_v1.4--Chappell.pdf.

Chou, D., 2010. Microsoft Cloud Computing Platform.

CIOL, 2008. SaaS – A revolutionary approach for building web applications - CIOL Feature. Available at: http://www.ciol.com/ec/feature/saas-%E2%80%93-a-revolutionary-approach-for-building-web-applications/23108103050/0/ [Accessed March 8, 2011].

Collard, R., 2009. Performance Innocations, Testing Implications. *Software Test & Performance. Vol. 6, no. 8, pp. 19-20 Aug. 2009*.

Dubey, A. & Wagle, D., 2007. Delivering software as a service. *The McKinsey Quarterly Web exclusive*. Available at: http://ai.kaist.ac.kr/~jkim/cs489-2007/Resources/DeliveringSWasaService.pdf.

Dumke, R.R. et al. eds., 2008. *Software Process and Product Measurement*, Berlin, Heidelberg: Springer Berlin Heidelberg. Available at: http://www.springerlink.com/index/10.1007/978-3-540-89403-2.

Durkee, D., 2010. Why cloud computing will never be free. *Communications of the ACM*, 53(5), p.62.

Ebert, C., 2008. A Brief History of Software Technology. *IEEE Software*, 25(6), pp.22-25.

Erl, T., 2005. *Service-oriented architecture : concepts, technology, and design*, Upper Saddle River NJ: Prentice Hall Professional Technical Reference.

Erl, T., 2008. *SOA : principles of service design*, Upper Saddle River NJ: Prentice Hall.

Everett, G., 2007. *Software testing : testing across the entire software development life cycle*, [Piscataway NJ] ;Hoboken N.J. IEEE Press ;;Wiley-Interscience.

Girmonsky, A., 2009. Cloud-Testing White Paper. Available at: http://www.cloud-intelligence.com/sites/www.cloud-intelligence.com/files/Cloud%20Testing%20White%20Paper_0.pdf.

Google, 2011a. Google Apps for Business. Available at: http://www.google.com/apps/intl/fi/business/index.html#utm_campaign=fi&utm_source=fi-ha-emea-fi-bk&utm_medium=ha&utm_term=google%20apps [Accessed March 8, 2011].

Google, 2011b. What Is Google App Engine? - Google App Engine - Google Code. Available at: http://code.google.com/intl/fi-FI/appengine/docs/whatisgoogleappengine.html [Accessed February 25, 2011].

Gotel, O., SEAFOOD, 2009. *Software engineering approaches for offshore and outsourced development third international conference, SEAFOOD 2009, Zurich, Switzerland, July 2-3, 2009 : proceedings*, Berlin ;;New York: Springer.

Greengard, S., 2010. Cloud computing and developing nations. *Communications of the ACM*, 53(5), p.18.

Haikala, I., 2004. *Ohjelmistotuotanto* 10th ed., Helsinki: Talentum.

Haines, M.N. & Rothenberger, M.A., 2010. How a service-oriented architecture may change the software development process. *Communications of the ACM*, 53(8), p.135.

Harris, T., 2007. SOA Test Methodology. Available at: http://www.thbs.com/pdfs/SOA_Test_Methodology.pdf.

Hull, R. & Su, J., 2005. Tools for composite web services. *ACM SIGMOD Record*, 34(2), p.86.

Hurwitz, J., 2007. *Service oriented architecture for dummies*, Hoboken NJ: Wiley Publishing.

IBM, 2007. Testing SOA Applications with IBM Rational quality management solutions. Available at: ftp://ftp.software.ibm.com/software/rational/web/whitepapers/10708268_Testing_SOA_WP_acc.pdf.

Josuttis, N., 2007. *SOA in practice*, Farnham: O'Reilly.

Kit, E., 1995. *Software testing in the real world : improving the process*, New York N.Y. ;Wokingham England ;;Reading Mass. ACM Press ;;Addison-Wesley Pub. Co.

Kommalapati, H., 2010. Cloud Computing - Windows Azure Platform for Enterprises. Available at: http://msdn.microsoft.com/en-us/magazine/ee309870.aspx [Accessed February 8, 2011].

Kommalapati, H., 2007. SaaS and SOA - Hanuk's Microsoft Platform Strategy Blog - Site Home - MSDN Blogs. Available at: http://blogs.msdn.com/b/hanuk/archive/2007/04/08/saas-and-soa.aspx [Accessed March 22, 2011].

Linthicum, D., 2010. *Cloud computing and SOA convergence in your enterprise : a step-by-step guide*, Upper Saddle River NJ: Addison-Wesley.

Linthicum, D., 2009. SOA-cloud computing relationship leaves some folks in a fog -- Government Computer News. Available at: http://gcn.com/articles/2009/03/09/guest-commentary-soa-cloud.aspx [Accessed March 22, 2011].

Louridas, P., 2010. Up in the Air: Moving Your Applications to the Cloud. *IEEE Software*, 27(4), pp.6-11.

Lucia, A. & Ferrucci, F. eds., 2009. *Software Engineering*, Berlin, Heidelberg: Springer Berlin Heidelberg. Available at: http://www.springerlink.com/index/10.1007/978-3-540-95888-8.

Macy, J., 2010. Hidden Cost of Open Source SOA Testing | Open Source Magazine. Available at: http://opensource.sys-con.com/node/1233468 [Accessed March 22, 2011].

Macy, J., 2009. SOA Testing Tools & Best Practices: Limits of Open Source SOA Testing Tools. Available at: http://www.soatesting.com/2009/06/limits-of-open-source-soa-testing-tools.html [Accessed March 22, 2011].

Manes, A., T, 2009. Application Platform Strategies Blog: SOA is Dead; Long Live Services. Available at: http://apsblog.burtongroup.com/2009/01/soa-is-dead-long-live-services.html [Accessed February 25, 2011].

McConnell, S., 2004. *Code complete* 2nd ed., Redmond Wash. Microsoft Press.

Melendy, B., 2008. Brad Melendy » SOA vs SaaS… What's the difference? Available at: http://brad.melendy.com/?p=16 [Accessed March 22, 2011].

MSDN, 2011. Application Life Cycle Management for Windows Azure Applications. Available at: http://msdn.microsoft.com/en-us/library/ff803362.aspx [Accessed March 22, 2011].

Myers, G., 2004. *The art of software testing* 2nd ed., Hoboken N.J. John Wiley & Sons.

OASIS, 2006. Reference Model for Service OrientedArchitecture 1.0. Available at: http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf.

OASIS, 2004. UDDI Version 3.0.2. Available at: http://uddi.org/pubs/uddi_v3.htm [Accessed February 25, 2011].

Oracle, 2008. Oracle SaaS Platform: Building On-Demand Applications. Available at: http://www.oracle.com/us/technologies/cloud/026989.pdf.

Palacios, M., García-Fanjul, J. & Tuya, J., 2010. Testing in Service Oriented Architectures with dynamic binding: A mapping study. *Information and Software Technology*, 53(3), pp.171-189.

Papazoglou, M., 2008. *Web services : principles and technology*, Harlow England ;;New York: Pearson/Prentice Hall.

Reese, G., 2009. *Cloud application architectures*, Sebastopol Calif. O'Reilly.

Ribarov, L., Manova, I. & Ilieva, S., 2007. Testing in a Service-Oriented World. *Proceedings of the International Conference on Information Technology*, 2007(1). Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.83.2314&rep=rep1&type=pdf.

Rittinghouse, J., 2010. *Cloud computing : implementation, management, and security*, Boca Raton: CRC Press.

Riungu, L.M., Taipale, O. & Smolander, K., 2010. Research Issues for Software Testing in the Cloud. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science*. 2010 IEEE 2nd International Conference on Cloud Computing Technology and Science (CloudCom). Indianapolis, IN, USA, pp. 557-564. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5708500.

Robillard, P., 2003. *Software engineering process with the UPEDU*, Boston: Addison Wesley.

Salesforce, 2011. CRM Applications & Software Solutions - salesforce.com Europe. Available at: http://www.salesforce.com/eu/crm/products.jsp [Accessed March 8, 2011].

SIIA, 2001. Software as a Service: Strategic Backgrounder. Available at: http://www.siia.net/estore/pubs/SSB-01.pdf.

SOA Systems Inc., 2009a. SOA Specifications - WS-* Specs. Available at: http://www.soaspecs.com/ws.php [Accessed February 25, 2011].

SOA Systems Inc., 2009b. SOA Specifications - XML - SOAP in a Nutshell. Available at: http://www.soaspecs.com/soap2.php [Accessed February 25, 2011].

SPIRENT, 2010a. Cloud Computing Testing.

SPIRENT, 2009. Data Center Testing: A Holistic Approach. Available at: http://www.spirent.com/~/media/White%20Papers/Broadband/PAB/Data_Center_Testing_WhitePaper.ashx.

SPIRENT, 2010b. The Ins and Outs of Cloud Computing and Its Impact on the Network. Available at: http://www.spirent.com/White-Papers/Broadband/PAB/Cloud_Computing_WhitePaper.aspx.

Suzuki, K. et al. eds., 2008. *Testing of Software and Communicating Systems*, Berlin, Heidelberg: Springer Berlin Heidelberg. Available at: http://www.springerlink.com/index/10.1007/978-3-540-68524-1.

Torode, C., 2009. Tapping the cloud as a software testing service. Available at: http://searchcio-midmarket.techtarget.com/news/1356175/Tapping-the-cloud-as-a-software-testing-service [Accessed March 22, 2011].

W3C, 2007. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). Available at: http://www.w3.org/TR/soap12-part1/ [Accessed February 25, 2011].

W3C, 2001. Web Service Definition Language (WSDL). Available at: http://www.w3.org/TR/wsdl [Accessed February 25, 2011].

Vainikka, M., 2010. Seminaarityö: SaaS - Software as a Service.

Vaquero, L.M. et al., 2008. A break in the clouds. *ACM SIGCOMM Computer Communication Review*, 39(1), p.50.

Xiaoying Bai, 2005. WSDL-Based Automatic Test Case Generation for Web Services Testing. In *IEEE International Workshop on Service-Oriented System Engineering (SOSE'05)*. IEEE International Workshop on Service-Oriented System Engineering (SOSE'05). Beijing, China, pp. 215-220. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1551150.

Yu, Q. et al., 2006. Deploying and managing Web services: issues, solutions, and directions. *The VLDB Journal*, 17(3), pp.537-572.

Yunus, M., 2010a. Federated SOA: A Pre-Requisite for Enterprise Cloud Computing | Cloud Computing Journal. Available at: http://cloudcomputing.sys-con.com/node/1233457 [Accessed March 2, 2011].

Yunus, M., 2010b. Using SQL Azure for SOA Quality Testing | Cloud Computing Journal. Available at: http://soa.sys-con.com/node/1379631 [Accessed March 1, 2011].

APPENDIX

A: SOA design principles

Service contract	Services have a contract that is standardized so that natural interoperability between services is formed. Service contract is provided with the service and it comprises of a technical interface or one or more service description documents The functionality of a service is expressed through the service contract.
Loose coupling	Services have a relationship that minimizes dependencies and requires only that services are aware of each other with the help of service contracts. Loose coupling eases the application of other principles.
Abstraction	Services hide everything that is not described in a service contract and only publishes the necessary information to other services. Only the service contract is visible to others and details of underlying technology, logic and function is hidden from outside world.
Reusability	This is principle is the most important principle of SOA but maybe the most challenging principle to achieve in practice. Services are designed to be reusable enterprise resources. By having a reusable inventory of services, the business agility and the return of investment is increased as the organization can react rapidly to changing business automation requirements. Achieving reusability includes several design characteristics. The service logic should be as general as possible allowing it to facilitate numerous usage scenarios by different types of service consumers. Services should be designed to be accessed concurrently so that multiple consumers can access them simultaneously. Reusability introduces more challenges for design and testing the services.
Autonomy	Autonomous services manage and are responsible of the functionality they encapsulate. The goal of autonomy is to enable services to have a high level of control over their underlying runtime execution environment. Autonomy results the increased level of runtime reliability, performance and behavioral predictability when services are being reused and composed.
Statelessness	Statelessness means that services minimize the resource consumption by deferring the management of state information whenever possible. The principle emphasizes the need to reduce or eliminate system resource consumption because of the unnecessary state management processing.
Discoverability	In discoverability, services are designed to be descriptive and supplemented with communicative meta data so that they can be effectively discovered and interpreted with different kinds of finding mechanisms. The goal of this principle is to make services positioned as discoverable resources within enterprise. The purpose and capabilities of each service should be expressed clearly so that they can be interpreted by humans and other services.
Composability	Composability introduces new design considerations that ensure services being able to participate in multiple compositions to solve multiple larger problems. Following this design principle, services can effectively be participants in a composition, no matter the size or complexity of the composition. Reusability and composability have similar goals, because service composition is one form of service reuse. Achieving service composability means achieving the ultimate goal of service oriented computing. By establishing an enterprise solution logic represented by an inventory of reusable services, future business automation requirements can be fulfilled through service composition.

Table 6: SOA design principles (Baresi & Nitto 2007a)

B: QoS Attributes of SOA

Runtime Quality	
Availability	The absence of service downtimes. Availability is the probability that the service is available. The higher the value, the more available the service is. A TTR (time-to-repair) value is usually represented with availability. It represents the time that it takes to repair a broken service.
Accessibility	A degree with which a service request is served. It may be a probability that describes the change of successful service utilization. A high degree of accessibility means that clients can use the service easily.
Integrity	The degree with which a Web service performs its tasks according to its WSDL and Service-Level Agreement. The higher the integrity, the closer is service's functionality to its WSDL or SLA.
Performance (or response time)	Performance is measured in terms of throughput and latency. Throughput is the number of service requests served at a given time period. Latency is the time spent on sending the request and receiving the response. High throughput and low latency mean good performance of the service.
Reliability	The ability of a service to function correctly and consistently. Measures the ability of service's operation to be executed within the maximum expected time frame. A reliable service provides the same service quality despite system or network failures. The reliability of a Web service is usually represented as a number of transactional failures in one month or year.
Scalability	A service's ability to serve the requests despite the variations in the volume of requests. High accessibility can be achieved by building scalable services.
Security	Issues such as authentication, authorization, message integrity and confidentiality are included in security. Security is important, because web services operate over the Internet. The desired level of security for the service that is utilized is defined in the SLA. Service provider must maintain this level of security.
Transactionality	Sometimes Web services need transactional behavior. If a particular service requires this behavior, it is described in SLA.
Business Quality	
Cost	Measures the units of money that the service client need to pay for using the service
Reputation	Service clients can evaluate the service after utilizing it
Regulatory	Measures the Web service's conformance of rules, laws and compliance with standards and the established service level agreement. It is important that service requestors know, which version of specific standard the service is conforming (for example, WSDL version 2.0). Service providers must comply with those standards specified in the Service-Level agreement (SLA) between service providers and requestors.

Table 7: Quality of service attributes in SOA (Baresi and Nitto 2007a)

C: The possible contents of an SLA

Purpose	The reasons why the SLA was created
Parties	Parties that are involved in SLA and roles of these parties (service provider, service consumer or client)
Validity Period	The period of time when the SLA is valid
Scope	Defines the services that are covered in agreement
Restrictions	Defines the necessary steps to be taken in order for the requested service levels to be provided
Service-level objectives	Levels of objectives that both the provider and client agree on. Includes usually a set of service level indicators, like availability, performance and reliability that each have a target level that is tried to achieve.
Penalties	Defines the actions that are taken if the objectives defined in SLA are failed to meet.
Optional services	Specifies the services that might be required in case of a failure
Exclusion terms	Specifies what is now covered in the SLA
Administration	Defines the processes and measurable objectives in an SLA and defines the organizational authority for overseeing them

Table 8: Possible contents of Service-level agreement (Baresi & Nitto 2007a)