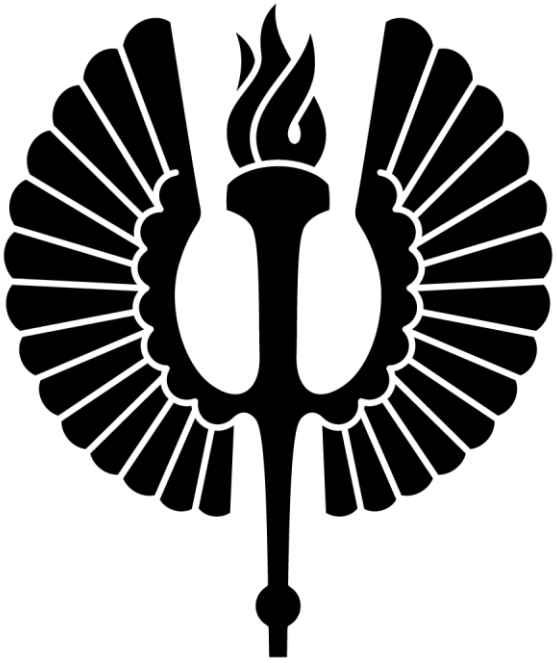
When to Utilize Software as a Service



UNIVERSITY OF TURKU
Department of Information Technology
Master of Science in Technology Thesis
June 2011
Markus Jakonen

Inspectors:
Ville Leppänen
Tuomas Mäkilä

UNIVERSITY OF TURKU
Department of Information Technology

JAKONEN, MARKUS: When to Utilize Software as a Service

Master of Science in Technology Thesis, 121 p.
Software Engineering
June 2011

Cloud computing enables on-demand network access to shared resources (e.g., computation, networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort. Cloud computing refers to both the applications delivered as services over the Internet and the hardware and system software in the data centers.

Software as a service (SaaS) is part of cloud computing. It is one of the cloud service models. SaaS is software deployed as a hosted service and accessed over the Internet. In SaaS, the consumer uses the provider's applications running in the cloud. SaaS separates the possession and ownership of software from its use. The applications can be accessed from any device through a thin client interface. A typical SaaS application is used with a web browser based on monthly pricing.

In this thesis, the characteristics of cloud computing and SaaS are presented. Also, a few implementation platforms for SaaS are discussed. Then, four different SaaS implementation cases and one transformation case are deliberated. The pros and cons of SaaS are studied. This is done based on literature references and analysis of the SaaS implementations and the transformation case. The analysis is done both from the customer's and service provider's point of view. In addition, the pros and cons of on-premises software are listed.

The purpose of this thesis is to find when SaaS should be utilized and when it is better to choose a traditional on-premises software. The qualities of SaaS bring many benefits both for the customer as well as the provider.

A customer should utilize SaaS when it provides cost savings, ease, and scalability over on-premises software. SaaS is reasonable when the customer does not need tailoring, but he only needs a simple, general-purpose service, and the application supports customer's core business.

A provider should utilize SaaS when it offers cost savings, scalability, faster development, and wider customer base over on-premises software. It is wise to choose SaaS when the application is cheap, aimed at mass market, needs frequent updating, needs high performance computing, needs storing large amounts of data, or there is some other direct value from the cloud infrastructure.

Keywords: Software as a Service, SaaS, service-oriented software, on-demand software, cloud computing

TURUN YLIOPISTO
Informaatioteknologian laitos

JAKONEN, MARKUS: Milloin hyödyntää Software as a Service -mallia

Diplomityö, 121 s.
Ohjelmistotekniikka
Kesäkuu 2011

Pilvilaskenta (cloud computing) mahdollistaa *on-demand*-periaatteella pääsyn jaettuihin resursseihin (esim. laskentateho, verkot, palvelimet, tallennustila, sovellukset ja palvelut), joita voidaan nopeasti varata ja vapauttaa hyvin vähäisellä työmäärällä. Pilvilaskenta viittaa Internetin yli palveluina jaettuihin sovelluksiin ja myös datakeskusten laitteistoihin sekä järjestelmäohjelmistoihin.

Software as a service (SaaS) on osa pilvilaskentaa. Se on yksi pilvipalvelumalleista. SaaS on isännöity palvelu ja sitä käytetään Internetin yli. SaaSissa kuluttaja käyttää tuottajan ohjelmistoja, joita ajetaan pilvessä. SaaS erottelee ohjelmiston omistamisen ja sen käytön. Sovelluksia voidaan käyttää millä tahansa laitteella *thin client* -rajapinnan kautta. Tavallisesti SaaS-sovellusta käytetään web-selaimella kuukausihinnoittelulla.

Tässä työssä esitellään pilvilaskennan ja SaaSin ominaispiirteet. Myös muutamaa SaaS-toteutusalustaa käydään läpi. Tämän jälkeen pohditaan neljää eri SaaS-toteutusta ja yhtä SaaS-muutostapausta. SaaS:n etuja ja haittoja tutkitaan kirjallisuuslähteiden ja SaaS-toteutuksien ja -muutoksen analysoinnin perusteella. Analysointi tehdään sekä asiakkaan että palveluntarjoajan kannalta. Lisäksi listataan *on-premises*-ohjelmistojen etuja ja haittoja.

Tämän työn haaste on selvittää, milloin SaaSia tulisi hyödyntää ja milloin kannattaa valita perinteinen *on-premises*-ohjelmisto. SaaSin ominaisuudet tuovat monia etuja sekä asiakkaalle että tuottajalle.

Asiakkaan tulisi valita SaaS, kun se tarjoaa kustannussäästöjä, helppoutta ja skaalautuvuutta *on-premises*-ohjelmistoihin verrattuna. SaaS on järkevä vaihtoehto silloin, kun asiakas ei tarvitse räätälöintiä, vaan hänelle riittää yksinkertainen, yleiskäyttöinen palvelu, joka tukee hänen ydinliiketoimintaansa.

Palveluntarjoajan kannattaa hyödyntää SaaS:ä silloin, kun se tarjoaa kustannussäästöjä, skaalautuvuutta, nopeamman kehityksen ja laajemman asiakaskunnan verrattuna *on-premises*-ohjelmistoihin. On järkevää valita SaaS, kun sovellus on halpa, suunnattu massamarkkinoille, tarvitsee tiheää päivittämistä, tarvitsee suurta laskentakapasiteettia, tarvitsee suurten datamäärien tallentamista tai kun pilvi-infrastruktuurista saadaan jotain muuta suoraa arvoa.

Avainsanat: Software as a Service, SaaS, ohjelmistoja palveluna, ohjelmistovuokraus, sovellusvuokraus, pilvilaskenta

CONTENTS

1 INTRODUCTION

Software as a service (SaaS) separates the possession and ownership of software from its use. The idea of SaaS is to deliver software's functionality as a set of services. Services can be configured and bound at delivery time. This kind of software deployment model can overcome many current limitations constraining software use, deployment, and evolution. [**1**]

Today, SaaS and cloud computing are hot topics. The phenomenon of SaaS is comparatively new, but SaaS services are fast becoming common. SaaS can and will provide new opportunities both technology-wise and business-wise. New SaaS services appear in the market every day. SaaS offers ease of use for the customer while offering cost savings for the service provider. Also importantly, SaaS brings the buyer the flexibility to unsubscribe if the service fails to fulfill the expectations. This is one of the economic benefits for the customer provided by SaaS.

Many of us have actually used SaaS in some form, possibly without even thinking about it. Web-based e-mail services are SaaS, and so are on-demand music streaming services. If you have enrolled for an event and the enrollment was done in the web, you have probably used SaaS.

There are many definitions of SaaS. A few, descriptive definitions will be presented in Chapter 2. Characteristics and service levels of cloud computing will be discussed as well. Section 2.3 will cover characteristics of SaaS, and it will be followed by a presentation of a maturity model for SaaS. Chapter 2 also includes some discussion on SaaS from the business point of view. Finally, three current implementation platforms for SaaS applications will be introduced.

In Chapter 3, four SaaS implementation cases and a SaaS transformation case will be studied. First, a case will be described based on the source article. Then, its SaaS characteristics and SaaS utilization will be deliberated. It will be also discussed that

what kind of additional value the sample case has received from the SaaS qualities. What are the benefits for the particular case?

Chapter 4 will compare SaaS to on-premises software. First, the benefits and drawbacks of SaaS will be discussed. This is done by performing comparative analysis on a SWOT analysis of SaaS and my own findings presented in Chapter 3. Section 4.1 covers the analysis of SaaS from the customer's point of view. Section 4.2 covers the analysis of SaaS from the provider's point of view, respectively. The SWOT analysis discussed in Sections 4.1 and 4.2 is based on a valuable source of this thesis. In addition to the SWOT analysis, some additional analysis is done in Sections 4.1.5 and 4.2.5. Sections 4.1.6 and 4.2.6 provide summaries of the analysis done in Sections 4.1.1 through 4.1.5 and Sections 4.2.1 through 4.2.5. In Section 4.3, the pros and cons of on-premises software are listed. The lists are my own reasoning and mostly based on the findings in Sections 4.1 and 4.2. The pros and cons will be presented from the customer's and the provider's point of view separately yet again.

In Chapter 5, a summary of the analysis done in Chapter 4 is given in Section 5.1. In addition, it will be discussed that what makes a good SaaS application from the customer's and provider's viewpoint. This discussion is a summary of the findings in Chapter 4. The conclusions of this thesis will be presented in Section 5.2. In Section 5.2.1, it will be discussed that which applications fit the SaaS model and which do not. Section 5.2.2 will have a discussion on the research question: "When to utilize SaaS?"

The challenge of this thesis is to discover whether or not to use SaaS. In other words, when should one utilize SaaS and when it is sensible to use on-premises software? What is a good SaaS application like? Also, one challenge is to present the essential points of the SaaS implementation cases. Furthermore, a challenge is to analyze SaaS based on different sources, including my own reasoning.

My examination suggests that a customer should utilize SaaS when it provides cost savings, ease, flexibility in pricing, and scalability over on-premises software. SaaS is reasonable when the customer does not need tailoring, but he only needs a simple, general-purpose service, and the application supports customer's core business.

My study also suggests that a provider should utilize SaaS when it offers cost savings, scalability, faster development, and wider customer base over on-premises software. It is wise to choose SaaS when the application is cheap, aimed at mass market, needs frequent updating, needs high performance computing, needs storing large amounts of data, or there is some other direct value from the cloud infrastructure.

2 SOFTWARE AS A SERVICE

What is software as a service all about? According to one general definition it is "Software deployed as a hosted service and accessed over the Internet" [**2**]. By this definition, SaaS includes a wide variety of services and applications, and can be divided into two major categories: business-oriented and consumer-oriented services. The former covers subscription-based services, e.g. customer relationship management (CRM) software, offered to enterprises and organizations of all sizes. The latter embodies advertising-based services, e.g. web-based e-mail, offered to the general public at no cost, and supported by advertising.

Another commercially oriented definition [**3**] says that "Instead of installing and maintaining software, you simply access it via the Internet, freeing yourself from complex software and hardware management." SaaS applications are sometimes called web-based software, on-demand software, or hosted software. Regardless of the name, SaaS applications run on a SaaS provider's servers. Access to the application, i.e. security, availability, and performance, is managed by the provider.

To present one more definition for the concept of "SaaS", Turner et al. [**1**] have stated the following: "The software as a service model composes services dynamically, as needed, by binding several lower-level services – thus overcoming many limitations that constrain traditional software use, deployment, and evolution."

Overview of a SaaS application is presented in Figure 1 and roles in a SaaS scenario in Figure 2.

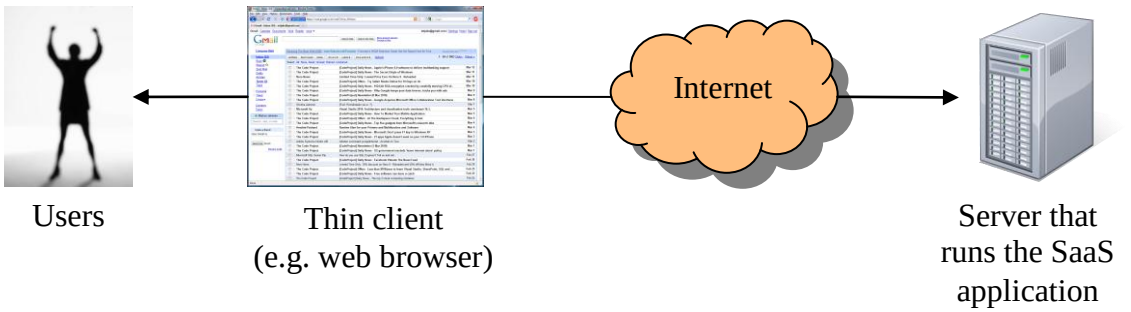


Figure 1. Overview of a SaaS application.

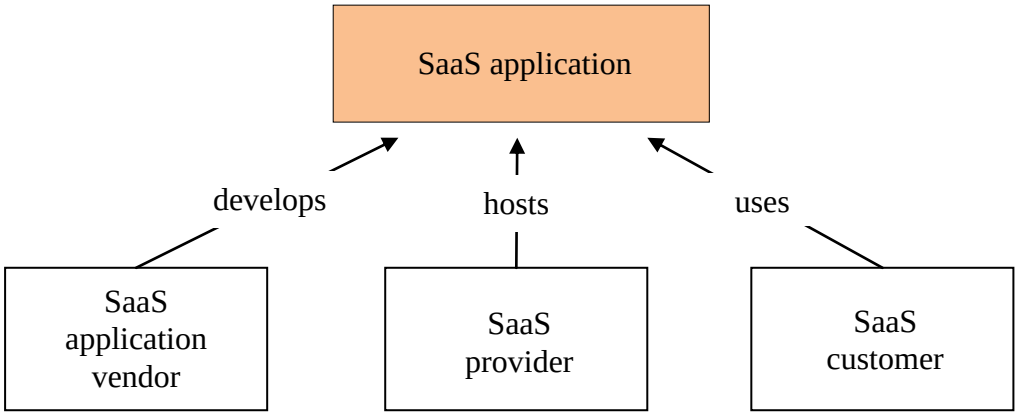


Figure 2. Roles in a SaaS scenario (adapted from [4].)

2.1 Cloud Computing

The NIST definition of Cloud Computing [**5**] states that "*Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential* ***characteristics,*** *three* ***service models****, and four* ***deployment models***."

Cloud software takes full advantage of the cloud paradigm by being service oriented and focusing on statelessness, low coupling, modularity, and semantic interoperability. [**5**]

Armbrust et al. [**6**] define cloud computing (CC) as follows: "…refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centers that provide those services." In this definition the services mean SaaS.

2.1.1 Cloud Characteristics

Mell and Grance [**5**] present five key cloud characteristics in their definition. *On-demand self-service* implies that a consumer can unilaterally purchase computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.

Broad network access comes from the quality that capabilities are available over the network and accessed through standard mechanisms that support use by heterogeneous thin or thick client platforms (e.g., mobile phones, laptops, and PDAs).

Resource pooling, or location independence, means that the provider's computing resources are pooled to serve multiple consumers using a multi-tenant model (see Section 2.4), with different physical and virtual resources that are dynamically assigned and reassigned according to consumer demand. The customer generally has no control or knowledge over the exact location of the provided resources, but may be able to specify the location at country, state, or datacenter level. Examples of resources include processing, storage, memory, network bandwidth, and virtual machines.

Rapid elasticity denotes that capabilities can be rapidly and elastically provisioned, possibly automatically, to quickly scale out and rapidly released to quickly scale in. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time.

Measured service means that cloud systems automatically manage and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., processing, storage, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, which provides transparency for both the provider and consumer of the utilized service.

Other common cloud characteristics include massive scale, homogeneity, virtualization, resilient computing, low cost software, geographic distribution, service orientation, and advanced security technologies. [**7**]

2.1.2 Cloud Service Models

Cloud computing contains three kinds of service models, or levels. These models are discussed in this section and presented in Figure 3. To be considered "cloud" they must be deployed on top of cloud infrastructure that has the key characteristics [**7**].

2.1.2.1 Cloud Software as a Service (SaaS)

SaaS is a model in which the consumer uses the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through a thin client interface such as a web browser (e.g., web-based e-mail). The consumer does not manage or control the underlying cloud infrastructure, network, servers, operating systems, storage, or individual application capabilities, with the possible exception of limited user-specific application configuration settings. Briefly: "*Use provider's applications over a network*". [**5**], [**7**]

2.1.2.2 Cloud Platform as a Service (PaaS)

In the PaaS model, the consumer has the capability to deploy onto the cloud infrastructure consumer-created applications using programming languages and tools (e.g., .NET, Java, Python) supported by the provider. The consumer does not manage or control the underlying cloud infrastructure, network, servers, operating systems, or storage. However, the consumer has control over the deployed applications and possibly application hosting environment configurations. In summary: "*Deploy customer-created applications to a cloud*". [**5**], [**7**]

2.1.2.3 Cloud Infrastructure as a Service (IaaS)

In the IaaS model, the capability provided to the consumer is to acquire processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, including operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure, but has control over operating systems, storage, and deployed applications. In addition, the consumer may be able to select networking components, e.g. host firewalls and load balancers. In short: "*Rent processing, storage, network capacity, and other fundamental computing resources*". [**5**], [**7**]

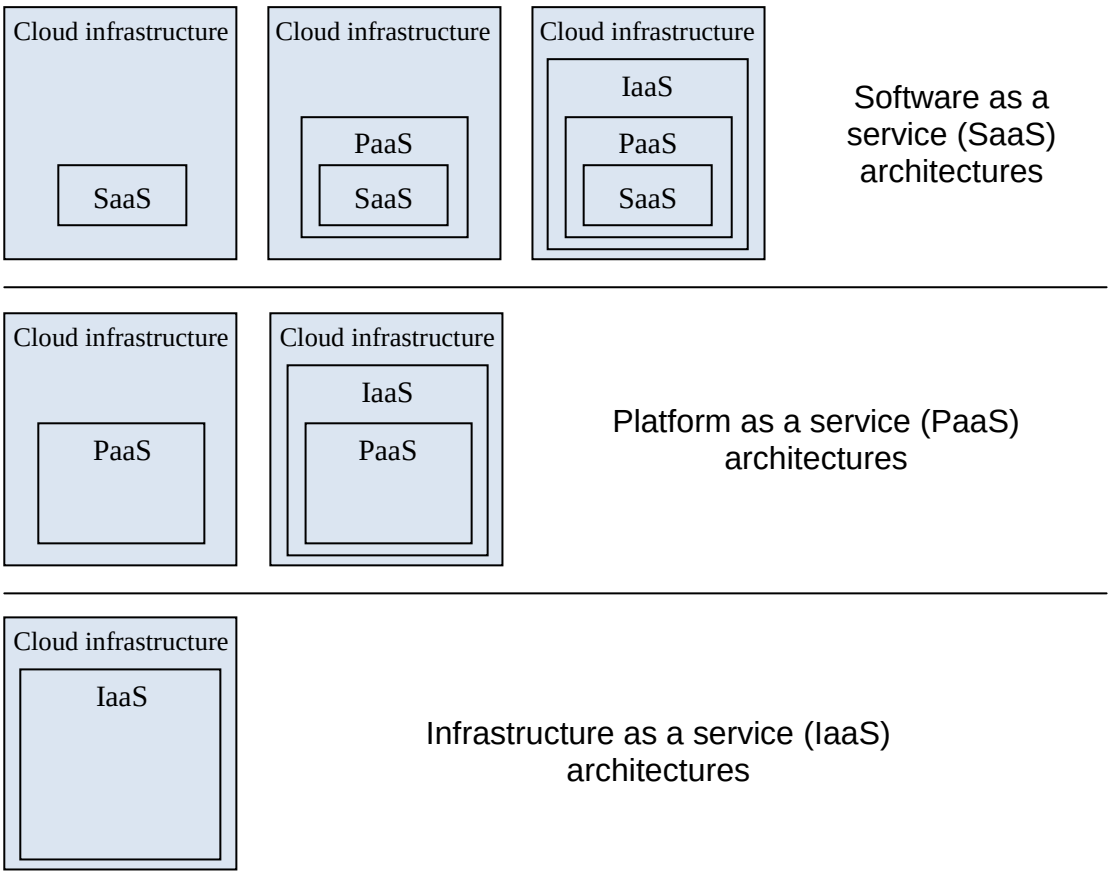


Figure 3. Cloud service model architectures (adapted from [7].)

2.1.3 Cloud Deployment Models

Finally, Mell and Grance [**5**], [**7**] come out with four deployment models. *Private cloud* denotes that the cloud infrastructure is operated solely for an organization. It may be administered by the organization or a third party, and may exist on premise or off premise. A private cloud can be enterprise owned or leased.

In *community cloud* model, the cloud infrastructure is shared by several organizations and supports a specific community with shared concerns (e.g., mission, security, requirements, policy, and compliance considerations). It may be managed by the organizations or a third party, and may exist on premise or off premise.

Public cloud is a model in which the cloud infrastructure is made available to the general public or a large industry group, and is owned by an organization that sells cloud services.

Hybrid cloud is a composition of two or more clouds (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load-balancing between clouds).

2.2 Application Service Provider

Application service provider (ASP) is a business model, or application delivery model, in which buyers rent access to applications addressing organization-wide needs like accounting, HR, and customer service. In this model, a vendor prices and packages applications and related services to speed up implementation and to minimize the expenses and risks of the entire application life cycle – spanning acquisition, implementation, and ongoing maintenance. In an IDC White Paper, Gillan and McCarty [**8**] define ASP as follows: "ASPs provide a contractual service offering to deploy, host, manage, and rent access to an application from a centrally managed facility. ASPs are responsible for (directly or indirectly) providing all of the specific activities and expertise aimed at managing the software application or set of applications."

The term "application service provider" has originally [**9**] referred to companies meeting a set of defining characteristics. These include the following [**8**]:

1. *Application centric*. The core value of the ASP service is to provide access to and management of an application that is commercially available.
2. *An ASP "sells" the application access*. A part of the value of the ASP services is that customers gain access to a new application environment without making up-front investments in the application license, servers, people, and other resources. The ASP is able to add this value to these services either by owning the software or having a contractual agreement with the software vendor to license access to the software as a part of the ASP's offering.
3. *Centrally managed*. The application service is managed from a central location rather than at each customer's site.
4. *One-to-many service*. The ASP service is designed to be a one-to-many offering.

5. *Delivers on the contract*. There are many partners working together to provide an ASP solution. The ASP is the company that is responsible, in the customer's eyes, for delivering on the customer contract; i.e., seeing that the application service is provided as promised.

Software & Information Industry Association (SIIA) remarks in the report [**9**] that since the term ASP was first invented, many companies who do not meet the criteria have begun to call themselves as ASPs, however. Firms that provide any outsourced application services call themselves an ASP. This causes a lot of confusion in the marketplace.

As discussed later in this thesis (see Section 2.4), the architecture of an ASP application is similar to the architecture of a SaaS application at the first maturity level, i.e., an application that lacks configurability, multi-tenancy, and scalability.

2.3 Characteristics of SaaS

Cheun et al. [**10**] identify six key features of SaaS in their paper. The features are reusability, data managed by provider, service customizability, availability, scalability, and pay per use.

In software engineering, *reusability* denotes that software components can be used again to add new functionalities with minor or no modification. In cloud computing, the fundamental idea is to reuse different types of Internet based services. In SaaS, software itself is a target of reuse and it is deployed to service consumers over the Internet. Typically, one-to-many relationships are used in SaaS services delivery. For example, Google Maps is a service that provides a set of operations to utilize shared information on a map, and the service can be used by various customers.

SaaS is a software deployment model (i.e., all the activities that make a software system available for use) in which a service provider licenses an application to a customer for use as a service on demand. The service provider is responsible for installation and data management of service on their own server. Thus, most of the *data* produced by service

consumers is stored on provider's data center and *managed by the provider* as well. Service consumers do not observe two things in detail: where is their data stored and how can the data be managed. It is essential that the service provider offers data security and reliability function so that service customers would not distrust the services, and thereby would bring down service utilization.

Service customizability means that service consumers can change services based on their individual requirements. This is a characteristic that allows a service provider to meet the different needs of each customer. It is impossible for service providers to customize their cloud services for all service consumers, because in cloud services, various consumers having Internet access can become potential cloud service users. Hence, services must be customized by service consumers for their own purposes. By not providing customizable SaaS services, service providers will limit the usage of SaaS services, because in that case service customers can only utilize the services.

In CC, customers are able to access SaaS services from a web browser over the Internet. In addition, customers do not have any ownership for the SaaS that is deployed and running on the service provider's server. Consequently, many SaaS vendors focus their attention to achieve the highest possible *availability* of services. If a SaaS is not available, customers cannot use the functionality of the SaaS. As an example, Google's mail service, Gmail, was completely down on September 1st, 2009. Because of that, many Gmail users were not able to use the service, i.e. read or write e-mail.

In software engineering, *scalability* means the ability of software to either handle growing amounts of work or to be easily extended. Because of the black-box nature of CC services, service customers cannot control resources, such as memory, network, or CPU utilization, that are utilized by the services. In other words, a service provider is responsible for rescaling the resources based on customer's requests without notifying the customer in detail.

The expenses for SaaS are estimated on the grounds of using services such as the number of service invocations or the duration of service utilization. This is different to traditional business model in which expenses are estimated based on ownership. Service

consumers can connect and use the service as much as they want, and then *pay for* just *the amount of usage* – as a utility like electricity or water. Thus, the customer takes an activity for using the service.

2.4 A Maturity Model for Software as a Service

A SaaS application architecture can be categorized into four maturity levels. The key attributes of a mature SaaS application are scalability, multi-tenant efficiency, and configurability. Scalability stands for maximizing concurrency and utilizing application resources more efficiently than before. Multi-tenancy means an architecture that maximizes the sharing of resources across tenants while still being able to distinguish data belonging to different customers. Configurability, for its part, implies that metadata is being used to configure the behavior and appearance of the application rather than customizing the application in the traditional sense. In the latter case writing custom code to customize the application for one customer would change the application for other customers, too. [**2**]

In the following sections four distinct maturity levels of a SaaS application will be introduced. Each new level adds one of the three attributes listed above. However, an application can meet all necessary business requirements, even if it possesses only one or two of these attributes. In fact, application architects may decide not to fulfill the other attributes, if doing so would not be cost-effective. The maturity levels are presented in Figure 3. [**2**]

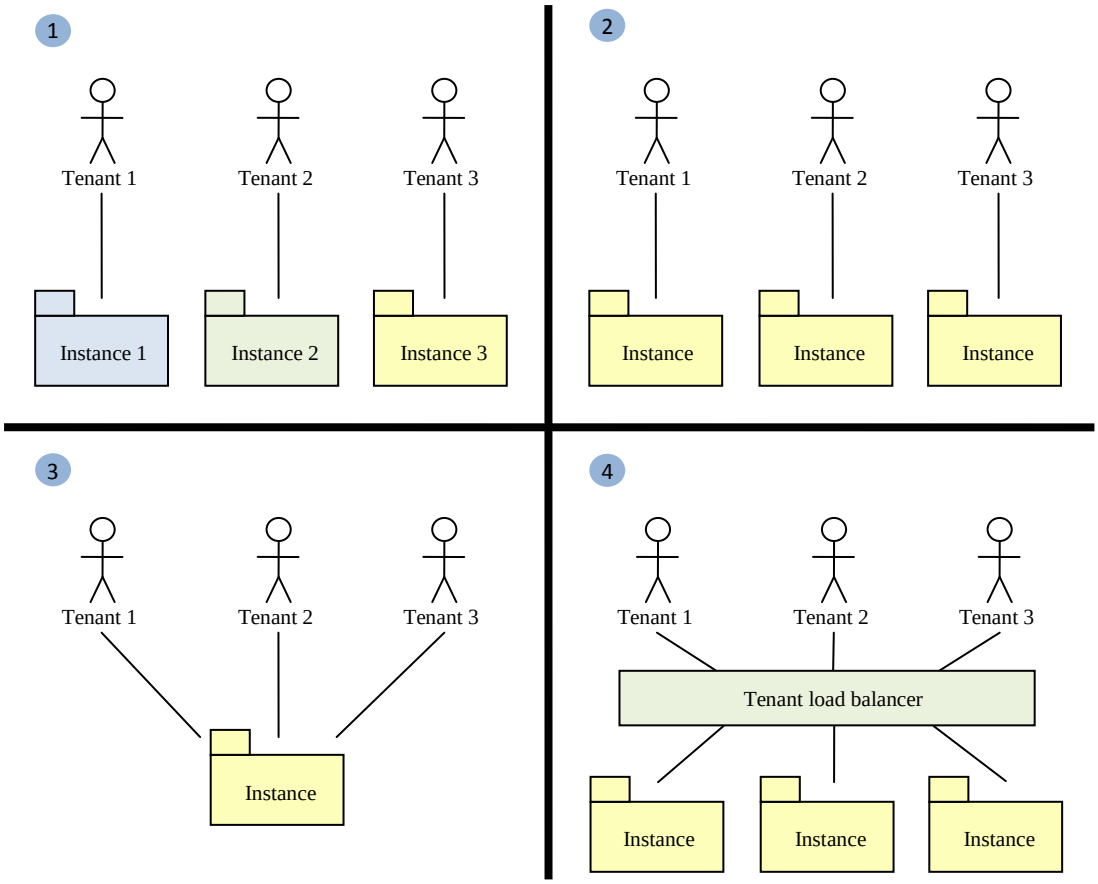


Figure 4. Four-level SaaS maturity model (adapted from [2].)

2.4.1 Level I: Ad-Hoc/Custom

At the first level, each customer has its own customized version of the hosted application, and is running its own instance of the application on the vendor's servers. This is similar to the traditional ASP model (see Section 2.2) from the 1990's. The architecture of software at this level is very much the same as the architecture of traditional software in which different clients connect to a single instance running on the server and instance is totally independent of any other instances or processes that the host is running in order to serve its other customers. Usually traditional client-server applications can be transformed to a SaaS model at the first maturity level with comparatively little development effort, without re-architecting the whole system in depth. This level allows vendors to cut costs by stabilizing server hardware and administration. However, it offers only few of the benefits of a fully mature SaaS system. [**2**]

2.4.2 Level II: Configurable

The second level has the idea that the vendor hosts a separate instance of the application for each customer, or tenant. At this level, all instances use the same code implementation. The vendor provides detailed configuration options that allow the customer to change the layout and behavior of the application. Each instance remains completely isolated from all the others, even though instances are identical to one another at the code level. Single code base for all of a vendor's customers denotes significantly reduced service requirements of a SaaS application, because any changes made to the code base can be easily provided to all of the vendor's customers at once. Both Level I and Level II require that the vendor provides sufficient hardware and storage to manage a potentially large number of application instances running concurrently. [**2**]

2.4.3 Level III: Configurable, Multi-Tenant-Efficient

At the third level, the vendor is running a single instance to serve every customer, and provides a unique user experience and feature set for each one by using configurable metadata. Customers do not see that the application instance is shared among multiple tenants. Authorization and security policies are needed to ensure that each customer's data is kept separate from that of other customers. A benefit of this approach is that there is no need to provide server space for as many instances as there are customers, thus enabling much more efficient use of computing resources than Level II, which basically means lower costs. However, there is a major disadvantage: the scalability of the application is limited. [**2**]

2.4.4 Level IV: Scalable, Configurable, Multi-Tenant-Efficient

At the fourth level, the vendor hosts multiple customers on a load-balanced farm of identical instances. Again, each customer's data is kept separate and with configurable metadata a unique user experience and feature set is provided for each customer. The number of servers and instances on the back-end can be increased or decreased based on demand, without needing additional re-architecting of the application. A SaaS system of this kind is scalable to an arbitrary large number of customers. [**2**]

2.4.5 Choosing a Maturity Level

What maturity level should one target? One might expect that the fourth level is the ultimate goal for any SaaS application, but it is not necessarily so. SaaS maturity can be thought of as a continuum between *isolated* data and code on one end and *shared* data and code on the other, as shown in Figure 4. [**2**]

Figure 5. SaaS maturity as a continuum (adapted from [2].)

Where the application should place itself on this continuum depends on the business, architectural, and operational needs, and on customer considerations.

Business needs signify asking the question "Does an isolated approach make financial sense?" Giving up the economic and management benefits of a shared approach means offering the application to the customer at a higher cost. Nevertheless, it may be worth it to meet other needs. For example, customers may have strong legal or cultural resistance to an architecture in which multiple tenants share access to an application. In the end, a SaaS provider needs a business model that shows how the application can make money regardless of the maturity level targeted.

Architectural needs denote "Can the application be made to run in a single logical instance?" Transforming a desktop-based or traditional client-server application to an Internet-based delivery system may be fundamentally incompatible with a single-instance architecture. Thus, it will probably never make financial sense to invest the development effort needed to transform it into a fully mature SaaS application. When designing and building a net-native application from the ground up, the situation will more likely be better.

Operational needs mean "Can the service level agreements (SLAs) be guaranteed without isolation?" A service provider needs to carefully examine the obligations imposed by any existing SLAs that the provider has with the customer, in regard to

considerations like downtime, support options, and disaster recovery, and determine whether these obligations can be met under the application architecture wherein multiple unrelated customers share access to a single application instance.

2.5 SaaS from the Business Point of View

Prof. Cusumano [**11**] from MIT Sloan School of Management deliberates a shift from software products to software services in his article. He states that over the past few years traditional product sales and license fees have declined and product company revenues have shifted to services. The transformation has been particularly seen among enterprise software vendors. There are some exceptions though. For example Microsoft continues to generate massive revenues from product sales. However, even Microsoft is encountering a change. A few years ago it derived all its revenues from products, but in fiscal year 2007 services in the server and tools segment accounted for about 3 percent of the company's revenues and online services for 5 percent of the revenues. In March 2010, Steve Ballmer [**12**] – CEO of Microsoft – made a strong statement by proclaiming "As I like to say at Microsoft, for the cloud, we're all in." He also said that "About 70 percent of our folks are doing things that are entirely cloud based or cloud inspired. And by a year from now, that'll be 90 percent." So Microsoft is betting the company's future on cloud computing.

Cusumano [**11**] also discusses that SaaS and "free, but not free" software as new business and pricing models have confused the traditional separation of product and service revenues as well as costs. The SaaS pricing model eliminates maintenance payments (a source of service revenues for software companies) and possibly includes bundled technical support (a source of costs). Companies like Google, Microsoft, and Yahoo! are delivering what used to be free products as a nominally free service, in which the user does not directly pay for the software. Instead, advertisers pay the software service vendor.

According to Cusumano, software product companies begin generating most of their revenues from product license fees, but later shift to a combination of products and services and eventually to mainly services. Because the marginal cost is zero to copy a

piece of software and firms can generate up to 99 percent gross margins, they may still want to focus on products. On the contrary, margins for IT services can be less than 30 percent. Cusumano also says that when competitors appear, the great profit opportunity from software products becomes theoretical and not practical. This means that software product companies' revenues gradually shift to services.

Cusumano also presents some preliminary findings based on a research project he launched at MIT in 2003 to examine the shift from products to services. A decade ago, almost all software product companies used the up-front license fee to sell software. Installations were done locally on the customers' hardware. Today there are many different business models: subscription, advertising-based, transaction-based, and several kinds of "free, but not free". Delivery models can be remote and web-based or bundled as hardware products, when it is less likely that the price will fall to zero. In one scenario, according to Cusumano, software prices will fall to zero or near zero and the future is really free software, inexpensive SaaS, or "free, but not free" software with some kind of indirect pricing model, e.g. advertising – a model used by Google.

Cusumano's research project includes a database of 500 publicly listed software product companies. The database contains about 10 years of data for each company. The data (excluding game-software firms and some other firms) shows that product revenues have dropped but have not continued to fall to zero. They have more like stabilized at about 50 percent of total revenues. Cusumano designates this as equilibrium point as a business. I.e., service (incl. maintenance) revenues are greater from their existing customers than new-product revenues, while products' value is still significant. Cusumano's and his colleagues' preliminary analysis also indicates that the optimal mix for operating profitability (excluding games and some other firms) seems to be something like 70 percent products and 30 percent services.

Cusumano's analysis yet shows that there appears to be two reasons for the transition toward services for product companies. The first one is that product sales might continue growing, but services grow faster, maybe because price level or the number of new customers falls. The other scenario is that the products business collapses, and therefore a firm crosses over to a majority of service revenues.

One general conclusion that Cusumano makes is that many or most software product companies can and should take advantage of services, particularly maintenance. Companies should not just let services "happen" because their product business is declining. In other words, software product firms should treat services as a strategic area and a target of opportunity to increase revenues and profits. Too many product companies seem to think that services are a necessary evil and manage them as a cost center.

In his article, Cusumano finally presents three challenges for those who are experiencing this shift toward services. First, managers need to identify the best mixture of product revenues for their particular business segments together with service and maintenance revenues and determine how to impact these percentages. A point worth remembering is that for most product companies, products are the engine that drives service and maintenance revenues. Even though there are some exceptions, products and services are coupled for most companies. Second, managers must think how to create service offerings that add value and distinctiveness to their products, e.g. by wrapping services around products. Third, managers should think about how they can deliver services more efficiently. Productization of services can mean component or design reuse, computer-aided tools, or standardized process frameworks and training.

2.6 Implementation Platforms

When an application runs in the cloud, uses services provided by the cloud, or both, some kind of application platform is needed. In this section, a few topical implementation platforms are introduced.

2.6.1 Google App Engine

The Google App Engine is a platform that makes it possible to build and host web applications on Google infrastructure – on the same systems that power Google applications. An application can be hosted on a Google Apps (See Section 3.1) domain (like http://www.example.com), or it can be hosted on the appspot.com domain with a

free name. Access to the application can be limited or it can be allowed to everyone in the world. [**13**]

The Google App Engine supports two programming environments: the Java environment and the Python environment. The Java runtime environment includes support for the Java Virtual Machine (JVM), Java Servlet, and the Java programming language – or any language using a JVM-based interpreter or compiler. The Python runtime environment includes a fast Python interpreter and the Python standard library. Each environment offers standard protocols and common technologies for web application development. [**13**]

The App Engine contains the following features: 1) dynamic web serving with support for common web technologies 2) persistent storage with queries, sorting and transactions 3) automatic scaling and load balancing 4) application programming interfaces (APIs) for authenticating users and sending e-mail using Google Accounts 5) local development environment that simulates the Google App Engine 6) task queues for performing work outside of the scope of a web request, and 7) scheduled tasks for triggering events at specified times and regular intervals. [**13**]

Applications run in a secure environment, "sandbox", which provides limited access to the underlying operating system. Because of these limitations the App Engine can distribute web requests for the application across multiple servers, and so start and stop servers to correspond traffic demands. The sandbox isolates an application in its own reliable environment that is independent of the hardware, operating system and geographical location of the web server. For instance, an application can only access other computers on the Internet through the provided URL fetch API and e-mail services. Other computers, for one, can only connect to the application by making HTTP or HTTPS requests on the standard ports. Another example of the limitations would be that an application cannot write to the file system. It can read files, but only those that are uploaded with the application code. The application must use the Datastore, the Memcache, or other services of the App Engine for the data that persists between requests. A third example is related to code execution; application code can only run in response to a web request, a queued task, or a scheduled task. In addition, it

must return response data within 30 seconds. A request handler is not allowed to spawn sub-processes or execute code after the response has been sent. [**13**]

Development for the Java runtime environment is possible using common Java web development tools and API standards. The application will interact with the environment using the Java Servlet standard. The Java runtime environment of the App Engine uses Java 6. I.e., it includes the Java SE Runtime Environment (JRE) 6 platform and libraries. The App Engine Java SDK has support for Java 5 and 6. The restrictions of the sandbox environment mentioned earlier are implemented in the JVM. For example, Java bytecode that tries to open a socket or write to a file will cause a runtime exception. Most App Engine services are accessed using Java standard APIs, which include the Java Data Objects (JDO) API, the Java Persistence API (JPA), the JavaMail API, and the java.net HTTP APIs. In addition, the App Engine contains the following low-level APIs for its services: the Datastore Java API, the Memcache Java API, the URL Fetch Java API, the Mail Java API, the Images Java API, and the Users Java API. Besides using the Java programming language and APIs to implement web applications for the JVM, one can use other languages, such as JavaScript, Ruby, or Scala, when using a JVM-compatible compiler or interpreter. [**13**]

Applications for the Python runtime environment can be implemented using the Python programming language, and the application can be run on an optimized Python interpreter. The App Engine offers APIs and tools for Python web application development. These include a feature rich data modeling API, a web application framework, and tools for managing and accessing data of the application. It is also possible to utilize mature libraries and frameworks, such as Django. The Python runtime environment uses Python version 2.5.2 at the moment, but additional support for Python 3 might be coming in the future. The Python environment contains the Python standard library, but not all of the library's features are allowed to be run in the sandboxed environment. Many modules in the standard library whose core features are not supported by the runtime environment are disabled, and code that imports them will throw an error. Application code for the Python environment shall be exclusively written in Python language. There is no support for extensions that are written in the C language. The Python environment provides the Python Datastore API, the URL Fetch

Python API, the Mail Python API, and the Users Python API. The App Engine also contains a simple Python web application framework called webapp. When developing for the Python environment, other third-party libraries are also supported, as long as they are implemented in pure Python. They must not require any unsupported standard library modules, either. [**13**]

The App Engine offers a distributed data storage service that includes a query engine and transactions. The distributed data storage grows with user's data. The App Engine provides two data storage options with different reliability and consistency guarantees. The Master/Slave Datastore is used by default for new applications, and it causes the lowest storage and CPU costs for storing data. This option is a master-slave replication system, which asynchronously replicates data when a user writes it to another datacenter. This option provides strong consistency for all reading and querying, because only one datacenter is the master for writing at any time. However, user's data may be temporarily unavailable during planned or unplanned datacenter downtime. The other option for data storage is the High Replication Datastore, which is a new, highly available, and highly reliable storage solution. It is available for reading and writing during planned downtime and is recommended mainly for those developers who are building mission critical applications on the App Engine. The High Replication Datastore is about three times more expensive than the Master/Slave option. Data is replicated across datacenters using a system based on the Paxos consensus algorithm. The App Engine Datastore is not like a traditional relational database. Data objects, or "entities", have a type and a set of properties. Queries can retrieve entities of a given type. The values of the properties are used to filter and sort the results. Datastore entities do not have a schema. The application code defines the structure of data entities. Applying and enforcing structure within an application is done through the Java JDO/JPA or the Python Datastore interfaces. [**13**]

The Google App Engine supports integrating an application with Google Accounts, which means that the application can allow a user to sign in with a Google account. Then the user does not necessarily need to create a new account, when he starts using the application. For developers it means that they do not have to implement a user account system for the application. [**13**]

The App Engine offers a set of services to perform common operations when managing the application. The URL Fetch API enables applications to access resources on the Internet, such as web services or other data. The Mail API enables applications to send e-mail messages. The Memcache API offers the application a high performance in-memory, key-value cache that can be accessed by multiple instances of the same application. The Memcache service is useful for temporary data or data copied from the datastore to the cache for performance reasons. The Images API enables the application to process images. Images can be e.g. resized, cropped, rotated, and flipped in JPEG and PNG formats. [**13**]

A Google App Engine application can perform tasks outside of responding to web requests. The application can perform these tasks scheduled by a developer or it can perform tasks added to the queue by the application itself. Scheduled tasks are called "cron jobs" and they are handled by the Cron service. At the moment task queues are an experimental feature and only available with the Python runtime environment. [**13**]

With the App Engine, there are no set-up costs and no recurring fees. You only pay for what you use. The billing is done so that the resources your application uses, such as storage and bandwidth, are measured by the gigabyte. You can control the maximum amount of resources that the application can consume. All applications may use up to 500 MB of storage and enough CPU time and bandwidth for serving around 5 million page views a month, for free. After enabling billing for your application, the free limits are raised, and then you only pay for the resources that are used above the free levels. [**13**]

2.6.2 Windows Azure

Microsoft offers Windows Azure [**14**] which is a cloud services operating system that serves as the development, service hosting and service management environment for the Windows Azure platform. I.e., Windows Azure is part of the larger Windows Azure platform [**15**].

The Windows Azure platform provides a group of cloud technologies, each providing a specific set of services to software developers. The Windows Azure platform supports applications, data, and infrastructure in the cloud. In addition, it includes a cloud marketplace. The platform today [**16**] (November 2010) has four parts: 1) Windows Azure, 2) SQL Azure, 3) Windows Azure AppFabric, and 4) Windows Azure Marketplace. The components are illustrated in Figure 7.

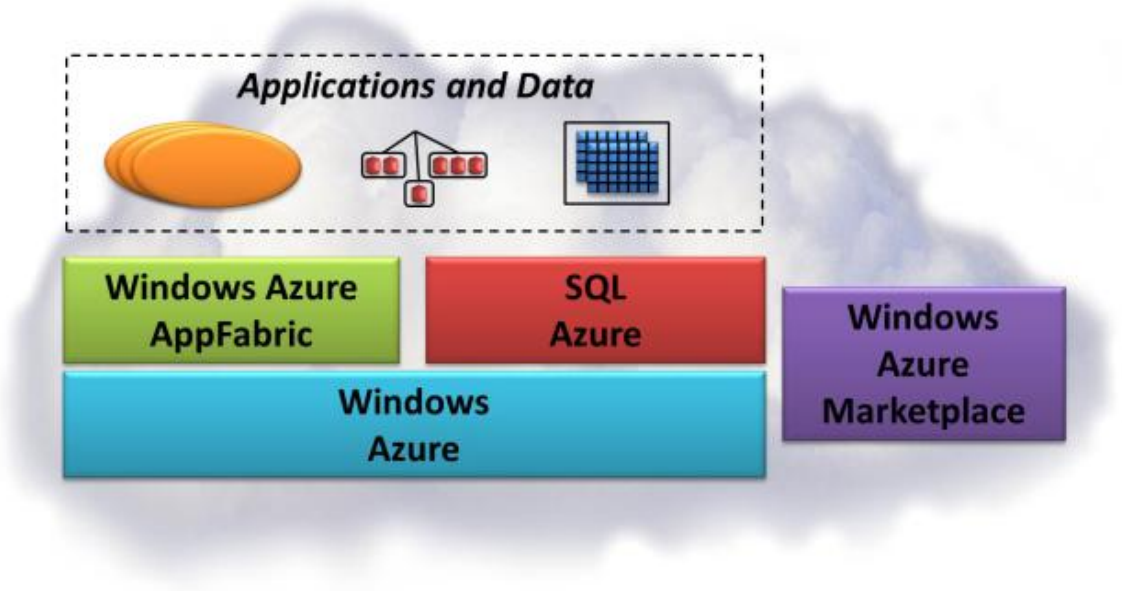


Figure 6. The Windows Azure platform components [16**].**

Windows Azure is a Windows environment for running applications and storing data. SQL Azure provides relational database services in the cloud based on Microsoft SQL Server. Windows Azure AppFabric provides cloud-based infrastructure services for applications that run in the cloud or on premises. Windows Azure Marketplace is an online shopping service for cloud-based data and applications. All the four components are running in Microsoft data centers located around the world (two in North America, two in Europe, and two in Asia). Developers are able to control which data center runs their applications and stores their data. [**16**]

Each four of the Windows Azure platform parts has its own role. Next, the parts are described at a high level.

Windows Azure simply runs Windows applications and stores data in the cloud. Its components are shown in Figure 8. [**16**]

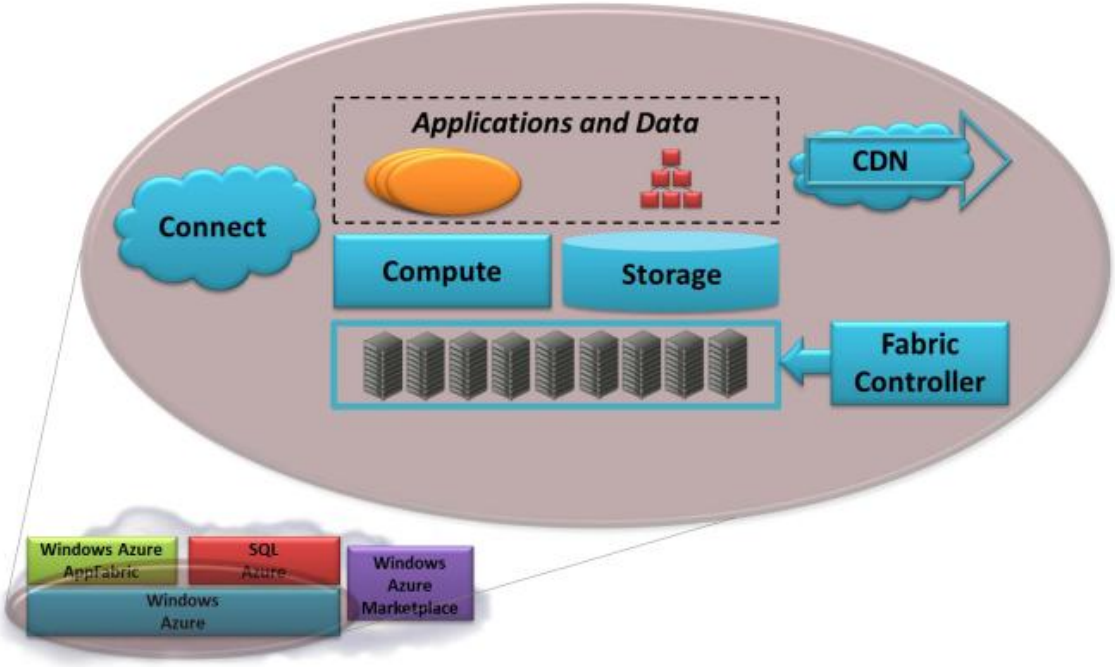


Figure 7. Windows Azure components [16**].**

Windows Azure today consists of five parts. *Compute* is a service that runs applications on a Windows Server foundation. These applications can be built using the .NET Framework (in C# or Visual Basic language) or without .NET in C++, Java, and other languages. *Storage* is a service that allows storing binary large objects, called blobs. It also offers communication queues between components of Windows Azure applications. Moreover, it provides a form of tables with a simple query language. Both Windows Azure applications and on-premises applications are able to access the Windows Azure storage service. *Fabric Controller* is responsible for knitting the machines in a single Windows Azure data center to form a cohesive whole. After this, the Windows Azure compute and storage services are built on top of this pool of processing power. *Content delivery network (CDN)* caches frequently accessed data closer to its users in order to speed up access to that data. The caching is done for blobs so that cached copies are maintained at sites around the world. *Connect* makes it easier for a Windows Azure application to access an on-premises database, as if both were inside the organization's own firewall. [**16**]

Windows Azure customers can use a browser-based portal to create, configure, and monitor applications. A customer logs in with a Windows Live ID and then selects whether to create a hosting account for running applications, a storage account for storing data, or both. Microsoft charges the customer based on how much computing time, storage, and bandwidth that customer uses. People who create an application may decide how that application charges its own customers – if they are charged at all. [**16**]

Windows Azure is a platform that can be used in a wide set of scenarios. In one scenario, an independent software vendor (ISV) needs to create a SaaS version of an existing on-premises Windows application. The ISV might choose to build it on Windows Azure. Transforming the application's business logic to this cloud platform is typically not too problematic, because Windows Azure mostly provides a standard Windows environment. Building on an existing platform allows the ISV focus on the important part (business logic), rather than spending time on infrastructure. In another scenario, an enterprise needs to create an application for its customers or employees. The enterprise might choose to build the application on Windows Azure, because it supports .NET and developers with proper skills are easy to find. The enterprise does not need to worry about managing its own servers when the application is running in Microsoft data centers. A third example scenario could be a start-up creating a new web site. The start-up might build its application on Windows Azure. The application is able to offer an interactive user interface and executing work for users asynchronously, because both web-facing services and background processes are supported in Windows Azure. [**16**]

SQL Azure offers cloud-based services for relational data. SQL Azure at the moment includes three components as presented in Figure 9. *SQL Azure Database* provides a cloud-based database management system (DBMS). Both on-premises and cloud applications can store relational data on Microsoft servers in Microsoft data centers. *SQL Azure Reporting* is a version of SQL Server Reporting Services (SSRS) that is running in the cloud. It is intended mainly for use with SQL Azure Database and it allows creating and publishing standard SSRS reports on cloud data. *SQL Azure Data Sync* is used to synchronize data between SQL Azure Database and on-premises SQL

Server databases. In addition, it allows synchronizing data across different SQL Azure databases in different Microsoft data centers. [**16**]

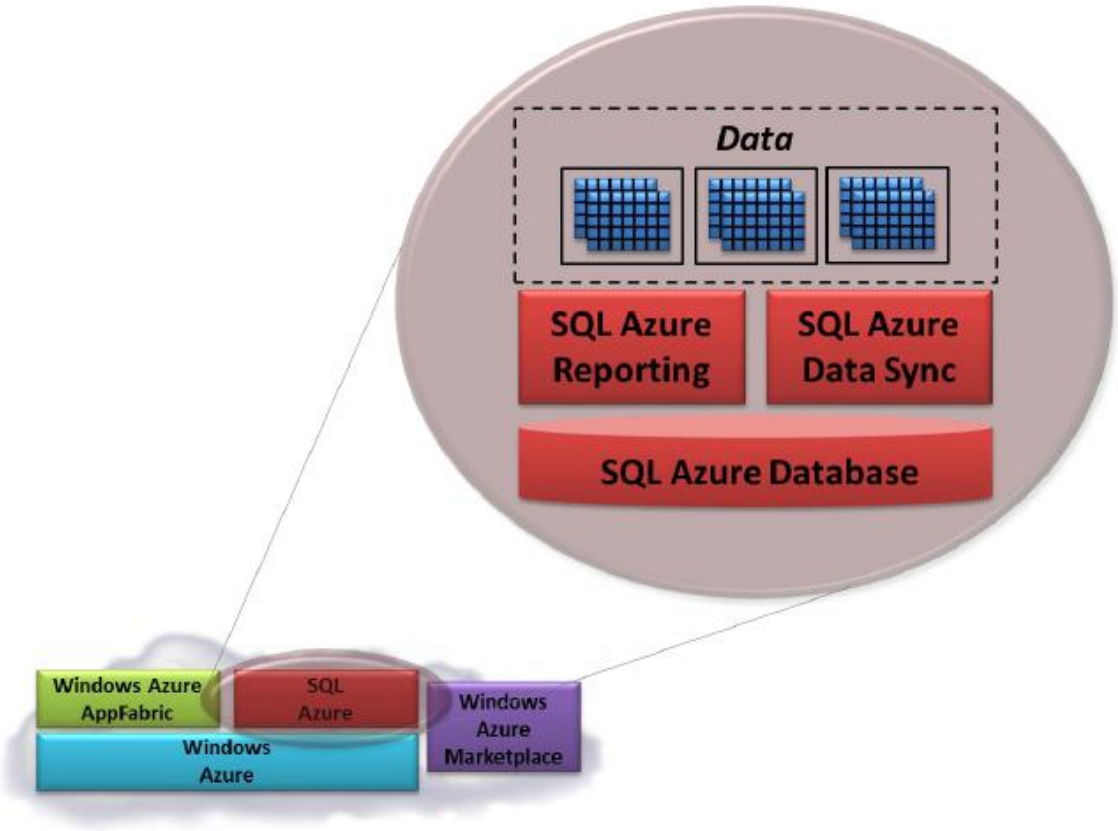


Figure 8. SQL Azure components [16**].**

SQL Azure is built on Microsoft SQL Server. Applications accessing SQL Server locally will mostly work unchanged with data in SQL Azure. SQL Azure customers can focus on their data, because operational details are handled by Microsoft. Customer's services are accessed through the common Windows Azure platform portal. [**16**]

Windows Azure AppFabric provides cloud-based infrastructure services. Its functions address common challenges in building distributed applications. The components of Windows Azure AppFabric are illustrated in Figure 10.

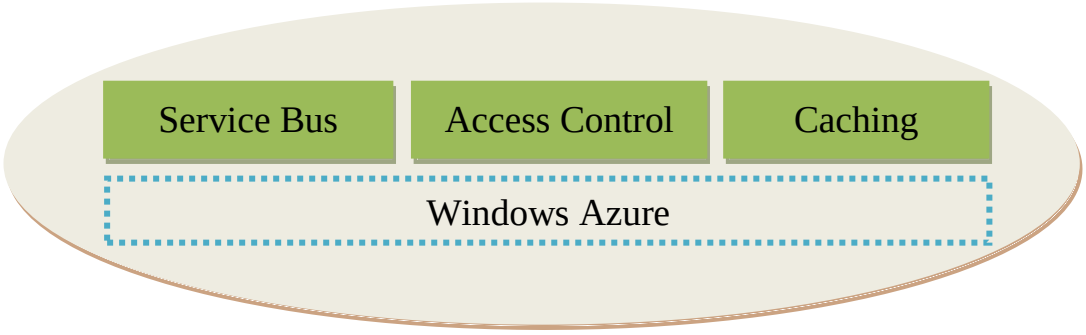


Figure 9. Windows Azure AppFabric components (adopted from [16].)

Windows Azure AppFabric components are all built on Windows Azure. However, they do not all provide services exclusively to Windows Azure applications. *Service Bus* aims at exposing an application's services on the Internet. It lets the application expose endpoints in the cloud. The endpoints can be accessed by other applications, both on-premises and in the cloud. Each endpoint is assigned a URI that clients can use to locate and access the service. Furthermore, Service Bus handles the challenges that are faced when dealing with network address translation (NAT) and getting through firewalls without opening new ports for exposed applications. *Access Control* provides a built-in support for several login options (such as Active Directory, Google Accounts, Windows Live ID etc.). It also offers a centralized rule defining to control what each user is allowed to access. *Caching* service provides one way to speed up an application that accesses the same data over and over. Frequently accessed information is cached, reducing the number of database queries from the application. Windows Azure AppFabric should not be confused with the Windows Azure fabric controller. They are completely separate technologies and address quite distinct problems. [**16**]

Windows Azure Marketplace provides an online marketplace for cloud applications and cloud-accessible data. It includes two components: Windows Azure Marketplace DataMarket and Windows Azure Marketplace AppMarket. *DataMarket* enables content providers to make datasets available. It is possible to browse the offerings and then purchase whatever seems to be useful. Applications can then access this data through RESTful[1] requests or the OData[2] protocol. *AppMarket* provides a way for creators of

[1] Representational State Transfer (REST) defines a collection of architectural principles by which one can design web services that focus on a system's resources. Conforming to the REST is referred to as being

cloud applications to expose those applications to potential customers. AppMarket will be available in Beta by the end of calendar year 2010. [**16**], [**14**]

What follows is a little bit more detailed description of the Compute and Storage services. When an application is built on the Windows Azure Compute service, it is composed of one or more roles. The application usually runs, when it is executed, two or more instances of each role so that each instance is running as its own virtual machine (VM). The three kinds of roles today are the following:

Web roles are meant mainly for running web-based applications. Each Web role instance includes a pre-configured Internet Information Services (IIS) 7, so developers can create applications using e.g. ASP.NET, Windows Communication Foundation (WCF), PHP, Java, or other technologies. *Worker roles* are intended to run a variety of code. A Worker role could run a simulation or video processing, for example. It is common that an application interacts with a user through a Web role and processing tasks are passed to a Worker role. *VM roles* are capable of running a Windows Server 2008 R2 image provided by the user. A VM role might be the right choice when moving some on-premises Windows Server applications to Windows Azure. When a Windows Azure application is run, configuration information is provided along with it. The information tells the platform how many instances of each role to run. Then the Windows Azure Fabric Controller creates a VM for each instance running the code for the appropriate role in each VM. Requests from outside are load balanced across all instances of a role. [**16**]

The Windows Azure Storage service provides three options for persistent storage. The simplest way to store data in Windows Azure Storage is to use *Blobs*. A storage account can consist of one or more containers, each of which is holding one or more Blobs. Blobs can be large, up to 1 TB each. They can be subdivided into blocks, which makes transferring large Blobs more efficient. In case of a failure it is not necessary to send the entire Blob again. Blobs can also be used through Windows Azure drives that can be mounted by a role instance. The underlying storage for a drive is a Blob. Once a drive is

---

‘RESTful’. https://www.ibm.com/developerworks/webservices/library/ws-restful/

[2] Open Data Protocol (OData) is a web protocol for querying and updating data. http://www.odata.org/

mounted, the instance is able to read and write file system data that gets stored persistently in a Blob. In order to allow applications to work with data in a more structured way, Windows Azure Storage offers *Tables*. In this context these are not relational tables, but the data that Tables contain is actually stored in a set of entities with properties. A Table does not have a defined schema, but properties can have various types like int, string, Bool, or DateTime. An application can access data in a Table using the simple query language defined by OData. A Table can contain billions of entities holding terabytes of data. The service can partition it across many servers to improve performance. The third option in Windows Azure Storage is *Queues*. Queues are primarily intended to let Web role instances communicate with Worker role instances. A user can submit a request to perform a compute-intensive task on a web page implemented by a Windows Azure Web role. The Web role instance receiving this request can write a message into a queue to describe the work to be done. A Worker role instance, which is waiting on this queue can read the message and perform the task it specifies. Results can be returned via another queue, for instance. Windows Azure Storage (Blobs, Tables, and Queues) is accessed in a RESTful style via HTTP, HTTPS, or OData (for Tables). Because everything is named using URIs and accessed with standard HTTP operations, clients can be created using .NET Framework, Java, or some other technology. [**16**]

Windows Azure provides what is commonly called Platform as a Service (See Section 2.1.2). It offers a service that is meant to make life easier for developers and administrators. Windows Azure platform has been commercially available since February 2010. [**16**]

2.6.3 Amazon Web Services

Amazon Web Services (AWS) [**17**] is a broad cloud services platform, which offers compute power, storage, content delivery, and other functionality that enables businesses to deploy applications and services – according to Amazon – with flexibility, scalability, reliability, and cost-effectiveness. AWS today [**18**] (December 2010) covers products and services from the following areas: compute, content delivery, database, e-commerce, messaging, monitoring, networking, payments & billing, storage, support,

web traffic, and workforce. The products and services are listed by groups in Figure 11. Next, some of these services are discussed at a high level.

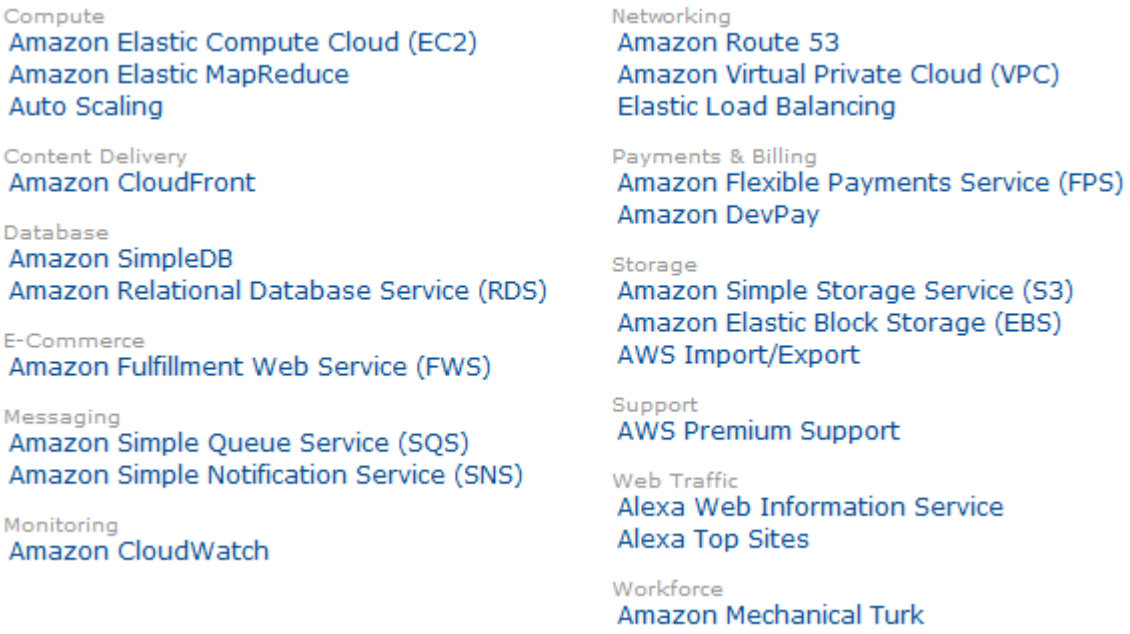


Figure 10. Products and services of AWS [18**].**

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that offers resizable computing capacity in the cloud. It is designed to make web-scale computing easier for developers. It includes a simple web service interface that allows a user to obtain and configure capacity easily. The user is provided with complete control of his computing resources on Amazon's computing environment. In Amazon EC2 the time required to obtain and boot new server instances is some minutes. This allows the user to quickly scale capacity, both up and down, as computing requirements change. Amazon EC2 also incorporates the tools for developers to build failure resilient applications and isolate themselves from common failure scenarios. [**17**]

Amazon EC2 makes it possible to place instances in multiple locations. The locations are constructed from Availability Zones and Regions. Regions are distributed and located in separate geographic areas, whereas Availability Zones are distinct locations inside a Region. It is possible to design an application to be closer to specific customers or to meet legal requirements, for instance, by launching instances in separate Regions. By launching instances in separate Availability Zones, it is possible to protect an

application from the failure of a single location. Figure 12 shows a representation of Amazon EC2.

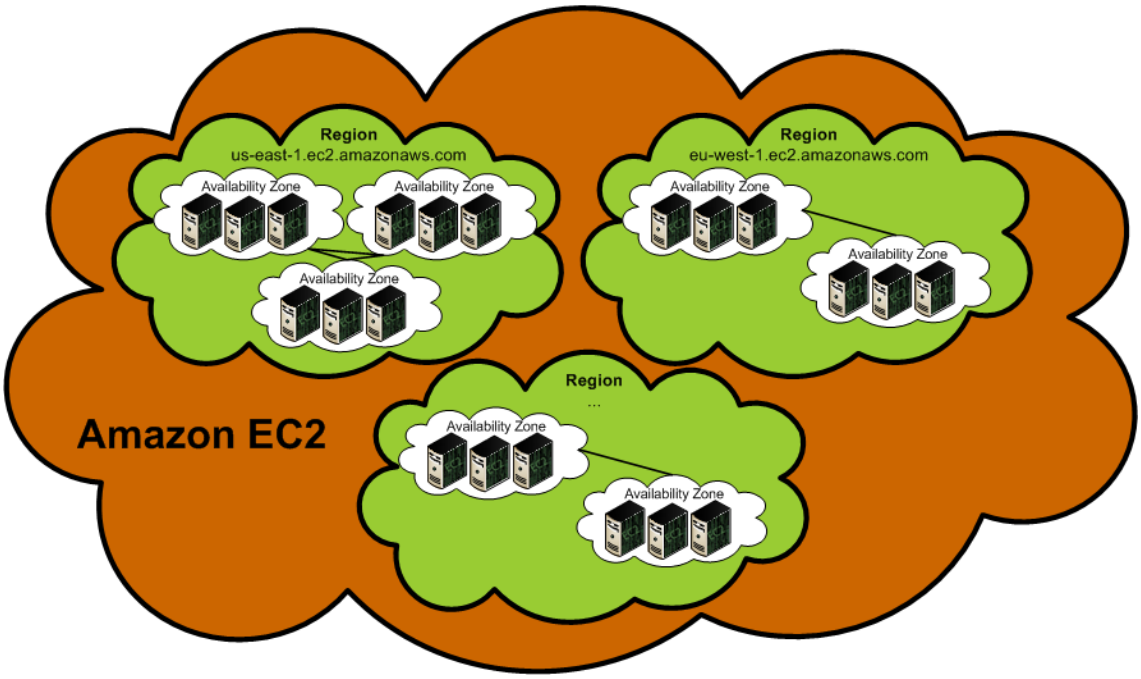


Figure 11. Amazon EC2 locations. Each Region is completely independent and each Availability Zone is isolated, but connected through low-latency links [19**].**

Amazon Simple Storage Service (Amazon S3) is storage in the cloud. It provides a web services interface to store and retrieve any amount of data, at any time, from anywhere on the web. Amazon S3 gives developers access to the same data storage infrastructure that Amazon uses to run its own network of web sites. Amazon S3 includes the following features. It is possible to write, read, and delete objects (containing from 1 byte to 5 GB of data each) and the number of objects that can be stored is unlimited. An object is stored in a bucket and retrieved via a unique key. It is possible to choose a region where a bucket is stored. E.g. objects stored in the EU region never leave the EU unless the user transfers them out. Data is secured with authentication mechanisms and rights can be granted to specific users. Standards-based REST and SOAP interfaces are used to enable support for any Internet-development toolkit. The default download protocol is HTTP, but protocol or functional layers can be added. Monthly uptime percentage is at least 99.9 %. [**17**], [**18**]

Amazon Elastic Block Store (Amazon EBS) offers block level storage volumes for use with Amazon EC2 instances. Amazon EBS volumes persist independently from the life of an instance. Storage volumes can be created from 1 GB to 1 TB and they can be mounted as devices by Amazon EC2 instances. It is possible to mount multiple volumes to the same instance. Storage volumes work like raw, unformatted block devices. One can create a file system on top of Amazon EBS volumes, or use them in any other way one would use a block device, like a hard drive. Amazon EBS volumes are placed in a specific Availability Zone. Each storage volume is automatically replicated inside the same Availability Zone, which prevents data loss in case of failure of any single hardware component. [**18**]

Amazon Virtual Private Cloud (Amazon VPC) is a secure bridge between a company's existing IT infrastructure and the AWS cloud. With Amazon VPC, enterprises can connect their existing infrastructure to a set of isolated AWS compute resources via a virtual private network (VPN) connection. Existing management capabilities (like security services, firewalls, and intrusion detection systems) can be extended to include AWS resources. Amazon VPC integrates at the moment with Amazon EC2, and will integrate with other AWS services in the future. [**17**]

Amazon CloudFront is a web service for content delivery. It integrates with other AWS services so that developers and businesses can easily distribute content to end users with low latency, high data transfer speed, and no commitments. In Amazon CloudFront, static and streaming content is delivered using a global network of edge locations. Object requests are automatically routed to the nearest location. Thus content is delivered with the best possible performance. Amazon CloudFront is optimized to work with other AWS. [**17**], [**18**]

Amazon Relational Database Service (Amazon RDS) enables easy set-up, operating, and scaling of a relational database in the cloud. It provides resizable capacity while performing time-consuming database administration tasks. Amazon RDS provides access to the full capabilities of a MySQL database. The code, applications, and tools already used with existing MySQL databases work with Amazon RDS as well. Users do not have to take care of database software patches, database backups, or storing the

backups. These actions are done automatically by Amazon RDS. The computing resources or storage capacity associated with the relational database instance can be scaled via a single API call. [**17**]

Amazon SimpleDB provides the core database functions of data indexing and querying in the cloud. It is a non-relational data store that offloads the work of database administration. Developers store and query data items via web services requests, and Amazon SimpleDB takes care of the rest. It does not require a schema, and automatically indexes the data and provides a simple API for storage and access. [**17**], [**18**]

Amazon Simple Queue Service (Amazon SQS) is a scalable, hosted queue for storing messages as they travel between computers. Developers can move data between distributed components of their applications, without losing messages or requiring each component to be always available. Amazon SQS works in conjunction with Amazon EC2 and the other AWS and it is possible to build an automated workflow. Amazon SQS exposes Amazon's web-scale messaging infrastructure as a web service, and any computer (on the Internet) can add or read messages without installed software or special firewall configurations. [**18**]

Amazon Simple Notification Service (Amazon SNS) is a web service that enables easy set-up, operating, and sending of notifications from the cloud. Developers are provided with a capability to publish messages from an application and immediately deliver them to subscribers or other applications. Amazon SNS can be used to create topics, subscribe clients to these topics, publish messages, and have the messages delivered over clients' protocol of choise (such as HTTP, e-mail, etc.). Notifications to clients are delivered using a "push" mechanism, which eliminates the need of periodical checks or "polls" for new information and updates. [**17**]

Amazon Elastic MapReduce is a web service for businesses, researchers, data analysts, and developers. It enables easy processing of vast amounts of data. Amazon Elastic MapReduce utilizes a hosted Hadoop framework that runs on the web-scale infrastructure of Amazon EC2 and Amazon S3. With Amazon Elastic MapReduce, it is

possible to perform data-intensive tasks for applications such as web indexing, data mining, long life analysis, data warehousing, machine learning, financial analysis, scientific simulation, and bioinformatics research. [**18**]

In contrast to Windows Azure, Amazon Web Services is a suite of "Infrastructure as a Service". Amazon EC2, for example, provides a host for virtual machines, and the user is responsible for setting up the virtual machines and applications that are run in them. Amazon Web Services was launched in July 2002 and it has grown into a significant cloud services provider.

3 SAAS IMPLEMENTATIONS AND TRANSFORMATIONS

The first realization of SaaS to be studied in this chapter, in Section 3.1, is Google Apps, a commercial service from Google. The second case, AzureBlast, is not a typical business application, but a science application, or an evaluation of the platform's suitability for scientific use. The SaaS implementation is built on the Windows Azure platform and is covered in Section 3.2. In Section 3.3, an implementation of a SaaS framework and a SaaS application built on that framework is discussed. After these three implementations, a case study on a Finnish software company is covered in Section 3.4. In this case study, the researchers examine a transition from software products business to software services business at F-Secure. In addition, a new Internet-based online backup and restore solution by F-Secure is introduced at the end of Section 3.4. Finally, Microsoft .NET Framework is discussed in Section 3.5 as it seems to have an important role in Microsoft's cloud strategy.

3.1 Google Apps

3.1.1 Overview

Currently, Google Apps include Google Apps for Business, Google Apps for Education, and Google Apps (Free). In this context, the first one will be discussed. According to Google, more than two million businesses run Google Apps. For example Motorola Mobile Devices Division, Salesforce.com, and City of Los Angeles are Google Apps customers. [**20**]

Google Apps for Business, formerly Google Apps Premier Edition, offers among others Google's web-based applications without hardware or software installation, mobile e-mail and calendar synchronization, guaranteed uptime availability of 99.9 %, 25 GB of e-mail storage per employee, data security, and 24/7 customer support. With Google Apps, the data and the applications are served from Google's own data centers (build by Google from scratch). An overview of Google Apps is presented in Figure 6. [**20**]

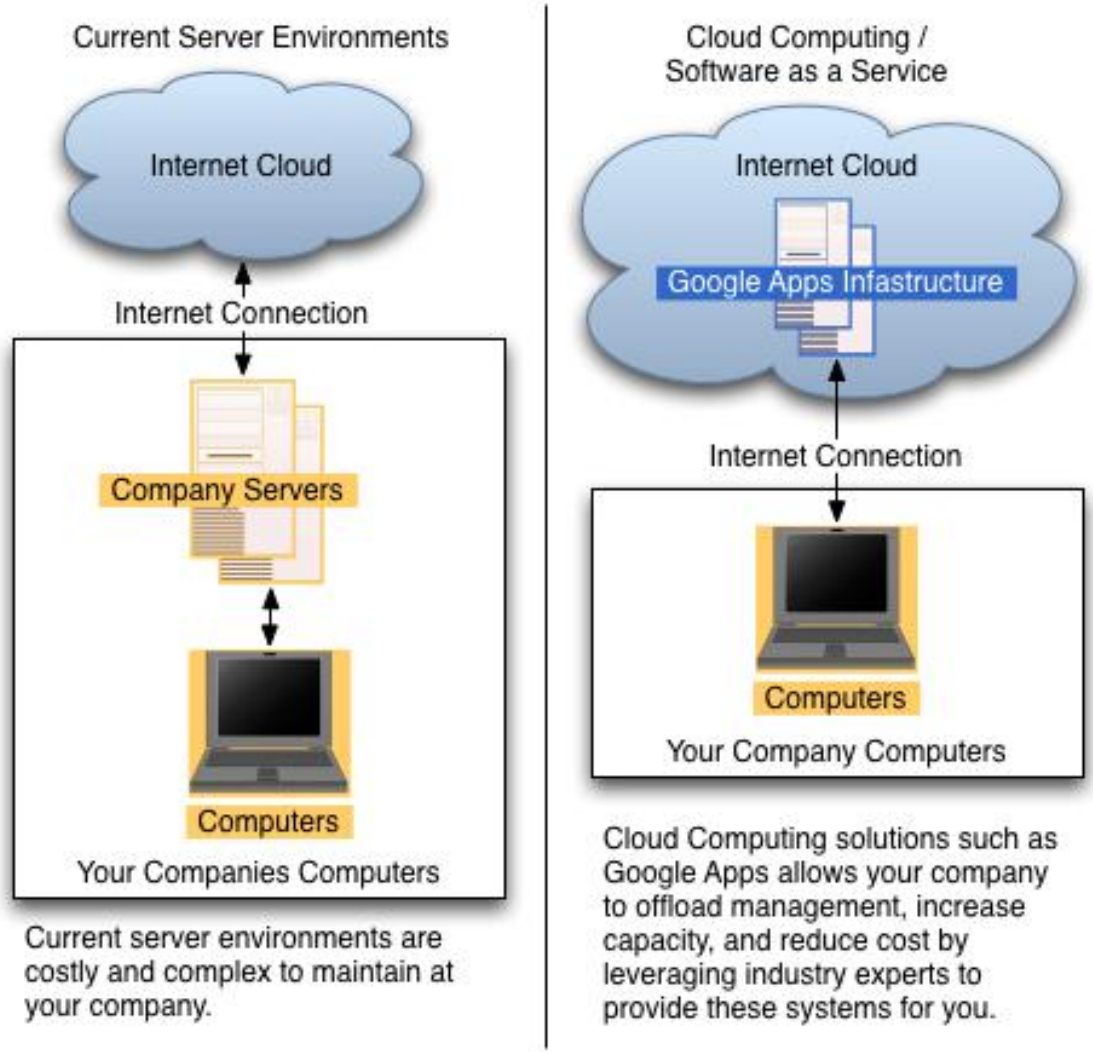


Figure 12. General idea of Google Apps [13].

At the moment (October 2010), Google Apps for Business (later Google Apps) includes the following messaging and collaboration applications: Gmail for business, Google Talk, Google Calendar, Google Docs, Google Groups, Google Sites, Google Video, and Google Wave. *Gmail for business* offers e-mail, instant messages (IM), voice and video chat, Microsoft Outlook and BlackBerry interoperability, Google-powered e-mail search, and a customizable spam filtering by Postini services. *Google Talk* covers a video and voice plug-in for a web browser to chat within Gmail, iGoogle[3], and orkut[4], and a downloadable application to chat from PC's desktop. In addition to IM, the plug-

[3] iGoogle is a customizable homepage that contains a Google search box and any number of gadgets of user's choice. Gadgets may include e.g. headlines from news sources, weather forecasts, or latest Gmail messages.
[4] orkut is an online community designed to make user's social life more active and stimulating.

in enables computer to computer voice and video calls. It works on Windows, Mac OS X, and Linux in the supported browsers. The Google Talk software makes possible text chat with IM, status updates, and file transfer. It also offers voice chat with free long distance PC-to-PC calls, audio conferencing, and Gmail integration. *Google Calendar* offers scheduled appointments, integration into Gmail and interoperability with popular calendar applications, sharing of project calendars with sharing permission controls, access with a mobile device, and publishing a calendar. *Google Docs* includes online documents with real-time collaboration. It works in a browser across operating systems (Windows, Mac, Linux), and supports popular formats such as .doc, .xls, .ppt, and .pdf. The files are stored and backed-up online, and it is possible that several users edit the same file at the same time. File sharing permissions can be managed by administrators and document owners. *Google Groups* can be used to share docs, calendars, sites, shared folders, and videos with a group of employees instead of individuals. Group discussions are archived to enable searching and viewing of past and present discussions. *Google Sites* is a way to create secure, dynamic web pages for intranets and team projects. No HTML knowledge is required. Google Sites comes with system and site-level security controls. *Google Video* allows a secure hosting and streaming of videos, so there is no need to send videos over e-mail. Video sharing can be used e.g. in internal trainings and corporate announcements. *Google Wave* is designed to help communicate and collaborate with groups of people. It is aimed to build consensus on project work. Each wave is a shared space on the web that is updated live as users type. It can be used for brainstorming, keeping meetings on track, discussing project plans, and gathering feedback. [**20**]

Google Apps also has message filtering and 90-day archiving service that allow an IT administrator to have more control on inbound and outbound messages so that a company is able to comply with corporate or regulatory messaging compliance policies. Moreover, Google Apps includes 24/7 technical support for critical issues. Mobile access (for users of iPhone, BlackBerry and other mobile devices) to e-mail and other applications is included as well. Google offers migration tools and services for businesses to migrate over to Google Apps from e.g. Lotus Notes or Microsoft Exchange platform. Google also offers integration APIs for IT administrators to integrate Google Apps with an existing system. E.g. a single sign-on API integrates

Google Apps with existing Active Directory or LDAP systems. Google Apps Engine allows IT administrators to have custom application hosting on Google's infrastructure. With Partner program it is possible to take advantage of Google's third party solution partners. It is also possible to turn off advertisements in the e-mail browser. Google Apps (with the facts presented above) costs $50 per user account per year. [**20**]

Google Apps is like a virtual desktop in the cloud. As discussed above, Google Apps offers lots of applications for dealing with project type of tasks. What Google has done, actually, is that it has created its own versions of Microsoft's popular Office applications and Windows Live applications, and complemented the collection with some new ideas like Google Video and Google Wave.

3.1.2 SaaS Qualities

In Section 2.3, a few key characteristics of SaaS were listed: data managed by provider, availability, service customizability, pay per use, reusability, and scalability. Next, it will be studied how these qualities can be found in Google Apps. In addition, there will be some discussion on security, which is closely related to the data managed by provider quality.

To get started, a customer needs to register on the web. First thing that is asked is the customer's domain name. An own domain name is needed for sign up. After purchasing the service, the customer can use the Google Apps setup wizard to get started quickly. The customer needs to do some administrative tasks like setting up Google Apps core features (e.g. Gmail). However, the service provider will take care of installation and data management. The customers do not observe where their data is stored and how the data is managed. This SaaS quality eliminates the need to install and keep the application updated. Installing a traditional office application suite can be quite time-consuming. Then, after the application is installed, the customer probably will install some kind of updates, or patches, to the software. The update may be a larger feature upgrade or a fix for a security vulnerability. In any case, a lot of effort will be needed to do the installation and applying all updates, especially if the application shall be used by many employees. The customer sees this as ease of installation – because there is no installation. This is a benefit also from the provider's point of view, because he does not

need to create installation software, which would perform the installation on customers' workstations and would work in all operating systems that need to be supported.

One benefit of the SaaS approach of Google Apps is that the customer can access all the applications with a browser anywhere there exists an Internet connection. He does not have to carry his own computer, but can use any computer connected to the Internet. The applications can be used on a smartphone, too. It may not be practical to use all of the office applications with a phone interface, but at least e-mail, calendar, and mapping service applications will probably function fine. In the SaaS model, applications are accessed through a thin client interface, which typically is a web browser. The benefit in this quality is that the applications are accessible from diverse client devices: a powerful workstation, a typical home desktop computer, a laptop, a smartphone, a netbook, or a tablet computer.

The customer does not have to worry about backups, because all the data is automatically stored in the cloud and thus backed up. Data is replicated in multiple data centers for redundancy and consistent availability. Based on my own experience, taking backups of important files will cause some extra work, if the back up process is not automated.

It is crucial that the consumer can be confident that his critical data is safe. Google Apps is built with security and reliability in mind [**21**]. Someone could think that there is less security when the information is stored in the cloud, on the Internet. Anyway, the biggest security risk is often the user itself. Many people use weak or easily guessed passwords. Some people will open e-mail attachments that should not be opened. One notable risk is that a laptop or a USB drive is stolen or lost and the information on it will end up in the wrong hands. The SaaS model will eliminate this risk. In Google Apps, and I believe in any SaaS service, the protection of customers' data is the provider's top priority. Google Apps has achieved FISMA[5], SAS 70 Type II[6], and Cloud Security

[5] The Federal Information Security Management Act of 2002 (FISMA) is a United States federal law relating to the information security of federal agencies' information systems. FISMA applies to all information systems used or operated by U.S. federal agencies [**21**].

Alliance (CSA) certifications. Google has a full-time information security team working on defense systems, security reviews, and security policies and standards. Each Google server is custom-built including only the necessary software components to reduce exploit risks. Google also provides a two-step verification feature, which adds an additional layer of security to users' Google Apps accounts. Even if someone cracks or guesses user's password, an attacker is not able to sign in without access to the verification code that only the user can obtain via his own mobile phone. Security is built into every application of Google Apps from day one. It has to be like this, because security is so essential factor in SaaS. It seems that a user of Google Apps is well prepared for possible disaster. When using Google Apps, the customer does not have to worry about disaster recovery [**22**], because it is taken care of by the provider. As discussed earlier, the SaaS model enables rapid updates, or the application stays up-to-date automatically. This is important also from a security point of view, because it reduces the risk that the application is vulnerable to new threats. Businesses also get some security benefits or customizable security features with Google Apps. These include custom spam and inbound mail filtering tools (in addition to spam filters that are used by default), custom outbound mail filtering tools, custom information sharing rules, custom password length requirements, and enforced SSL connections to ensure secure HTTPS access. In conclusion, SaaS brings several security benefits over traditional software deployment model. However, some questions related to security arise: Who owns the data? Google's answer is that it does not own the data that organizations put into Google Apps, but it does not make a stand on whether the data belongs to the organization singing up for Google Apps, or the individual user. Moreover, how can the consumer be sure that his information is not used by Google? Google has admitted that it has wrongly gathered information sent over unsecured wireless networks [**23**]. Data has been collected by the software used in the Street View cars. How would Google use the collected data? If Google did not plan to use it, then why did it collect it in the first place?

[6] SAS 70 Type II audit means that an independent auditor has examined the controls protecting the data in Google Apps (including logical security, privacy, data center security, etc.) and provided reasonable assurance that these controls are in place and operating effectively [**21**].

While security is vital for SaaS, so is availability. Google guarantees in the Google Apps SLA that Google Apps will be operational and available at least 99.9 % of the time in any calendar month. This means that there might be small breaks in service uptime; in total, there may exist 43 minutes of non-operational time in a month (if you assume that a month contains 30 days). According to Google [**24**], a Google Apps customer typically experiences less than 15 minutes of downtime per month, and that 15 minutes average represents small delays of a couple of seconds here and there. Google Apps uses synchronous replication, which means that customer's data in Gmail, Google Calendar, Google Docs, and Google Sites is simultaneously saved in multiple secure data centers. If there is a problem in one data center, the system will instantly redirect to another data center, which can serve the customer's account. This quality reduces the possibility of interruption in service. When using a SaaS application, there is a small chance to face some downtime in service. On the other hand, if the customer would use his own computer to run similar applications, he could someday face a hard drive crash, which would cause an interruption to his work as well. Then there is always a chance that the customer will have an interruption in his Internet connection availability. In that case the problem would be in customer's Internet service provider, not in the SaaS service. However, the result would be the same; the customer would not be able to use the service. Which one provides the highest availability, a SaaS application running in the cloud or an application running on a local computer? It is hard to say. Achieving a high availability, like 99.9 %, probably causes the service provider some additional work. The customer does not care about this, he is just happy when he can use the service when he wants.

Customers have administrative and data control, which means they can customize Google Apps to meet their technical, branding and business requirements. Integration options are provided to connect Google Apps to customer's existing authentication system and existing user directory system. There exist also e-mail routing and e-mail gateway support, and e-mail migration utility and API. System branding enables customer's own logo and colors being used in Google Apps. The customer also has contractual ownership of employee data. Customizable security features were discussed earlier in this section.

Pricing of Google Apps is simple; Google Apps for Business costs $50 per user account per year. One user account is regarded as to be one e-mail inbox. User aliases and domain aliases are not counted as additional user accounts. If a customer has high employee turnover, it is not a problem; the customer just requests a total number of users at a domain and can then delete and create accounts so that he stays within the user account limit. In addition to the annual plan mentioned above, Google also offers another pricing plan called flexible plan. The contract term is one year with the annual plan, and there is no contract with the flexible plan. The flexible plan is $5 per user account per month, which totals $60 per user account per year. This is useful for companies that may double in size during the summer, for example. Consequently, the pay per use quality of SaaS is clearly utilized. The amount of usage here means the number of user accounts purchased by the customer. What are the advantages of this quality? Probably cost savings. 50 dollars does not sound that big amount of money for one user account per year. With this money an organization will get Gmail storage of 25 GB per user, Google Docs storage of 1 GB per user, and Google Sites storage of 10 GB plus 500 MB per user. It is plausible that an organization will save in IT costs if they change from a commercial on-premises system to Google Apps. At least the organization will know exactly how much money is needed to maintain web-based office tools (Google Apps) in a year. Because of the on demand nature of SaaS, a customer can start using the service whenever he wants and, respectively, stop using it at any time, independently.

It is possible to find reusability in Google Apps. For instance, Gmail users can use the built-in Google search to find e-mail messages. Google did not have to develop search functionality again, but it could just utilize its existing technology. The same applies to the cloud infrastructure. It has already been there when Google offered only Internet search services. There is no doubt that Google has used the same infrastructure to build Google Apps.

Obviously, Google Apps is hosted by Google. The company has invested in cloud computing for more than ten years. Google’s multi-tenant infrastructure is designed so that it can deliver improvements or new functionality to its entire customer base on short iteration cycles, like on a weekly basis. When Google launches upgrades or

patches, customers do not need to do anything. Upgrades happen in the background. Google has a global infrastructure, originally built for search, which is capable of rescaling resources (e.g. memory, disk space, network, or CPU usage) based on customer's requests. Google itself states that "Cloud computing is in Google's DNA". This is doubtless true, considering the history of the company. The scalability quality of SaaS is fulfilled. Multi-tenancy means that all Google Apps customers are running a single, large instance of Google's application. For the provider, this is a benefit, because it can concentrate on keeping this one process running instead of thousands or millions of processes. It is likely that one instance will be cheaper than millions of instances. This is also the reason why Google can provide Google Apps as cheap as it is at the moment. For a customer, this means that he is using provider's shared hardware resources. Employees of the customer are not required to have high-end workstations anymore, because the heavy computing is done in the cloud. It is enough that each employee has a working thin client available.

In conclusion, all the qualities of SaaS listed at the beginning of this section could be found in Google Apps. The qualities were well utilized. Furthermore, most of them have brought some extra value to the provider or the customer, or both. Only the availability quality can be read as a disadvantage. It is a downside for both the consumer and the provider. The consumer might suffer an interruption in service, even though the provider is able to offer service availability of 99.9 % of the time. The provider needs to take extra effort to achieve the availability promised in the SLA. On the other hand, thanks to SaaS, the application is available (accessible) from various client devices through a thin client interface.

3.2 AzureBlast

3.2.1 Overview

Lu et al. [**25**] have examined applicability of the cloud model to scientific applications by investigating an implementation of a known and computationally intensive algorithm called BLAST (Basic Local Alignment Search Tool). In their paper, Lu et al. introduce a methodology that they use to study how well cloud platforms are applicable to scientific computing. They also analyze the results from their study, by examining

especially the best practices of handling the large scale parallelism and large volumes of data. The performance evaluation is carried out on Windows Azure of Microsoft, but the results can be easily generalized to other cloud platforms as well.

BLAST is a popular life sciences algorithm that is used generally in bioinformatics research. It is crucial to many life sciences applications and its characteristics are typical of many applications important to data intensive scientific research. BLAST is one of the most widely used bioinformatics algorithms. It can discover the similarities between the two biological sequences (e.g., proteins). The BLAST algorithm is given one nucleotide or peptide sequence. It then searches against a database of subject sequences and discovers all the local similarities between the query sequence and subject sequences. [**25**]

Executing the BLAST algorithm can be computationally very intensive because of the large number of the pairwise genome alignment operations. It can also be data-intensive due to the large size of the reference databases and query output. However, it is also relatively easy to parallelize the BLAST implementation, because every pairwise alignment can be conducted independently. Two parallel schemes, query segmentation and database segmentation, have been widely adopted and applied to BLAST. Several solutions have been proposed to run the algorithm on a multi-core machine or a cluster. The majority of researchers have not had access to the large-scale resources needed to perform this parallelization. However, cloud computing provides possibilities to expand the availability of large-scale alignment search to every researcher as a single researcher or small team can have access to the same large-scale compute resources as large, well-funded research organizations. [**25**]

AzureBlast is an experimental prototype that is intended to assess the applicability of cloud platforms for science applications. It is a parallel BLAST engine that runs on Windows Azure. AzureBlast is built directly on the basic services of Windows Azure that can assemble the compute power of thousands of Azure instances. The National Center for Biotechnology Information (NCBI) offers a publicly downloadable implementation of BLAST algorithm called *blastall*. The blastall application can parallelize the search on a multi-core machine by partitioning the database into

segments and generating multiple threads to search against each of them in parallel. Lu et al. adopted the query-segmentation data-parallel pattern to run BLAST on multiple instances. They gave AzureBlast an input file with a number of query sequences, and it split the sequences into multiple partitions that were delivered to worker instances for execution (one partition per one worker instance). Finally the results were merged once all partitions had been processed. The benefit of the query segmentation is that it needs little communication between instances unlike the alternative, database segmentation scheme, which requires inter-node communication for each query. One or more web role instances receive the requests from the user via a web portal or a web service interface, and a bunch of worker role instances do the heavy compute work, i.e. executes the NCBI blastall software. Thus, AzureBlast is following the general model for an Azure application (See Section 2.6.2). [**25**]

3.2.2 Realization of SaaS Characteristics

In AzureBlast, the job scheduler and the job submission portal are separated by the job table, which is a dedicated Azure Table. See Figure 13 for clarification. The job scheduler is an independent process. It fetches the job from the job table, schedules its execution, and maintains the state of the job. Windows Azure and Amazon EC2 recommend using a reliable message queue as the communication method between instances. Queues provide the buffer required to handle workload bursts. They decouple the parts in order to make the system more robust against the instance failure. In addition, the application can operate without having to know the exact number of worker instances. This feature takes into account transparent scaling of the system, which is one of the most essential features for scientific applications where data inputs can vary significantly between runs. Moreover, this (scalability) is one of the characteristics of SaaS, as discussed earlier in Section 2.3. [**25**]

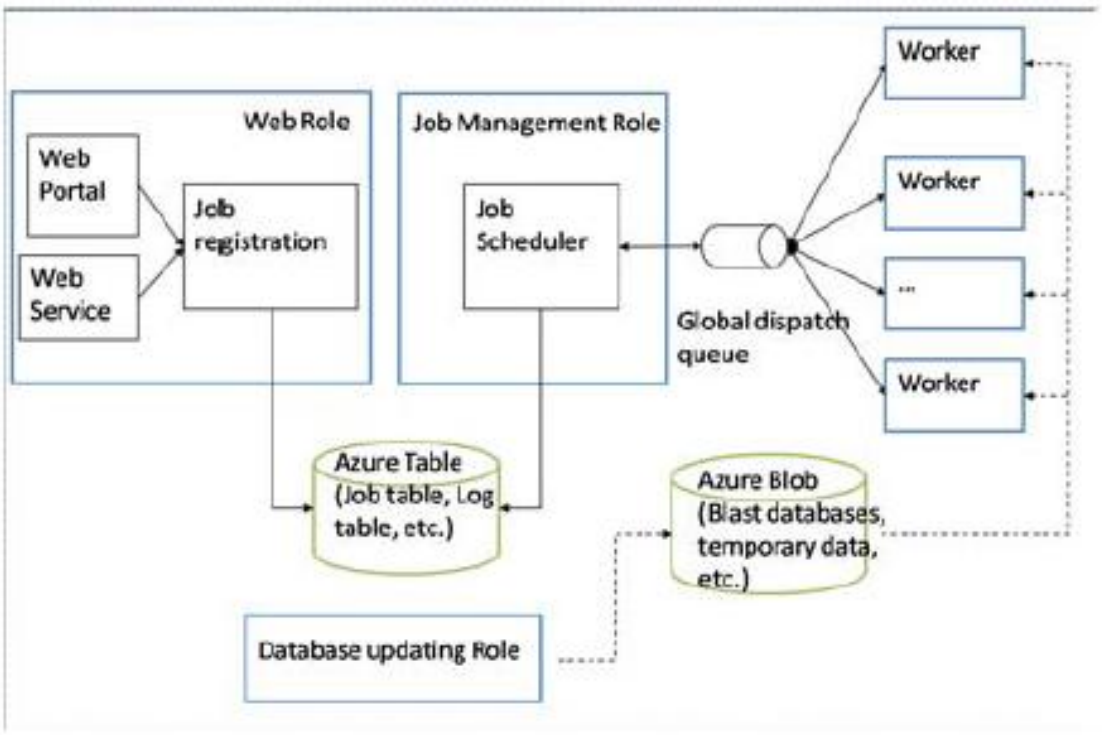


Figure 13. The architecture of AzureBlast [25**].**

The job submission portal will register the job into the job table before it returns a job ID to the user. This is important for fault tolerance; the immediate persistence mitigates loss in case of a worker role crash. Similarly, all the job states are kept in the Azure Table to prevent data loss caused by instance failure. Because the two components of AzureBlast are decoupled by the job table, it is possible to run them on two different instances, which gives better fault tolerance. The loss of one will not affect the other, for the failover mechanism of the cloud platform keeps the entire system working. The cloud fabric takes care of failure recovery and health monitoring. As mentioned before, queues are essential in managing workload bursts. AzureBlast maintains a global dispatch queue, which enables dynamic scaling to the number of worker instances based on the length of the queue. Fault tolerance is important to guarantee decent usability and availability of the service, which is one of the characteristics referred to in Section 2.3. In this special case it is also vital, because a BLAST query over a typical size genomics database can take hours or even days. Therefore a batch system is required. It allows the user to submit jobs and periodically query the status of the jobs. [**25**]

Lu et al. measured the scalability of AzureBlast. They used 64 statically allocated, large size instances. The database staging was done during the instance initialization in order to guarantee that each instance had a local replica and the database staging time was excluded from the measurement. First they deployed AzureBlast on Azure with one worker instance to measure the throughput of one job. Then they re-deployed the project with number of instances doubled and repeated the measurement. The input query had 4096 sequences and was partitioned into 64 partitions. Their measurement shows that the throughput of AzureBlast increases almost linearly when the amount of instances grows. [**25**]

When implementing AzureBlast, Lu et al. developed their own task parallel library for Azure. The library is a thin abstraction layer upon Azure messages. It has the necessary concurrency and coordination support required for the task parallelism patterns that are widely used in science applications. A task can be serialized into an Azure message. Then, all task messages are enqueued into the global dispatch queue. A worker instance deserializes the message and executes the task, when it gets a message from the dispatch queue. The developers of AzureBlast also built an exception handling mechanism for tasks. They also utilized a parallelism pattern called Join/Fork and the "continuation task" pattern. With this task library developed by Lu et al., it is quite easy to implement the data-parallel BLAST on Azure. Since they have created the task parallel library and used patterns in implementation, there should be reusability of some level, at least. [**25**]

Even though AzureBlast is a case study of a science application, there is some sort of service customizability, however. The main task of a BLAST job is to split the input sequences into multiple partitions, each of which is stored as one Azure Blob. The data partitioning task then creates one child task for each partition and sets up a continuation task to merge the results from all child tasks. Each child task downloads the partition from the Blob storage, and executes the NCBI blastall binary over it. Now what can be customized is the number of partitions, which can be set to 2x or 3x the number of instances. Another customizable setting is the value of *visibilitytimeout*, which basically is the estimate of the task running time, for each child task. It is also possible to define the number of processor cores that AzureBlast can use. In fact, AzureBlast automatically adjusts a command line argument of blastall that tells the BLAST

implementation how many cores it can use. The values are 1, 2, 4, and 8, which correspond to number of cores in an Azure instance. [**25**]

3.2.3 SaaS Qualities

The data managed by provider quality of SaaS exists. Earlier it was discussed that AzureBlast includes a web portal or a web service interface that works as a user interface. This removes the need for installation. It seems that blastall application is part of a C-based BLAST package, which is now deprecated. The package is called *legacy blast* while the new substitute, C++ based BLAST programs, is called *blast+*. In any case, installation of standalone BLAST (either blast+ or legacy blast) on Windows OS seems to be inconvenient [**26**]. It is obviously a benefit when a user can avoid the standalone BLAST setup and gets the same functionality with AzureBlast. An advantage for the provider is that he does not need to create detailed and long installation instructions (such as [**26**]) for users. If there will be updates of any kind for AzureBlast, they are applied automatically in the background.

When all the heavy work is done in the cloud, it is possible to access and use the application from various client devices through a thin client interface. Once again, only a suitable web-browser and an Internet connection are needed. This quality is definitively a benefit for the user who is not required to invest in expensive PC hardware anymore.

Availability of the service is essential for the user. Since AzureBlast is an evaluation project, there was no mention about availability percentage. One could think that it would be at the same level as it is in Windows Azure. At the moment, Windows Azure has separate SLAs for compute and storage. For compute, when two or more role instances are deployed in different fault and upgrade domains, external connectivity is guaranteed to be at least 99.95 % of the time. For storage, it is guaranteed that at least 99.9 % of the time requests to add, update, read and delete data are successfully processed. The providers of AzureBlast, Lu et al., cannot promise more than the provider of the platform, I think. Are these percentages 99.95 and 99.9 enough for the user? I do not know, but I would assume that they are acceptable for this kind of scientific use.

As AzureBlast is not a commercial service, it is understandable that there is nothing to say concerning pricing, which means the pay per use quality of SaaS is not utilized. Anyway, the pay per use model could be somehow based on the platform pricing policy. The Windows Azure platform charges for compute time, which is measured in service hours. Partial compute hours are billed as full hours. Storage is charged in Gigabytes and measured in units of average daily amount of data stored over a month. Moreover, data transfers are charged meaning transmissions to and from the Windows Azure datacenter. Data transfers within a subregion are free. Also transactions are charged and measured as application requests. [**14**]

Scalability of AzureBlast was discussed in Section 3.2.2. It turned out that AzureBlast is very scalable. Load balancing is handled using reliable message queues, and the system is quite fault tolerant. The application is operable even without knowing the exact number of worker instances. This kind of fault tolerance or performance offered by AzureBlast cannot be achieved on a traditional, local system, I believe.

Also reusability and service customizability were covered in Section 3.2.2. There is not enough information available to say whether some extra value has been gained because of reusability in AzureBlast. Customizing the number of processor cores in AzureBlast is easier than it would be in an on-premises system. When changing this setting in a local system, it is also necessary to change the hardware (1, 2, 4, or 8 processor cores). Thus, it can be said that SaaS brings benefits to the user.

To sum up, almost all SaaS qualities listed in Section 2.3 could be found in AzureBlast. The pay per use quality was not utilized, because of the non-commercial nature of AzureBlast. Moreover, there were not enough details concerning reusability to say if it had brought some benefits. Concerning the availability quality, it can be a drawback like in Google Apps. Otherwise, the rest of the qualities show that SaaS has provided extra value in this implementation case.

3.3 NASA Web-Accessible Open Software as a Service Framework

3.3.1 Overview

The next case presented here is the Grid XML Datastore Framework (GXD Framework) that is discussed in a paper by Maluf and Okimura [**27**]. GXD Framework is a modern approach of the implementation of SaaS for NASA[7]. Its goal is to reduce costs and increase efficiency. GXD Framework is an extension of a SaaS framework based on eXtensible Database technology (XDB). GXD Framework is meant for providing seamless access to distributed information resources regardless of location. At NASA information and information processing services are highly distributed. NASA and its contractors have hundreds of databases (with millions of records) and hundreds of desktop computers with millions of files. Numerous decision making applications utilize this information with hundreds of procedures and guidelines and thousands of diverse work practices.

The SaaS implementation consists of software as an application hosted on NASA and generic services – not hosted on NASA, but SalesForce.com – accessed via the Internet. GXD Framework is an open and extensible database architecture that enables efficient and flexible integration of diverse and distributed information resources. The new approach of GXD Framework does not include a database schema, but uses a document-centric object-relational XML database mapping. Structured, unstructured, and semi-structured information can be integrated without requiring document schemas or translation tables. GXD Framework utilizes existing international protocol standards such as HTTP, eXtensible Markup Language (XML), and Web-based Distributed Authoring and Versioning (WebDAV). GXD Framework makes possible – through a combination of these international protocols, universal database record identifiers, and physical address data types – that an unlimited number of desktop computers and distributed information sources can be linked seamlessly and efficiently into an information grid. [**27**]

[7] National Aeronautics and Space Administration

Instead of using the traditional relational database management system (RDBMS) GXD Framework exploits an object-relational (OR) model defined within an object-relational database management system (ORDBMS). The ORDBMS still uses relations (tables) to store database information, but also adds object-oriented features like data abstraction, encapsulation, inheritance, and polymorphism. There is a problem that is faced by both database programmers and application developers. The developers structure their data in a different way than it is structured in a database. Object-to-relational mapping is needed to convert data that is being inserted into a tabular format that the database can understand. Similarly, a conversion to other direction is needed when the relational information is returned from the database into the object format developers want to use for their programs. In order to take advantage of the OR model, a standard for common data representation and exchange is needed. Currently, the emerging standard is XML. [**27**]

GXD Framework has the ability to map XML-encoded information into a data model through a customizable parser (driver) that parses dynamically the hierarchical model of XML data. The customizable driver of GXD Framework simulates the Document Object Model (DOM) Level 1 specification on parsing and decomposition of elements. In object-relational mapping from XML to relational database schema, the data in the XML documents is modeled as a tree of objects, which are specific to the data in the document. Element type with attributes, content, or complex element types are usually modeled as object classes. Element types with parsed character data (PCDATA) and attributes are modeled as scalar types. Then, this model is mapped to the relational database so that classes are mapped to tables, scalar types are mapped to columns, and object-valued properties are mapped to key pairs. [**27**]

The GXD Framework API provides two major sets of interfaces. The first set is The Java Database Connectivity (JDBC) API, Open Database Connectivity (ODBC) API, and C/C++ API for application writers. The second is the lower-level Structured Query Language (SQL) and corresponding server level procedure language API for driver writers. GXD Framework can be accessed by applications and servers using standard compliant SQL-based drivers in particular for GXD Framework core schema-less configuration. [**27**]

GXD Framework driver API does not require configuration on the server side. All the information needed to map the information content is fully seamless as it is defined by the markup language. The driver library of GXD Framework does not require special installation, but it can be set to be automatically downloaded when needed. This reduces both the up-front development costs for applications and maintenance costs for database administration. [**27**]

3.3.2 Realization of SaaS Characteristics

The NASA Technology Transfer System (NTTS) is an agency database that handles intellectual property management and technology transfer, and new innovative partnerships. The software services contain contract/grant compliance, awards processing, partnership development, patent docketing, success stories, Innovative Partnership Program marketing and licensing. There are three interfaces in the system to serve three distinct communities: mainly agency-wide, center specific, and the public. [**27**]

NTTS is an initial application that was built using the current GXD Framework architecture. NTTS consists of a distributed information on demand model for document management. Modules of NTTS are extensible and can adapt to different data sources. An example assembly of NTTS consists of the following modules: 1) a set of interfaces that support various communication protocols (like HTTP, WebDAV, FTP), 2) an information bus to communicate between the client interfaces and the GXD Framework core components, 3) the daemon process that enables automatic processing of inputs, 4) the GXD Framework search on both document context and content, 5) a set of extensible APIs, and 6) the GXD Framework implementation for Oracle backend ORDBMS. [**27**]

There are three core components in GXD Framework: the high-throughput information bus, the asynchronous daemon process, and the set of customizable and extensible APIs built on Java 2 Platform, Enterprise Edition (J2EE) and Oracle PL/SQL stored procedures and packages. The information bus enables virtually three major communication protocols to meet the information on demand model. These protocols

are HTTP/WebDAV and HTTPS, the WebDAV, and the new Java Remote Method Invocation technology run over Internet Inter-Orb Protocol (Java RMI over IIOP). The GXD Framework daemon is a unidirectional asynchronous process, and its purpose is to increase performance and scalability compared to traditional synchronous models (like Remote Procedure Call (RPC) or Java RMI). The set of extensible APIs are used to enhance database access and data manipulation from most applications. [**27**]

A new aspect of NTTS is an automatic conversion of heterogeneous documents like word processing documents, presentations, and spreadsheets into HTML or XML. The conversion is done by the daemon process. Then, the parser stores the content in a connected node structure in the schema-less database. After that, users can query the heterogeneous documents as if they were a database. It is possible to retrieve the specific records of data from the heterogeneous documents by entering context+content keywords and phrases into the toolbar of NTTS. This retrieved record could be e.g. a cell of a spreadsheet document. [**27**]

GXD Framework makes it possible to have orders of millions of desktops and distributed information sources to be linked into a highly scalable information grid (see Figure 14). [**27**]

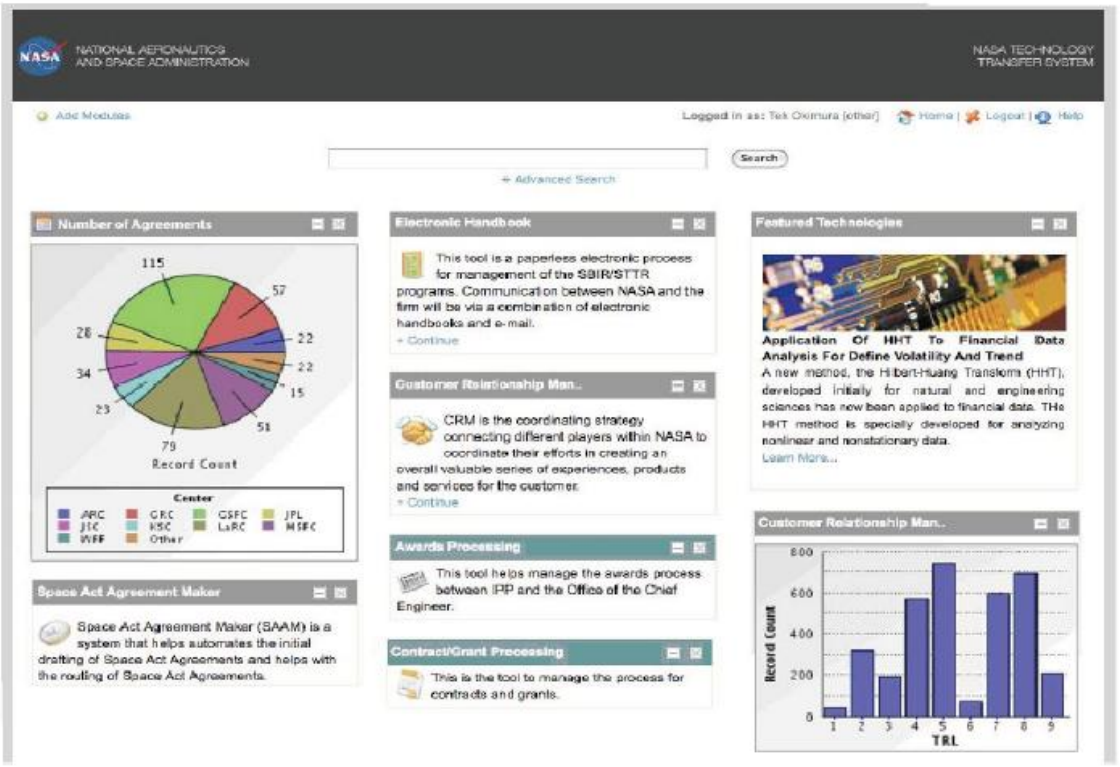


Figure 14. GXD Framework used in a NTTS web page [27].

Unfortunately the paper by Maluf and Okimura [**27**] did not provide that much information on the actual SaaS application, i.e. NTTS. Therefore, it is difficult to review how well it fulfills the SaaS characteristics (see Section 2.3). The main focus in the article is describing GXD Framework, i.e. the SaaS framework. It provides an extensible, schema-less, information on demand architecture that allows integration of distributed and heterogeneous information resources for scientific and engineering applications. GXD Framework is a scalable, high-throughput database framework for transforming unstructured or semi-structured [8] documents into well-structured and standardized XML or HTML formats. The framework is also meant for managing, storing, and retrieving unstructured or semi-structured data.

[8] Semi-structured document here means a document whose data may be irregular or incomplete, and the document structure can be changing fast. E.g. a web page or a word processing document that is constantly changing is a semi-structured document.

3.3.3 SaaS Qualities

Only limited information is available on NTTS, but it is possible to do some analysis on SaaS qualities through the qualities of GXD Framework, i.e. the SaaS framework used to build NTTS.

In Section 3.3.1 it was discovered that the SaaS implementation consists of software as an application hosted on NASA and generic SalesForce.com services provided across the Internet. It is unclear what the role of this software as an application is. The article by Maluf and Okimura did not explain it further. What can be said is that NTTS is used through a web page and thus it is accessible through a thin client interface. This is a clear benefit, because there is no need to install the application on every desktop computer of NASA.

The paper did not give any information on service availability. Therefore, it cannot be analyzed at all.

From a software acquisition perspective, NASA pays once for the database and storage and thus third-party application plug in on demand [**27**]. At the beginning of the paper, Maluf and Okimura say that implementation of software as a service for NASA is a way to reduce costs and increase efficiency. This is all what is said about pricing.

GXD Framework allows that orders of millions of desktop computers and distributed information sources can be linked into a scalable information grid. The set of extensible APIs is one of the core components of GXD Framework. In addition, NTTS consists of extensible modules and they can adapt to different data sources. It seems that NTTS can be easily extended. There exists scalability to that extent. However, it is not possible to say how NTTS rescales the resources based on customer's actions.

In SaaS, software itself is a target of reuse; NTTS is deployed to service consumers over the Internet and the service can be used by various customers. Thus, the reusability quality is fulfilled.

One of the three core components in GXD Framework is the set of customizable and extensible APIs. Also, a customizable parser (driver) is used when XML-encoded information is mapped into a data model. But can service customers do customizing in NTTS or GXD Framework? This question cannot be answered due to lack of information.

In conclusion, some of the SaaS qualities listed in Section 2.3 could be found in NTTS or GXD Framework. However, analysis of some of the qualities (data managed by provider, availability, pricing, and service customizability) could not be done, because the article did not provide enough information related to those. The web-based interface of NTTS is an advantage brought by SaaS in any case.

3.4 F-Secure – a Transformation Case

3.4.1 Overview

F-Secure Corporation is a company with over 800 employees in Europe, North America, and Asia. The main business of F-Secure is to sell commercially available off-the-shelf (COTS) software and services for virus protection and intrusion prevention. The company's solutions – targeted at Internet service providers (ISPs) and mobile operators, and their customers – are based on the SaaS model. In 2005, SaaS business through ISPs became the company's main business area, and it represented 44 percent of total revenues in the fourth quarter of 2008. [**28**]

Komssi et al. [**28**] discuss transforming F-Secure from software products to software services. To study a transformation, they performed a retrospective case study at F-Secure. As a result, they present a set of success factors and challenges related to utilizing the SaaS model. The F-Secure case was selected by Komssi et al. because F-Secure was the first software product company in its field to take advantage of SaaS. Also the implementation of SaaS at F-Secure has been successful business-wise. The paper [**28**] does not go into details of F-Secure solutions. Hence, to give more practical image of SaaS solutions of F-Secure, one of the current services is discussed in Section 3.4.3, which is based on other sources.

One statement that is presented by Komssi et al. [**28**] is that software companies need to become service companies in order to be competitive in the software business. They state that a transformation from the product business to service business seems to be vital for a software company. It is also said that the shift is demanding and challenging for the company. Instead of producing software products for a mass market a company has to start serving individual customers. In addition, the changes to the business model of the company need to be emphasized.

Based on a source of theirs, Komssi et al. [**28**] say that a successful transformation requires leadership, vision, and strategy. Another source (Grönroos) cited by Komssi et al. states that implementing a service strategy means a service culture, and thus a cultural change is needed in many companies. According to Grönroos, employees of an organization should be characterized as service-focused (in a service culture). A cultural change is a long process that requires extensive and long-range activity programs. The change has to be led by the top management.

According to another source (Sääksjärvi et al.) used by Komssi et al. [**28**], the major challenges for a service supplier when adopting the SaaS model are: 1) managing the complex network of partners, 2) the smaller service fee per single transaction compared to license sales and consultation, 3) the probable performance and scalability issues, and 4) the high initial investment.

Gold et al. (referenced by Komssi et al. [**28**]) suggest that the transformation is expected to be quite slow and gradual. Organizations will wrap their existing offerings as services step-by-step. Gold et al. propose that when an organization initiates the shift with existing systems, it reduces the transformation risk and increases the potential return on investment. The shift also demands new, non-technical skills for software engineers. They must be capable of negotiating and communicating with clients and service providers. It is required to understand the business policies and activities of their customers.

Komssi et al. [**28**] established a research partnership with F-Secure in June 2006. Their case study was part of their research work examining service-orientation in software

product companies. The data collection and analysis were carried out iteratively between years 2006 and 2008. Nine key persons who had responsibility for the development of SaaS and/or who will have a key role in the development of future solutions were interviewed. The interviewees included executive team members, vice-presidents, directors, and managers from R&D, customer advocacy, and service development.

F-Secure was focusing on the software products development during the years 1991–1999. The company trademarked its SaaS solution as *Security as a Service*. F-Secure has launched the following Security as a Service solutions [**28**]:

- 10/1999: solution for outsourcing
- 02/2001: solution for consumers
- 08/2004: solution for mobile market
- 10/2005: solution for enterprises
- 02/2006: solution for small businesses

The first solution, launched in 1999, was not originally planned to be SaaS and lacked flexibility, which hindered its commercial success. The second solution was more successful. F-Secure was a supplier of software and service solutions for consumers through ISPs, like France Telecom and Charter Communications. In the solution, F-Secure hosts the service and an ISP takes care of selling and billing the service for consumers. The security service is a complement to the ISP's broadband service. [**28**]

About three years after the second launch, F-Secure announced that SaaS was the fastest growing business area in the field of software security. The company branched out their mobile and enterprise businesses into service-oriented businesses through ISPs. In addition, F-Secure launched a solution for the small business market. All the three solutions followed the principles of the second Security as a Service solution. [**28**]

All five Security as a Service solutions were based on technologies that were already used in COTS products. F-Secure adopted the SaaS model over the existing core technologies in these five solutions. However, in 2008, the company launched a service solution that was developed especially for the SaaS business. The solution is based on a

new technology whose intention is to advance the SaaS model and ISP partnerships. The solution is not mentioned by name in the case study, but when looking at F-Secure press releases from the year 2008, it is obvious that the solution in question is F-Secure Online Backup. [**28**], [**29**]

3.4.2 Findings: Success Factors and Challenges

Komssi et al. [**28**] studied a transformation from traditional software production to the SaaS model at F-Secure. The case study was performed on the basis of the research question: "What are the success factors and challenges for the company in its transformation over time?"

In the case study, Komssi et al. [**28**] present four success factors they have found: 1) the key personnel have to have great confidence in the services, 2) a dedicated team is vital in service development, 3) close collaboration with the most important customers is essential in new service development, and 4) flexibility is a key characteristic of a successful supplier.

The top management of F-Secure has had a solid confidence in its services since the launch of the first service solution. A separate service subsidiary was founded to manage and develop SaaS solutions at F-Secure. The service subsidiary had a strong belief in the success of SaaS. During the recession between the years 2001 and 2003 the company stayed the course towards becoming a service-oriented business, even though losses were significant. [**28**]

It was essential that F-Secure founded the separate service subsidiary. It was founded less than 10 months after the first service solution launch. This separate organization was greatly committed to service development. The team had clear focus, which was recognized as one of the key success factors in the transition. The subsidiary was merged with the parent company in 2002, which made the service organization a little bit disoriented. Thus, F-Secure later established separate business units for the needs of different markets. [**28**]

F-Secure was already in cooperation with at least one premier customer each time a new SaaS solution was launched. The service development team was analyzing the needs of the premier customers to create acceptable solutions for them. The customers did not want to invest in a SaaS solution until they were confident enough. Therefore, the collaboration including piloting was very important. [**28**]

Customers regarded F-Secure as a flexible and reliable supplier. It was possible to customize the solutions and promote the customers' brands. The implementation and utilization of the service were efficient and easy for the customer, because F-Secure was hosting the service. The premier customers also had the highest priority in F-Secure's development actions. [**28**]

In their findings, Komssi et al. [**28**] also come up with four challenges: 1) it may take a long time to get a SaaS solution profitable, 2) cultural change from software products to software services may be slow and demanding, 3) it is challenging to find a balance between the customer and the customers' customers, and 4) analyzing customer feedback becomes more difficult when the customer base increases.

Only one of the SaaS solutions, the service for consumers, has been highly successful. The other SaaS solutions have not yet made any significant profit. F-Secure was able to make the SaaS solution for consumers profitable in one year. Seven years later, its revenue made up about 40 percent of the total revenue of F-Secure. At that time the company had 169 partners in 38 countries. It is still a challenge to find the right way and timing for making the other SaaS solutions successful. [**28**]

The personnel of F-Secure has a history of developing, marketing, selling, and delivering software products. Many employees still see F-Secure as a software product company, which causes overemphasizing the role of software and software features. According to one of the interviewees the biggest issue for the company is the cultural change towards services. The cultural change requires a new mindset from the employees. All the workers need to understand that F-Secure provides services, not just software and software features. [**28**]

Half of the interviewees had the opinion that ISPs are the most important customer group, which should drive the development of service solutions. The other half thought that either consumers who are customers' customers are the most important, or both ISPs and consumers are equally important. Some of the people that were interviewed pointed out that F-Secure has to know the needs of the whole customer chain. [**28**]

During the years the number of service customers has increased significantly. Thus, also the amount of customer feedback has increased, which has made processing the feedback more challenging. According to the informants, the customers mostly ask for software corrections and feature enhancements in the feedback. Very little feedback about the service processes has been received so far. Some of the interviewees thought that the service process should be actively reviewed to guarantee customer satisfaction. [**28**]

As mentioned in Section 3.4.1, F-Secure launched five solutions that were based on existing technology used in software products of F-Secure. It is likely that F-Secure has exploited the findings from these cases when it launched a service that was built particularly for the SaaS model in 2008. Next, this new service will be studied.

3.4.3 SaaS Qualities

One of the solutions offered by F-Secure today is F-Secure Online Backup Service for Consumers [**30**] (later Online Backup). It is a security, or storage service aimed at ISPs so that they can provide their subscribers an easy-to-use, unlimited and fully automated backup solution that protects their customers' important files against hardware failure or accidental deletion. Unlimited here means that Online Backup allows backing up all files on a hard drive, regardless of the number of files [**31**]. However, external hard drives or devices like USB memory sticks are not backed up. With Online Backup, customers can prevent the loss of any digital content like pictures and music files.

F-Secure offers ISPs an outsourced solution that is managed by global hosting centers, which provide continuous support. With this turnkey solution, ISPs get an easy and fast service deployment. Online Backup is designed to easily link with CRM systems. Currently, there are two solution types available: Standard and Custom. While Standard

solution is meant for small and medium-sized service providers, Custom is aimed at medium and large-sized service providers who want to combine their existing service portfolio with automated backup services. The Custom solution comes with support for branding. [**30**]

Online Backup comes with a locally installed graphical user interface (see Figure 15), but files are backed up in the cloud. In order to utilize the service, the user needs to have one of the supported operating systems: Windows 7, Windows Vista, Windows XP (32-bit, Service Pack 2 or later), or Mac OS X (10.4.9, 10.5.x, 10.6.x or later) [**31**]. Naturally, a high-speed Internet connection is required, too.

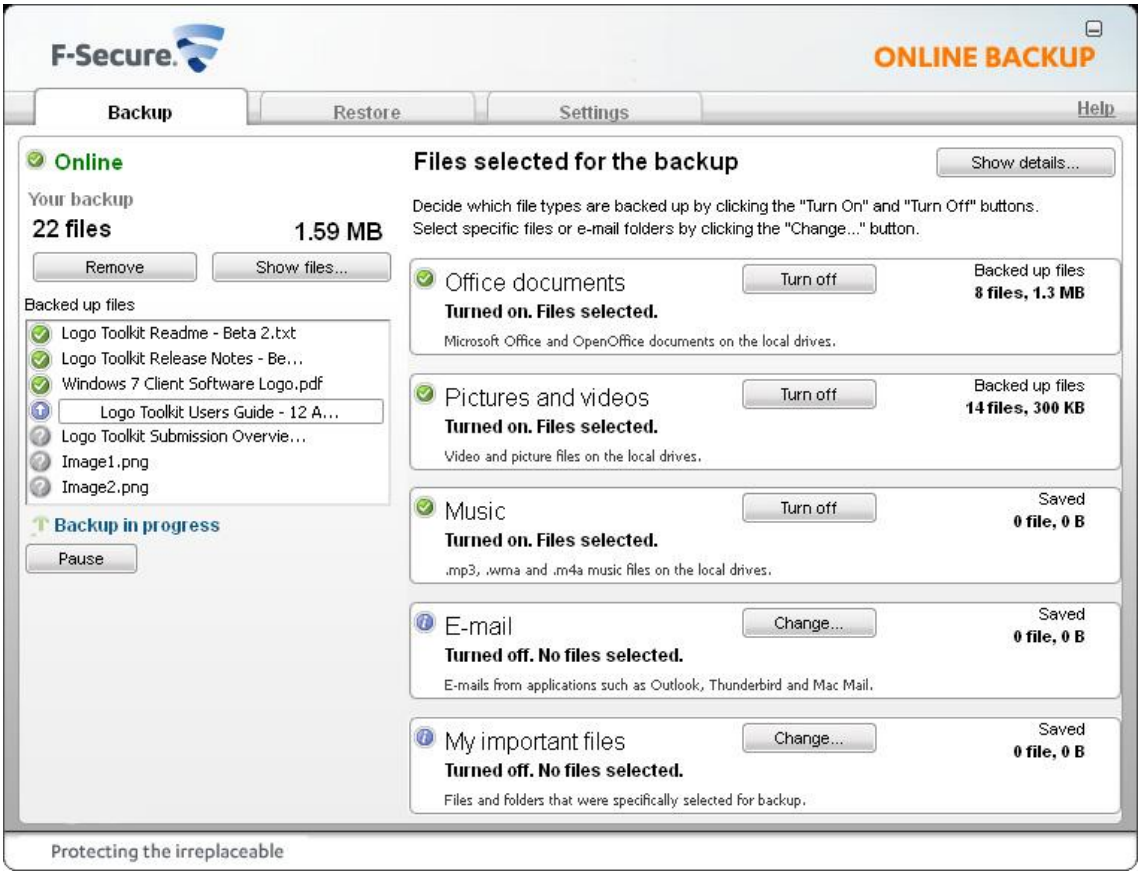


Figure 15. F-Secure Online Backup GUI seen by end-users [30**].**

From the ISPs' point of view, Online Backup fulfills many of the SaaS qualities. Online Backup is a fully hosted service by F-Secure. File transfer and storage is secured. Customers get limited or unlimited capacity for their content. Customers can also access

the content from any PC. Online Backup is fully customizable according to ISP's brand. [**30**]

These SaaS qualities bring some benefits for an ISP. Because the data is managed by F-Secure, the ISP does not need to build an infrastructure to host the backup solution. The ISP is not responsible for file storage and recovery. It just sells the service to subscribers in addition to the broadband connection. Online Backup solution frees ISPs from the setup, management, and maintenance of the needed backend servers to F-Secure, but ISPs still preserve ownership of the subscribers. F-Secure provides a customization process, which allows the ISP to offer the backup service under its own brand. If the ISP is able to sell the backup solution to a customer, it means that the customer, or his data, is tied to the ISP. Thus, it will be less attractive for the customer to change to a competing ISP.

F-Secure has developed a platform called F-Secure Content Enabler [**32**] (later Content Enabler). It enables building of online storage solutions and is designed for operators to offer content related services. Online Backup is one of the two ready-to-use storage services provided by Content Enabler. The architecture of Content Enabler consists of three layers: the application layer, the data virtualization layer, and the physical layer. The application layer handles all the data management functions, like file indexing, document conversion capabilities, and server-side encryption or decryption. The data virtualization layer optimizes bandwidth usage and caches requests in order to ensure performance and robustness allowing all the main components (physical storage, database, and streaming capabilities) to be split on multiple servers and sites. The physical layer guarantees physical storage based on industry standards. This modular architecture, presented in Figure 16, makes Online Backup highly customizable.

The application layer manages files. It provides the following features: feature-based access rights, extended public and private notion (virtual projects, sharing, etc.), user-profile management, playlist management, and album, set and collection management. The layer also saves all the necessary data associated with the files. For instance, contextual information like e-mail header and context (e-mail), Exif metadata (pictures), ID3 tags (music), and PIM (address books) is saved. It also generates thumbnails and

attaches music covers automatically. The application layer also protects the content by supporting encryption and virus cleaning. [**32**]

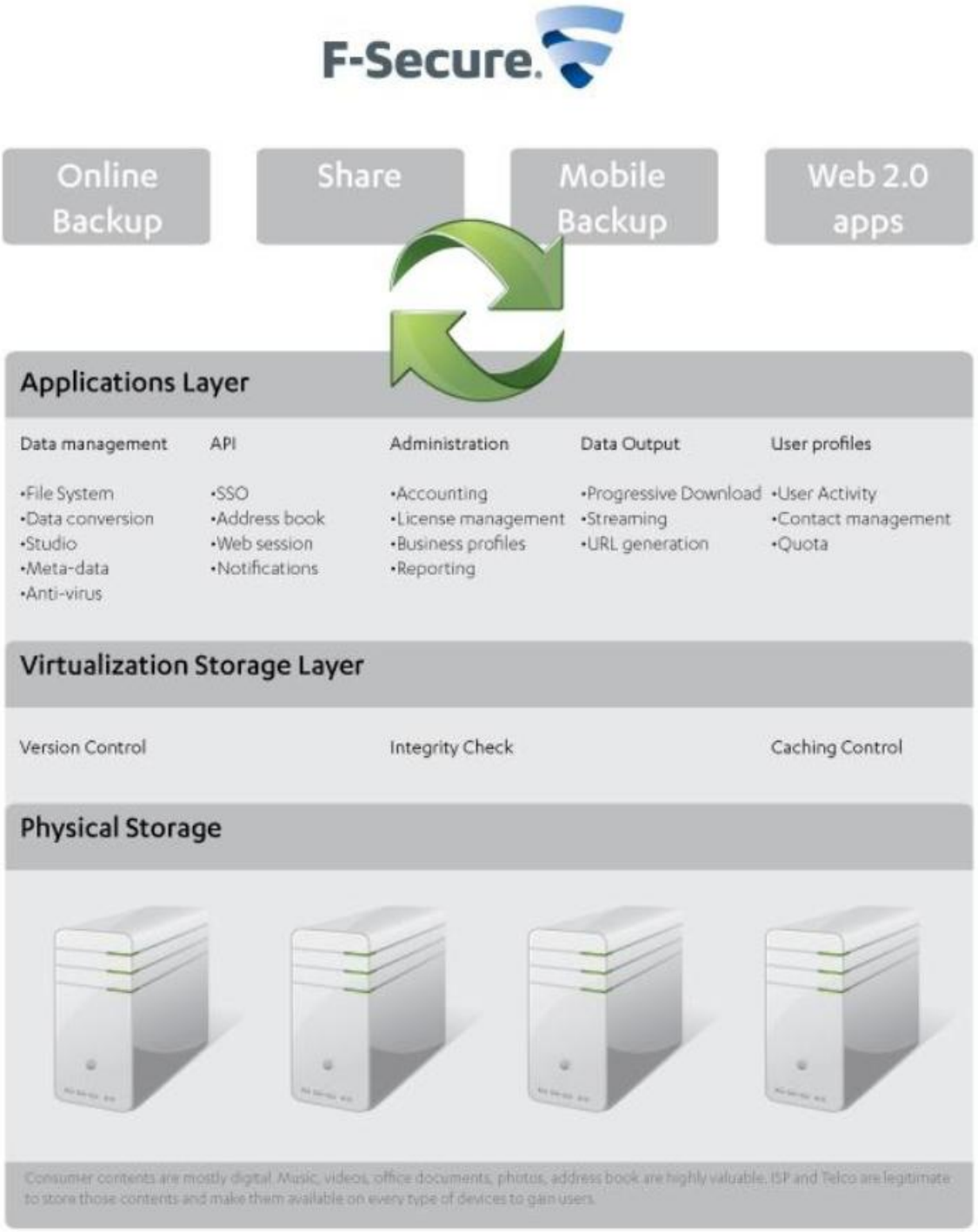


Figure 16. The layers of F-Secure Content Enabler [32**].**

The virtualization storage layer includes functions to ensure scalability, robustness, and high performance of the overall storage platform. The functions are integrity check,

version control, and caching control. Caching control is used to reduce the access time to the most used files and services. Integrity check is the main function of the layer and it has an essential impact on the performance of the main components of Content Enabler illustrated in Figure 17. [**32**]

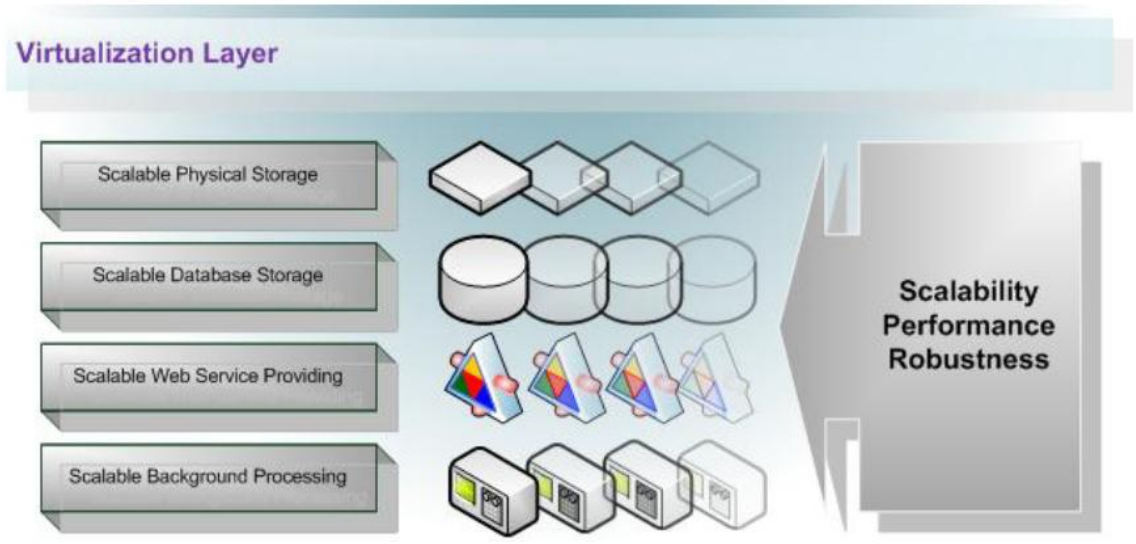


Figure 17. The main components of F-Secure Content Enabler [32**].**

Physical storage means separate physical storages, e.g. two data centers with full mirroring. Database storage is a split database to achieve better load response time when the storage service is heavily loaded. Web service providing is a robust web access. Background processing gives better performance for file encoding and thumbnail generation. [**32**]

The scalability quality of SaaS is well utilized in Content Enabler and therefore in Online Backup, too. The virtualization layer of Content Enabler supports a virtually unlimited number of users.

Also from the customers' point of view, SaaS qualities of Online Backup give benefits. Backup files are managed by the provider. Backup data is stored in the cloud rather than customer's local hard drive. This is definitively an advantage, because backup files consume lots of disk space. Online Backup offers unlimited storage for all customer's files, regardless of size or type of data. Optionally a cheaper solution with a limited quota may be provided to the customer. Backup files can be accessed anywhere with

any device through a personalized web page. Online Backup compresses files to minimize bandwidth usage. All files are encrypted in transit and in storage.

F-Secure offers Online Backup through ISPs but also sells subscription to the service on its own web page. It is possible to subscribe for 12 or 24 months. A license for one computer for 12 months costs 49,90 € at the moment. For a license of 24 months the user has to pay 82,90 €. A license with a limited storage space of 5 GB costs 15,50 € per year. With subscription length of 24 months the customer pays 3,45 € per month, which does not sound that expensive price.

A new feature in Online Backup is the possibility to share backed up files with friends, family members, and coworkers. Sharing is easy; no uploading of files, no accounts to create – just a couple of clicks from user's desktop. This would not be possible without SaaS.

As a SaaS solution, Online Backup stays always up-to-date. Also client software is automatically updated. Online Backup works in the background so it needs minimal user interaction. It detects and automatically saves all the content that is generated on a computer. Automatic backups is a great feature, because even though we know that taking regular backups is important, only few of us have the time or knowledge to do this. Online Backup will also send regular e-mail reports on the backup status to the customer.

There was no information on availability of the service on F-Secure web pages. In the White Paper [**32**], it was mentioned that SLA associated with the service affects the design of the physical layer of Content Enabler. This is all what could be found regarding availability.

Since Online Backup has been developed as a SaaS application from the scratch, it is natural that all key qualities of SaaS can be found in it.

3.5 .NET Framework – a Microsoft Solution

3.5.1 Overview

In the year 2000, Microsoft [**33**] announced *.NET Framework* at Forum 2000. The Microsoft .NET platform replaced the previous working title of Next Generation Windows Services (NGWS) [**34**]. The first stable version, .NET Framework 1.0, was released in February, 2002 [**35**]. Currently, the latest version of the platform is .NET Framework 4.0 that was released alongside Visual Studio 2010 in April, 2010 [**36**].

The .NET Framework today is an essential component of the Windows operating system. According to Microsoft [**37**], .NET Framework supports building and running the next generation of applications and XML web services. It is designed to fulfill the following goals:

- To provide a coherent object-oriented programming environment, in which object code is stored and executed locally, executed locally but distributed on the Internet, or executed remotely.
- To provide a code execution environment that minimizes conflicts in software deployment and versioning.
- To provide a code execution environment that improves safe execution of code, including third-party code.
- To provide a code execution environment that increases performance of scripted or interpreted environments.
- To make the developer experience consistent across Windows-based applications, web-based applications, and other varying types of applications.
- To build all communication on industry standards so that code based on the .NET Framework can integrate with any other code.

The main components of the .NET Framework are the common language runtime and the .NET Framework class library. The former is the foundation of the .NET Framework. It manages memory, thread execution, code execution, code safety verification, compilation, and other system services. Code that targets the runtime is known as managed code, while code that does not target the runtime is unmanaged code. The managed environment of the runtime (and its automatic memory

management) eliminates the most common application errors: memory leaks and invalid memory references. The latter component, the class library, is an object-oriented collection of reusable types that integrate with the common language runtime. Managed code can derive functionality from the types of the class library. In addition to common tasks (like string management, data collection, database connectivity, and file access), the class library includes types supporting specialized development scenarios: console applications, Windows GUI applications (Windows Forms), Windows Presentation Foundation (WPF) applications, ASP.NET applications, web services, Windows services, service-oriented applications using Windows Communication Foundation (WCF), and workflow-enabled applications using Windows Workflow Foundation (WF). [**37**]

3.5.2 Connection with Cloud Computing

Steve Ballmer [**33**] already used the word "cloud" when he was talking about .NET in his speech in 2000. He said "What is .NET? .NET represents a set, an environment, a programming infrastructure that supports the next generation of the Internet as a platform. It is an enabling environment for that. It is also, though, and Bill [Gates] made the analogy, I think, with Windows here pretty well for its day, .NET is also a user environment, a set of fundamental user services that live on the client, in the server, in the cloud, that are consistent with and build off that programming model. So, it's both a user experience and a set of developer experiences, that's the conceptual description of what is .NET."

In an article by John Markoff [**38**] published in April 2001, the term "cloud" is used again when Markoff states the following: "For Microsoft, the idea behind .NET is software programs that do not reside on any one computer but instead exist in the "cloud" of computers that make up the Internet."

Cusumano [**39**] observes that Microsoft has been preparing for SaaS and cloud computing for years. Recently, Microsoft has created the Windows Azure cloud platform to compete with Amazon and Google. In this journal article published in April 2010, Cusumano says that it seems that with Azure, application developers should be able to use various programming languages, not just the .NET environment. They

should be able to integrate features from other web services platforms, too. However, Microsoft is packaging its own online services and products into Azure. Windows Live, Office Live, Microsoft SQL Server services, .NET services, and SharePoint services will be available, for example. Thus, Microsoft is creating an alternative to the packaged software.

Today, it seems to be clear that .NET Framework is a major part of Microsoft's cloud strategy. As discussed above and in Section 2.6.2, .NET is one of the supported technologies when developing applications for Windows Azure.

On the Windows Azure web page, Microsoft [**14**] points out reasons why use Windows Azure. One of the supporting reasons is that a developer can use existing skills with Visual Studio and the .NET Framework to build cloud applications. Also, responding to customer needs is fast, because developers can get started quickly with familiar development tools (Visual Studio). In addition, it is possible to debug and test applications locally before they are deployed to the cloud.

Because .NET Framework is an exceptional case in this chapter, there will be no discussion on SaaS qualities.

4 SOFTWARE AS A SERVICE VERSUS ON-PREMISES SOFTWARE

Järvi et al. [**40**] do a SWOT analysis (Strengths, Weaknesses, Opportunities, and Threats) on SaaS in their book (p. 12–15). The analysis is done from the customer's and the provider's point of view separately. The SWOT analysis of SaaS is done comparing SaaS to the traditional on-premises model (see Section 4.3). Next, the qualities of SaaS presented by Järvi et al. are listed. The strengths and the opportunities listed below can be regarded as advantages of SaaS. On the other hand, the weaknesses and the threats are disadvantages of SaaS. What follows is a comparison between these observations by Järvi et al. (in *italics*) and the thoughts presented in Chapter 3 (i.e., Sections 3.1.2, 3.2.3, 3.3.3, and 3.4.3). There will be some references to Sections 3.2.2 and 3.4.2, too.

4.1 SaaS Customer's Point of View

4.1.1 Strengths

CS1 *The service is centrally hosted, which denotes that local IT is less needed for installation and maintenance in the customer company.*

This is in line with my discussion in Section 3.1.2. I mentioned that an organization may save in IT costs if it changes from a commercial on-premises system to Google Apps. I also talked about installation, updates, and backups. In medium and large-sized organizations there will likely be an IT department that is responsible for these actions. Also in Sections 3.2.3, 3.3.3, and 3.4.3, the services in question were centrally hosted. It was pointed out that installation of the standalone BLAST (Section 3.2.3) was not straightforward, which supports a use of AzureBlast, i.e. the SaaS version of BLAST. For NASA (Section 3.3.3) it is a benefit that it is not required to install NTTS on every computer, because the number of desktops at NASA is substantial.

CS2 *Applications used over the network are accessible anywhere and anytime, typically with a browser.*

This benefit is also clearly identified in Section 3.1.2 where it was discussed that in the SaaS model, the application is accessible anywhere there exists an Internet connection, and it is accessed through a thin client. Thus, any client device can be used. Also Sections 3.2.3 and 3.3.3 highlighted this as an advantage. F-Secure Online Backup (Section 3.4.3) is a little bit different case, since its purpose is to backup the content of a particular computer, and it comes with a locally installed client application that works as a user interface. However, F-Secure also provides a web page through which the customer can access his backup files from any device. In conclusion, all the SaaS implementations presented in Chapter 3 can be accessed with a web browser.

CS3 *The pay per use model eliminates high up-front costs.*

Up-front costs were not directly mentioned in Section 3.1.2, 3.2.3, 3.3.3, or 3.4.3, but the pay per use quality of SaaS was deliberated. However, up-front investments – or lack of them – were already mentioned in the characteristics of ASP in Section 2.2. In Section 3.1.2, it was stated that Google Apps costs $50/user account/year (the annual pricing plan) or $5/user account/month (the flexible pricing plan) so there are no high up-front costs. In Section 3.2.3, there was an indirect reference to reducing high up-front costs when it was stated that the user does not need to invest in expensive PC hardware, because all the heavy computing is done in the cloud. For NASA, the SaaS implementation is a way to reduce costs, as said in Section 3.3.3. In Section 3.4.3, the pricing of F-Secure Online Backup was discussed. If the customer purchases the service through the F-Secure web page, he has to pay for it for one year at a minimum. Thus, eliminating high up-front costs is not realized that well. The situation is different if and when the customer can pay for the service on a monthly basis when he buys the service through an ISP.

CS4 *The introduction is fast due to missing installation phase. SaaS often lacks a customization phase, too.*

My thoughts in Section 3.1.2 only included that there is no installation at all in SaaS, and a customer can start using the service at any time he wants. Some sort of customizability was found in Sections 3.1.2, 3.2.2, and 3.4.3. In addition, service customizability was listed as one of the key characteristics of SaaS in Section 2.3. This is at variance with the statement by Järvi et al. Now, it should be asked what is meant by customizability. Perhaps a better word for this concept would be configurability. While customizability would mean that an application can be tailored for the customer based on his specific needs, configurability would imply that the customer can do some less significant actions like putting his logo and own colors to the user interface of the application. As mentioned in Section 3.3.3, configurability of NTTS could not be analyzed, because there was not enough information available.

CS5 *Risks of purchase are low, because it is possible to evaluate the application for free and starting investments are anyway low.*

The low risks were not highlighted in Chapter 3, but they were indirectly discussed in Section 3.1.2, for example. Thanks to the flexible pricing plan of Google Apps, the customer can stop using the service whenever he wants. Starting investments are low, because Google Apps with the annual pricing plan costs $50/user account/year and with the flexible plan $5/user account/month. One thing I did not mention in Section 3.1.2 is a possibility to try Google Apps for 30 days. This free trial allows evaluating the service with up to 10 users. It was not mentioned in Section 3.4.3, either, but it is possible to try F-Secure Online Backup for 90 days with a limited storage space of 2 GB. AzureBlast, discussed in Section 3.2.3, does not have a pricing model, because it was created for evaluation purposes from the

beginning. As stated in Section 3.3.3, there were no details available on NTTS regarding the pricing.

CS6 *The service will scale up and scale down; it is easy and fast to increase the number of users. Also, the costs will go down when the usage goes down.*

As mentioned in Section 3.2.2, this is important in order to manage workload bursts. AzureBlast has a dispatch queue that allows dynamic scaling of worker instances based on the length of the queue. If the number of users increases, the dispatch queue will grow respectively when jobs are registered from new users. On the other hand, the service will scale down by limiting the number of worker instances when there are less users or jobs. In the case of Google Apps (Section 3.1.2), the costs are proportional to the number of users. In the flexible pricing plan, there is no contract term so the customer only pays for the number of user accounts in every month. When the customer limits the number of user accounts, the costs will go down accordingly. In Section 3.3.3, it was said that although NTTS seems to be easily extended, there was no information available regarding how it will rescale the resources according to customer's actions. F-Secure Online Backup, discussed in Section 3.4.3, was stated to have ensured scalability thanks to the virtualization layer of F-Secure Content Enabler. It supports unlimited number of users, in principle. The virtualization layer optimizes bandwidth and caches requests.

CS7 *There is no need to fight with version updates anymore, because only one version of the application exists, and the service provider is responsible for updating it.*

Installation, updates, and multi-tenant architecture of Google Apps were discussed in Section 3.1.2. It clearly is an advantage that the service provider does updating instead of customers. This saves a lot of effort from the customers. The laborious installation of standalone BLAST was discussed in Section 3.2.3. This goes for updates, too. As said, possible

updates for AzureBlast will be applied automatically in the background. In Section 3.3.3, it was mentioned that analysis of the data managed by provider quality of NTTS could not be done due to lack of information. In Section 3.4.3, I pointed out that Online Backup frees ISPs from setup, management, and maintenance of the service, because it is fully hosted by F-Secure. Online Backup always stays up-to-date.

CS8 *Technical data security is possibly better, because it is taken care of by the service provider, and secure hosting is part of provider's core competency.*

My discussion on security of Google Apps in Section 3.1.2 supports this. Security is built into Google Apps from the very beginning. The security aspect did not come up in Section 3.2.3 or 3.3.3. In Section 3.4.3, it was stated that the application layer of F-Secure Content Enabler includes encryption and virus cleaning. Related to file management, it also provides feature-based access rights and user profile management. In Online Backup, files are secured (encrypted) in transit and in storage. It would be very odd if a company that provides anti-virus and computer security software did not have security in its services.

4.1.2 Weaknesses

CW1 *There is less control over the service, because the same application serves all customers.*

Although there is some sort of customizability, or configurability, in Google Apps (discussed in Section 3.1.2), this weakness is a fact. A SaaS solution usually has to serve as many customers as possible, which will lead to a compromise. Some configurability could also be found in AzureBlast (Section 3.2.2) and F-Secure Online Backup (Section 3.4.3). However, configurability of NTTS (Section 3.3.3) could not be analyzed due to lack of information. CW1 was not discussed in Sections 3.1, 3.2, 3.3, and 3.4.

CW2 *The SaaS application is not tailored for customer's needs. There is sometimes customization that is done by the customer, however.*

In my opinion, this weakness is the same as CW1. Once again, the service cannot be tailored for every customer, because that would break the idea that SaaS will allow cost savings in research and development and maintenance. See also PO5 below.

CW3 *It is difficult to customize the service if customer's needs change; the service will develop only based on the needs of the majority.*

In Section 3.4.2, it was stated that one of the success factors in the transformation at F-Secure was close collaboration with the premier customers. In that case, the most important customers represented the majority. CW3 was not discussed in Sections 3.1, 3.2, and 3.3.

CW4 *The customer is dependent on the service provider. In the on-premises model, an application that is once purchased remains unchanged, but this does not apply to SaaS. The service will develop or end based on service provider's actions.*

This is a good point from Järvi et al. In SaaS, it is not possible to continue using an old version of the application, since it is always updated to the latest version automatically. In the worst case, the service may be suddenly closed. CW4 was not discussed in Sections 3.1, 3.2, 3.3, and 3.4.

CW5 *Contracts are ready-made and the provider does not have much will or possibilities for client specific contracts.*

This is likely true for mainstream customers. However, if close collaboration is so important with the premier customers (see CW3), why would it not be possible to tailor contracts for them? Of course, that would mean more costs for the service provider, but it could be worth it, because

the premier customers may bring the service provider a major part of the income, depending on the type of SaaS. CW5 was not discussed in Sections 3.1, 3.2, 3.3, and 3.4, but there was discussion on SLAs in Sections 3.1.2 and 3.2.3. SLAs could not be studied in Sections 3.3.3 and 3.4.3, because the needed information was not available.

CW6 *In self-service SaaS, the customer may be one of the thousands so there will not be one-to-one communication between the service provider and the customer*.

This is without a doubt true. The customer who decides to invest in self-service SaaS probably knows this, anyway. The service is simple enough and very cheap, typically. CW6 was not discussed in Sections 3.1, 3.2, 3.3, and 3.4.

CW7 *It is not possible to use the service offline; the service is available only over the Internet.*

In most cases, the service cannot be used offline. However, Google Apps, for instance, includes support for offline Gmail. The user can access his e-mail even when he does not have an Internet connection. Offline Gmail downloads a local cache of user's e-mail to his computer. It is possible to compose, search, and organize messages. This feature was not discussed in Section 3.1.2, but it was mentioned that there might be an interruption in service due to a break in the Internet connection. Otherwise CW7 was not speculated in Sections 3.2, 3.3, and 3.4.

4.1.3 Opportunities

CO1 *Supply is wider, because the total cost of services have decreased, which has expanded the number of applications available.*

This was not brought out in Chapter 3, but the tendency can be seen. The

number of SaaS applications available is increasing. During the last few years, many new services have appeared in the market.

CO2 *The customer can focus more on its core competence; purchasing and maintaining applications needs less effort from the customer.*

A decreased need for installation and maintenance from the customer was discussed in Sections 3.1.2, 3.2.3, 3.3.3, and 3.4.3. Otherwise CO2 was not particularly highlighted in Chapter 3.

CO3 *The service provider experiences cost efficiency and is able to offer the service cheaper. Thus, the total cost of service utilization will often be lower in SaaS.*

Low cost of service utilization was discussed in Sections 3.1.2 and 3.4.3. Also in Section 3.3.3, it was stated that the SaaS implementation is a way to reduce costs. AzureBlast discussion (Section 3.2.3) did not include cost related speculation, because the service in question was a non-commercial evaluation project. See also PS2 and PO5 below.

CO4 *It is usually possible to evaluate the service for free, and if it is not possible, the customer needs to pay only for a short period of time to test the service.*

See CS5 and CS3 above.

CO5 *Vendor lock-in is weaker; pay per use based pricing lowers the barrier to stop the unwanted service. It is still not easy to change to another service.*

The pay per use model was discussed in CS3. Vendor lock-in was not mentioned in Sections 3.1, 3.2, and 3.3. CO5 is the result of CS3 and CO3 above. On the other hand, putting your data in the service provider's system creates a lock-in, because the data in that specific system will be stored in the provider's own format. Actually, this was mentioned in Section 3.4.3

where I said that when the ISP is able to sell the backup solution to a customer, his data is tied to the ISP and it will be less attractive to change to a competing ISP.

CO6 *Self-service works well. When a service operates on a self-service principle, the provider is forced to invest in self-service. Application's purpose of use and features of the application are clearly highlighted.*

Google Apps (Section 3.1) and F-Secure Online Backup (Section 3.4.3) are good examples of self-service SaaS, when thinking from the customer's point of view. In other words, cooperation between F-Secure and ISPs is not self-service, but Online Backup is self-service for the end-user. Discussion in Sections 3.1 and 3.4.3 supports CO6; purpose of use and features are clearly underlined. AzureBlast (Section 3.2) and NTTS (Section 3.3) are left out of this, since they are not commercial products available to the general public.

CO7 *There is a high quality-price ratio due to cost efficiency experienced by the service provider and competition.*

Cost efficiency experienced by the provider is discussed in PS2 and PO5 below. Competition was mentioned in CO1.

4.1.4 Threats

CT1 *Service providers do not do integration projects for their customers; it is more difficult to buy integrations.*

This is probably true in most cases. SaaS integration requires that the integrator takes into account both functional and non-functional requirements, which creates many challenges, as pointed out by Sun et al. [**41**]. Because of these challenges, service providers do not want to offer integration projects, as they would be very laborious. CT1 is the result of CW1 and CW2. CT1 was not discussed in Sections 3.1, 3.2, 3.3, and 3.4.

CT2 *Trust in the provider and security; the service provider is responsible for general functionality of the service, data preservation and security, and the customer cannot see the facts that affect them.*

I stated in Section 3.1.2 that it is essential that the customer can trust the provider. Also security aspects were discussed in Section 3.1.2. In addition, one drawback for Google's reputation regarding privacy was mentioned. As said in CS8 above, the security perspective did not come up in Section 3.2.3 or 3.3.3. Especially for a company like F-Secure, customer's confidence in the service provider is extremely important, because security is company's core competence. Then, I want to present one important example. On April 21st, 2011, there was a large crash in Amazon Web Services. The crash took down thousands of web sites and also permanently destroyed many customers' data [**42**]. Something obviously went wrong when Amazon's mission-critical and "bomb-proof" systems crashed. How is it possible that the company permanently lost some of its customers' data? Amazon has been selling its cloud service with the promise that customer's data is always backed-up and the back-up is automatically replicated at multiple independent locations. In addition, the company has promised the 99.9 % uptime. Now, both promises were broken. This outage could be a disaster for companies who trusted their business and data in the hands of Amazon. The customers of these companies, who went down along with the crash, were not able to access the service they were paying for. However, the data loss is probably even more serious issue in this case, because some data was destroyed for good.

CT3 *Legal issues; there may be limitations regarding e.g. the physical location of data that prevents the use of cloud services.*

This is a possible threat. CT3 was not mentioned in Chapter 3, but some cloud platforms offer a possibility to choose a geographic location where (which datacenter) the data is stored. Windows Azure and locations were

discussed in Section 2.6.2 and Amazon EC2 and geographic locations were discussed in Section 2.6.3.

CT4 *The service provider may develop the service into a direction that is not favorable to the customer, or, it may entirely stop the service. In such a case, the customer is forced to seek for substitute service or even change its policy.*

See CW4 above.

CT5 *The lack of local IT support may turn out a problem; in SaaS, support is likely provided only remotely, in English.*

This was not discussed in Sections 3.1, 3.2, 3.3, and 3.4. The language may be a problem between the support person of the provider and the person in the customer company. Everyone does not speak fluently English. Support provided remotely can be a great help. Thus, I do not find that as a big threat.

CT6 *Service interruption is a risk in the SaaS model. The customer has to decide whether to trust in the service provider.*

Service availability of Google Apps was discussed in Section 3.1.2. Also availability of AzureBlast was discussed in Section 3.2.3. The providers promise availability of 99.9 % of the time, but is it enough? Service interruption is a risk in SaaS. Availability could not be discussed in Sections 3.3.3 (NTTS) and 3.4.3 (Online Backup) due to lack of information. See also CT2 above.

4.1.5 Additional Analysis on Customer's Viewpoint

Sections 4.1.1 through 4.1.4 covered the pros and cons of SaaS presented by Järvi et al. These sections included comparative analysis on the SWOT analysis by Järvi et al. and my own findings in Chapter 3. This section adds some analysis to the customer's point of view.

Satake [**43**] presents four concerns or problems that customers should take into account when utilizing SaaS. These concerns are presented in Section 4.1.5.1 (in *italics*).

4.1.5.1 Concerns for the SaaS Customer

C1 *There may be trade-offs between enjoying SaaS benefits and satisfying business requirements. If it is assumed that SaaS cannot meet all the business functional requirements, the customer's organization needs to have individuals who can decide the pros and cons of applying SaaS from the business point of view.*

This is possibly related to CW1, CW2 and CW5 above in Section 4.1.2. A SaaS solution may not meet all the business functional requirements, because there is no tailoring for customer's needs. When commenting CW1, I mentioned that serving as many customers as possible will lead to a compromise. C1 supports this statement.

C2 *The overall service level may drop if multiple services are used in combination. Customers should perform a review of how the use of multiple providers or multiple services in combination may affect the performance, support, and general service level. Also integrity verification of the total service level should be performed.*

This is a new concern that was not highlighted in Sections 4.1.1 through 4.1.4, nor was it highlighted in Chapter 3. Anyway, this may be related to CT1 above in Section 4.1.4. Since service providers do not do integrations for their customers, utilizing multiple services in combination can have some kind of effect on the performance.

C3 *Quality itself will not necessarily be improved by utilizing SaaS. When considering utilizing SaaS, the customer must think about the expense, time, and personnel needed to obtain service functions and quality equivalent to that of internally developed services.*

In CS1 it was stated that local IT is less needed for installation and maintenance. This means that fewer employees are needed in the IT department of a customer company. In addition, CO3 said that the total cost of service utilization will be lower in SaaS. Moreover, CO2 stated that the customer can focus more on its core competence, because purchasing and maintaining applications requires less time from the user. Therefore, there is a small contradiction between C3 and discussion in Sections 4.1.1 through 4.1.4.

C4 *Business might be interrupted if the SaaS provider withdraws. When the customer investigates possible SaaS providers, he should include the financial condition of the provider in the selection criteria. Also alternatives should be studied.*

This supports CW4 and CT4 above in Section 4.1.2. It is a real risk for the customer that the SaaS provider may suddenly stop the entire service, if it goes to bankrupt.

4.1.5.2 Benefits for the SaaS Customer

Armbrust et al. [**44**] add one more benefit to the list of SaaS advantages: SaaS allows data sharing and collaboration more easily. In Section 3.1.1, it was stated that Google Apps is composed of collaboration and messaging applications. It also includes sharing features. In Section 3.4.3, it was discussed that F-Secure Online Backup has a new feature that enables sharing backed up files with friends, family, and coworkers. It was also pointed out that this would not be achievable without SaaS. These two examples from Chapter 3 support the statement made by Armbrust et al. This quality was not discussed in Sections 4.1.1 through 4.1.4, but it is certainly related to CS2 in Section 4.1.1. Other benefits of SaaS stated by Armbrust et al. were already covered in Sections 4.1.1 through 4.1.4 and Chapter 3.

Sääksjärvi et al. [**45**] list benefits and risks for the SaaS customer. The lists are based on the articles they have reviewed for their paper. The following benefits and risks, or

disadvantages, are direct quotations (in *italics*). First, the benefit issues for the customer are listed.

CB1 *"SaaS enables the customer to focus more on core competencies"*

This supports CO2 above.

CB2 *"SaaS makes it easier and/or less costly to get access to required technical expertise"*

I am not sure what this means. This would require a further description.

CB3 *"The system implementation time is shorter with SaaS"*

This supports PS5 and PO5 that will also benefit the customer.

CB4 *"SaaS enables a wider and more flexible array of payment methods (predictable and/or lower costs)"*

This supports discussion in CS3 above.

CB5 *"SaaS makes version management easier for the customer (free upgrades, no technology obsolescence etc.)"*

This supports CS7 above.

CB6 *SaaS provider aggregates software applications from several sources and builds a complete service offering.*

This is about the reusability quality of SaaS. Reusability was discussed in Chapter 3.

CB7 *"SaaS enables the customer to get access to "best-of-breed" applications that would be too expensive to buy"*

This was not highlighted in the SWOT analysis by Järvi et al., but the case of Section 3.2 shows that cloud computing democratizes technology, because researchers, or even a single researcher, can have access to the large-scale compute resources that until recently were available only for well-funded organizations. Similarly, SaaS democratizes technology by enabling the customer to have access to state of the art applications.

CB8 *"SaaS makes it possible to access the software independently of location and time"*

This supports CS2 above.

CB9 *"The initial/investments and costs are much lower in SaaS"*

This supports CS3 and CS5 above.

CB10 *"With SaaS, the customer can get access to a superior IT infrastructure regarding reliability, security and scalability"*

This supports CS6 and CS8 above.

CB11 *"SaaS broadens the selection of potential applications available to the customer"*

This supports CO1 above.

CB12 *"SaaS enhances the available customization options of applications to the customer"*

There is a contradiction between this and CW1 and CW2 above. Perhaps the word “customization” here denotes configurability.

Next, the risk issues for the customer presented by Sääksjärvi et al. [**45**] are listed in Section 4.1.5.3.

4.1.5.3 Risks for the SaaS Customer

CD1 *“There are less tailoring and integration options available for the customer”*

This supports CW2, CW1, CW5, and CT1 above.

CD2 *“SaaS increases the risk of losing business-critical data or exposing it to third parties”*

This supports discussion in CT2 above.

CD3 *“Availability, reliability and performance-related issues are to be expected, depending on the technological solution of the SaaS provider”*

This supports CT2 and CT6 above. However, performance-related issues were not discussed in Sections 4.1.1 through 4.1.4.

CD4 *“In exchange for the lower price, the customer is typically bound with a long-term contract (switching costs)”*

This is a little bit at variance with CS3 and CO5 above. Of course, it may be dependent on the SaaS application in question, but some providers offer payment method on a monthly basis.

4.1.6 Summary

I find the lists, i.e. the SWOT analysis, by Järvi et al. rather extensive and apt. My own findings support the statements by Järvi et al. There were a couple of qualities in the

lists of Järvi et al. that I found to be duplicates. E.g., CW2 is the same as CW1. Also CO4 means almost the same as CS5. Finally, CT4 is the same as CW4, in my opinion. There were some qualities that did not get support from my discussion in Chapter 3. However, in most cases I agree with them, although they do not appear in the implementation cases of Chapter 3. In the case of CW5, I disagree with Järvi et al. at some level, but I also understand their point. In Section 4.1.5.1, at least one new concern or threat was presented. Satake stated in C2 that the overall service level may drop when using multiple services in combination. In addition, C1 can be a new concern. It stated that there may be trade-offs between enjoying SaaS benefits and satisfying business requirements. One concern, C3, presented by Satake conflicted with the lists of Järvi et al. In this case, I will be for the statements by Järvi et al. Finally, Armbrust et al. were able to add one more benefit to the current list: SaaS allows data sharing and collaboration more easily. As clear as it seems now, it was missing from the list of benefits. Finally, the list of benefits and risks by Sääksjärvi et al. were presented. As it could be seen, almost all of the benefits in the list in Section 4.1.5.2 supported the SWOT analysis by Järvi et al. CB12 was the only benefit that conflicted with the analysis presented in Section 4.1.2. CB6 and CB7 supported discussion done in Chapter 3. CB2 remained unclear. In Section 4.1.5.3, the majority of the benefits in the list supported the SWOT analysis by Järvi et al. Only CD4 conflicted with the analysis presented in Sections 4.1.1 and 4.1.3. After these additions, I do not have anything else to add at this point.

4.2 SaaS Provider's Point of View

Next, SaaS qualities are discussed from the service provider's point of view. Some of the strengths experienced by the customer will be repeated in service provider's SWOT as weaknesses.

4.2.1 Strengths

PS1 *There are more potential customers, because customers will see cost savings, ease, and low need for IT.*

In Section 2.3, it was stated that various consumers having Internet access

can be potential cloud service users. This is related to service configurability, because it allows the service provider to meet the different needs of customers. Configuration is done by service customers themselves. PS1 was not directly discussed in Sections 3.1, 3.2, 3.3, and 3.4. Cost savings were discussed above in CS3 and CO3, ease was covered in CS2, CS4, CS6, CS7, CS8, CO2, and CO6, and low need for IT was speculated in CS1.

PS2 *Costs per customer are low due to the multi-tenant model in which all customers use the same instance of the application. The technical expenses caused by a customer are very small and the service will be available for new customers automatically.*

In Section 3.1.2, I said that Google can provide Google Apps as cheap as it is thanks to multi-tenancy. Also in Section 3.4.3, it was discussed that F-Secure Online Backup is fairly cheap when you calculate the price per month. This is, once again, the result of the multi-tenant model. Sections 3.2.3 (AzureBlast) and 3.3.3 (NTTS) did not include discussion related to costs per customer. See also CO3 above. The quality that the service will be available for new customers automatically was covered in CO6.

PS3 *The provider is in direct contact with the customer; typically there are no third parties between the customer and the service provider, which allows fast feedback and development cycle.*

An exception to this is the case of F-Secure, in which there is an ISP between the service provider and the customer. Of course, the ISP is also F-Secure's customer, but the end-user is the real customer of the service. Or, maybe it should be said that both ISPs and end-users are the customers of F-Secure. In Section 3.4.2, it was pointed out that one challenge is to find a balance between the customer (i.e. the ISP) and the customers' customers (i.e. end-users who are customers of the ISP). In fact, half of the F-Secure personnel interviewed thought that ISPs are the most important customer

group while other half thought that end-users are the most important, or both ISPs and end-users are equally important. In addition, it was stated in Section 3.4.2 that analyzing customer feedback becomes more complex when the customer base increases. Typically, like in the case of Google Apps (Section 3.1), there are no third parties involved. PS3 was not discussed in Sections 3.1, 3.2, and 3.3.

PS4 *Version control and maintenance are simpler; maintaining several software versions is laborious, especially if versions are tailored for customers. Also, the need to support many versions disappears.*

Maintenance and multi-tenancy were discussed in CS7 above. In Section 3.1.2, I stated that running a single, multi-tenant instance is a benefit for the provider, because it can concentrate on keeping this one process running. It is also probable that one instance will be cheaper than thousands or millions of instances. I also said that Google's infrastructure is designed in a way that it enables delivering improvements or new functionality to its entire customer base, on short iteration cycles. Also Sections 3.2.3 and 3.4.3 covered maintenance. Section 3.3.3 did not include analysis of this kind due to lack of information. Version control was not discussed in Sections 3.1, 3.2, 3.3, and 3.4, but it is obvious that version control is simple for the provider thanks to the multi-tenant model. There is only one software version to control at a time. See also PO5 below.

PS5 *Development process is simple, because there is only one version to maintain. There is no need to prepare for incompatibilities of different versions. It is also easy to deliver minor updates thanks to the delivery model of SaaS.*

This is the result of PS4. Development process was not covered in Sections 3.1, 3.2, 3.3, and 3.4. Anyway, benefits of one version to maintain were discussed in PS4 and CS7 above.

PS6 *Sales cycle is fast; the actual purchase transaction can be automated and it is very quick.*

Introduction of Google Apps was discussed in Section 3.1.2. F-Secure Online Backup, discussed in Section 3.4.3, can be purchased through an ISP or F-Secure's web page. In all cases (Google Apps and Online Backup), the purchase is done in the web so it can be automated and made fast. Sections 3.2.3 (AzureBlast) and 3.3.3 (NTTS) did not include speculation related to purchasing the service, because AzureBlast was a non-commercial project and in the case of NTTS there was not enough information provided in the source article.

4.2.2 Weaknesses

PW1 *Cash flow; expenses related to customer acquisition will be paid a long after the purchase transaction, and only if customership lasts. Flexible, scalable pricing is good for the customer, but problematic for the service provider.*

This was one of the strengths experienced by the customer: CS6 above. PW1 was not discussed in Sections 3.1, 3.2, 3.3, and 3.4. Also CS5 is related to PW1, because it allows free evaluation of the service.

PW2 *Global competition is hard; quality requirements are high and prices are low, especially if the service provider operates mainly in the web and on a self-service principle.*

This, for one, was experienced by the customer in CO1 and CO7. Global competition was not discussed in Sections 3.1, 3.2, 3.3, and 3.4.

PW3 *Customer Acquisition Cost (CAC) relative to purchase price is high. It may take months for the service provider to cover the expenses of customer acquisition, if the service requires selling.*

See PW1. PW3 was not discussed in Sections 3.1, 3.2, 3.3, and 3.4.

PW4 *Lock-in is weaker; high CAC, pay per use based pricing, and short-term customership may lead to service provider's bankruptcy.*

This was experienced by the customer as an opportunity in CO5 above. Pay per use based pricing was one of the strengths for the customer: CS3.

4.2.3 Opportunities

PO1 *The business model is scalable; the SaaS model will scale to mass market. The most potential bottleneck in this is sales.*

This was not discussed in Sections 3.1, 3.2, 3.3. However, PO1 is the likely reason why F-Secure decided to transform its business from software products to software services, as discussed in Sections 3.4.1 and 3.4.2. It was stated that SaaS became the company's main business area already in 2005. In 2008 (Q4), the split of SaaS business had grown to 44 % of the total revenues. In 2010, F-Secure's SaaS business through ISPs, mobile operators and cable operators still continued growing and represented 52 % of the total revenues [**46**].

PO2 *A multi-user application whose self-service introduction is very easy enables viral marketing.*

This was not discussed in Sections 3.1, 3.2, 3.3, and 3.4. Anyway, today it is surely possible that viral marketing can boost a SaaS service when people utilize the Internet and social networks more than before.

PO3 *Fast access to the global market; an application that is accessible in the web is global, and a service that utilizes viral marketing can grow exponentially.*

This is related to PS1; various consumers having Internet access can be potential cloud service users, as stated in Section 2.3. On the other hand, competition in the global market is hard (see PW2 above). PO3 was not discussed in Sections 3.1, 3.2, 3.3, and 3.4.

PO4 *The service provider can start a business with low risks. Building a SaaS may not require any up-front costs, if salaries are excluded. The costs will come along large number of customers or wide marketing campaigns. SaaS fits those who want to bootstrap a startup.*

This is an interesting point. Also the provider can give a try with low risks in the same way as the customer can try or purchase a service with low risks, as stated in CS5 above. PO4 was not discussed in Sections 3.1, 3.2, 3.3, and 3.4.

PO5 *There will be cost savings in software development and maintenance due to the multi-tenant model.*

It was shortly mentioned in Section 3.3.1 that the GXD Framework API, or the driver library of GXD Framework, does not require installation, but it can be automatically downloaded on-demand. This brings cost savings to software development and maintenance (database administration). Cost savings in software development were not directly discussed in Sections 3.1, 3.2, and 3.4. Of course, the multi-tenant model of SaaS enables savings in software development, because there is only one version to be developed at a time. Nor are there tailored versions for different customers. The developers always know that the customer has the latest version of the application. There is no need to think about the compatibility of different application versions. This makes support easier, too. As stated in PS4 above, maintenance and multi-tenancy were discussed in CS7.

PO6 *Feedback cycle is faster, because fast upgradability together with direct customer contacts allows rapid feedback on the new features, and thus makes agile development possible.*

This was not directly discussed in Sections 3.1, 3.2, 3.3, and 3.4. Anyway, fast upgradability is the result of the multi-tenant architecture. In Section 3.1.2, I mentioned that Google is able to launch upgrades or patches to all its customers on short iteration cycles. Also AzureBlast (discussed in Section 3.2.3) and F-Secure Online Backup (discussed in Section 3.4.3) can be quickly updated. In Section 3.4.2, it was said that F-Secure had the service development team analyzing the needs of the premier customers (i.e. ISPs). This close collaboration enabled rapid feedback.

PO7 *User community can be utilized and possibly integrated into the service, because the service provider operates in the web.*

This was not discussed in Sections 3.1, 3.2, 3.3, and 3.4. However, it seems that e.g. Google and F-Secure utilize Facebook to better communicate with consumers.

4.2.4 Threats

PT1 *Customers may be suspicious of data security and reliability. The question is "Will customers trust the service provider enough so that they leave their business critical data to the service provider?"*

One of the strengths experienced by the customer was CS8; technical data security is possibly better. Nevertheless, this can be a threat for the service provider. This, PT1, is actually the same as CT2 above. As discussed in CT2, the Amazon cloud crash showed that there are pitfalls in the cloud and thus in SaaS.

PT2 *Product/service can be copied fast, because in SaaS business competition is global and implementing a competing product is typically faster than implementing the original product.*

This was not discussed in Sections 3.1, 3.2, 3.3, and 3.4. Global competition was one of the weaknesses experienced by the provider: PW2 above. A service may be copied fast, because anyone on the Internet can access a new service that appears in the market. The new service can be investigated, and if it seems promising, it is possible to steal the idea behind the service.

PT3 *Value of customership goes down. This is a problem particularly if the provider tries to transform an existing product to the SaaS model.*

This is the result of CO4 and CO5 above. On the other hand, it was stated in Section 3.4.2 that close collaboration with the premier customers is essential in new service development when transforming from existing software products to software services. This implies that customership still has value. PT3 was not discussed in Sections 3.1, 3.2, and 3.3.

PT4 *SaaS requires its own kind of approach and way of thinking. Procedures of the on-premises model will not always work well in SaaS. The provider may be able to gain a few large customers, but it is a different matter to sell the service to a huge clientele at a cheap price with viral marketing.*

In Section 3.4.2, it was highlighted that cultural change from software products to software services can be slow and demanding. Many employees still saw F-Secure as a software product company. The biggest issue for the company was the cultural change towards services, at least according to one of the interviewees. The cultural change requires a new way of thinking from the employees. Each employee needs to understand that the company provides services, not just software and software features. This includes also the top management of the company, of course. As said, the key personnel

have to have great confidence in the services. PT4 was not discussed in Sections 3.1, 3.2, and 3.3.

PT5 *Research and development (R&D) in SaaS differs from on-premises R&D. Technical differences between on-premises and SaaS products are not that essential, but e.g. utilizing cloud infrastructure may be new to a team who has worked in on-premises development.*

This was not directly discussed in Sections 3.1, 3.2, 3.3, and 3.4, but in Section 3.4.2, it was stated that a dedicated team is vital in service development when transforming from software products to the SaaS model. In Section 3.5.2, it was pointed out that a developer who is familiar with Visual Studio and .NET Framework can use his existing skills to build cloud applications on Windows Azure. It is also possible to debug and test applications locally before deploying them to the cloud. However, utilizing cloud infrastructure (see Section 2.6) may require some learning, for sure.

4.2.5 Additional Analysis on Provider's Viewpoint

Sections 4.2.1 through 4.2.4 covered the pros and cons of SaaS presented by Järvi et al. and comparison to my own findings, the same as Sections 4.1.1 through 4.1.4. This section adds some analysis to the provider's point of view.

York [**47**] lists ten *do's* and *do not's* in a whitepaper that is based partially on his blog posts concerning SaaS business strategy. These advices, do's and do not's, are meant for SaaS providers. Next, in Sections 4.2.5.1 and 4.2.5.2, the advice by York are summarized (in *italics*). First, ten actions that a SaaS provider should do are being examined.

4.2.5.1 Actions to Do as a SaaS Provider for SaaS Success

DO1 *Choose a large market. In SaaS, volume is essential. The provider has to ask: "Is my market big enough?" The provider should focus only on those*

customers that have the highest need and are the easiest to reach. This is needed in order to achieve growth and profitability.

profit = volume x (subscription price – variable costs [acquisition, support, etc.]) – fixed cost

This advice denotes that a large market is a benefit or an opportunity for the SaaS provider. Scaling to mass market was also discussed in PO1 above in Section 4.2.3.

DO2 *Create a hub on the web. SaaS providers operate on the web and need site traffic. To get traffic, the provider needs links to its website. Links can come from various sources: links from search results, links from search ads, links from banner ads, links from websites, links from blogs, etc. The SaaS provider should have links from sources that are relevant to its business and relevant to its prospects' needs.*

This advice was not directly present in Sections 4.2.1 through 4.2.4. However, this is possibly related to PO3 above. If the provider can get plenty of links to its website, it may grow fast in the global market.

DO3 *Accelerate organic growth. This means that the SaaS provider should stimulate demand and make purchase easier by understanding and leveraging its prospect's natural buying behavior. The question to be asked is: "How can I get more prospects to find my product, evaluate it, and buy it over the web even if no one shows up for work in the morning?" To enable organic growth, the provider has to remove purchase barriers and respond on-demand with the information necessary to motivate the prospect so that he takes the next step in the buying process.*

organic growth = revenue generated with (near) zero marginal acquisition cost

This advice is about self-service and viral marketing that were discussed in PO2 above.

DO4 *Craft a compelling story. It has been claimed that people do not buy products, they buy stories. Great stories should be the basis of provider's online marketing plan, because on the web the provider has a) less time to get people's attention, b) loads of low cost, multi-media options to deliver its story, and c) built-in automation to spread its content virally assuming that it is interesting and entertaining. It is important that the story is great, not just good, nor good enough. The provider should craft a great story and, for example, put it on YouTube to get millions of hits to generate organic demand.*

This advice did not directly occur in any strength or opportunity in Sections 4.2.1 and 4.2.3. I personally think that this advice applies to many business cases, not just SaaS offerings. E.g., new smartphones, tablet computers, and video games have been marketed like this. Putting a video on YouTube to get organic demand is viral marketing that was discussed in PO2 above.

DO5 *Build the business into the product. When moving a software product online into a SaaS delivery model, it allows the provider to connect the product directly to its customers out on the web and directly to its internal systems. This enables the provider to reengineer its essential business processes when they are built out from the product. A good example of such a case is Amazon.com. Prior to the era of Amazon, it was possible to buy physical products by ordering them through a direct mail catalog. However, Amazon has introduced convenience and credibility that are derived from features of Amazon.com (i.e., the SaaS offering of Amazon): product search, offers, recommendations, one-click checkout, order management, support, etc. The automation of Amazon.com facilitates the purchase process.*

Building the business into the product was not highlighted in Sections 4.2.1 through 4.2.4. It can be a valuable benefit of the SaaS model, as the example

of Amazon shows. A purchase process, or transaction, was discussed in PS6 above.

DO6 *Reach across the firewall. The SaaS provider should look beyond the client-server application that is being replaced. The provider must connect its SaaS offering to the rest of the web to enable order of magnitude productivity gains in core business processes. Productivity gains can be found in prospect, customer, partner, community, and computer-to-computer interaction derived from integration, network effects, user-generated content, and on-demand processing and data.*

In Section 4.1.4, it was stated in CT1 that service providers do not do integration projects for their customers and buying integrations is more difficult. See also C2 in Section 4.1.5. It seems that DO6 encourages the provider to find productivity gains from integration that was one of the threats for a SaaS customer.

DO7 *Monetize creatively. In short, it is the network. It is the web. SaaS is not software. New business value comes from the quality that provider's SaaS offering can become a network hub that can connect any business entity, user or system to any other: provider's prospects, provider's customers, provider's partners, provider's customers' customers, provider's customers' vendors, provider's customers' partners' customers, and so on. Because the value is created by the network, it follows that new network-based monetization opportunities are created. The opportunities open to the provider will depend on many factors, but in the end, it will depend on provider's own creativity.*

Monetization opportunities were not discussed in Sections 4.2.1 through 4.2.4. The value that comes from the network was discussed in PO1 and PO7 above. See also DO2 above.

DO8 *Enable mass customization. To enable mass customization, setup costs must be eliminated. Setup costs include labor, time, and tooling it takes to switch a production line from one product to the other. When talking about enterprise software, each customer installation can be regarded as a product and the associated, customer specific implementation, configuration, customization, and maintenance time and effort can be regarded as the setup costs. Provider's customers will have unique requirements. This architectural requirement needs automated deployment that consumes minimal resources, extensive, easy-to-use, self-service configuration and full interoperability built on open, standards-based APIs.*

As said in CS4 above, the word "customization" here means probably configurability.

DO9 *Open up to the cloud. Software interacts with people, hardware, and other software. The SaaS revolution has been about reducing costs by combining remote and scattered groups of people over the web – software interacting with people to lower total cost of ownership (TCO). Cloud computing is about the computers talking to each other over the web – software interacting with other software, and thus other hardware. A SaaS product will not be able to stand on its own. Therefore, the provider must reach out to complementary applications on the web to complete its offering. Customers expect that provider's application slips seamlessly and effortlessly into their environments. Provider's SaaS offering has to have extensive, open, standards-based APIs to survive.*

This advice is possibly related to integrations discussed in CT1 in Section 4.1.4. On the other hand, it may be about reusability. In Section 2.3, reusability was listed as one of the key characteristics of SaaS.

DO10 *Leverage your community. Every action the provider takes should ultimately result in either increased revenue or reduced costs. Community means free, honest, loyal, and viral labor to provider's business. SaaS businesses must*

reduce costs, especially labor costs, by taking advantage of web-based automation and network effects. The provider can get improvements implemented without writing a single line of code itself if it can encourage its customers to do crowdsourcing. SaaS business must leverage the community to survive.

This advice is about utilizing user community and it was discussed in PO7 above.

Next, ten actions that a SaaS provider should not do are being examined.

4.2.5.2 Actions Not to Do as a SaaS Provider

DN1 *Chase elephants. A prospect that has lots of cash, needs special features, requires a long sales cycle, and has limiting legal requirements. Chasing this kind of a customer is difficult and out of SaaS provider's core strategy. It will result to a situation where the provider is a small consulting business constrained by the special needs of a few, powerful customers.*

This is related to PT3 above. Instead of few, large customers, it is more profitable to get lots of smaller customers. Volume is essential, as said in DO1. Actually, DN1 is the opposite of DO1.

DN2 *Waste money marketing offline. Offline marketing is typically more expensive than online marketing. In SaaS, the average lifetime revenue per customer is usually too low to cover offline marketing. All real prospects are online, because you cannot use SaaS offline.*

This was not directly discussed in Sections 4.2.1 through 4.2.4. Some forms of online marketing, such as viral marketing in PO2, were mentioned anyway.

DN3 *Launch without online trial. It is a major error to bring a SaaS offering to market without a free trial. Without online evaluation, the provider*

interrupts its prospect at the most critical phase in the purchase cycle: the phase after the provider has spent all that money generating demand.

In CS5 and CO4 above, it was said that risks are low for a customer, because it is possible to try the application for free. If online trial is left out, the customer will have higher risks and he may not want to purchase the application at all, which means revenue loss for the provider.

DN4 *Cover up shortcomings with people. In a SaaS business, direct labor is provider's enemy and automation is provider's friend. It is difficult for the provider to cover up shortcomings in its SaaS business with people. When the provider fails to automate core business processes and then covers it up with people, the provider ensures long-term unprofitability. This kind of a culture undermines long term success.*

This was not pointed out in Sections 4.2.1 through 4.2.4. Automation is crucial in SaaS business. Online trial, online purchase, bug reporting, and upgrading to the latest version should all be automated.

DN5 *Invest in channel partners too early. No one wants to resell provider's product or service if they cannot make money at it. First the provider has to convince a partner to invest time and resources building capacity that is needed to make money at the service. Usually the provider has to kick-start revenue itself to prove the business potential.*

This was not present in the previous sections either, but this is related to PW2 above: global competition is hard. It is very challenging to break through when the market is full of competitors, quality is high, and prices are low.

DN6 *Bleed cash indefinitely. Growth follows efficiency, not the other way around. A stable subscription revenue stream sounds exiting, but it is worth remembering that the requirement of long term profitability means a stable*

cost structure that is below that revenue line. No business can sustain negative operating margins. The provider cannot bleed cash indefinitely due to high acquisition and support costs. The provider must intentionally make revenue meet costs at a profitable intersection.

High CAC was mentioned in PW3 and PW1 above. It is a clear weakness, but the provider cannot rationalize wasting money with high acquisition costs forever.

DN7 *Ignore the long tail. It is in practice impossible to move down-market once the provider is locked into a high price and high-cost business model. The provider would need to simplify its product offering, reduce selling costs and frequently cannibalize revenue. If, however, the provider creates its business with a vision of profitably servicing the long tail, i.e., the market segments with the lowest average revenue per customer, then the provider will have no problem profitably servicing higher revenue segments. It is easier to increase revenue and costs than it is to reduce them.*

This quality was not discussed in Sections 4.2.1 through 4.2.4.

DN8 *Think you can control it. In online business, the provider does not have lots of face-time with a prospect. The need for control can lead good managers to make poor SaaS business decisions by favoring control over performance.*

I think this is about PT4 above: SaaS requires its own kind of approach and a new way of thinking. Although a manager could control it in the on-premises model, he may not be able to control it in the SaaS business.

DN9 *Fail to be creative. There is a lot of cheap software out there already. Why should someone pay attention to provider's search ad? Provider's application must be different. The message must be different. The provider's business must be different.*

This is closely related to PW2 above. The provider and its application must be somehow different to break through. The provider needs to be creative.

DN10 *Depend on network effects. If the provider builds an offering, there is no guarantee that anyone whatsoever will come and use it. The provider cannot build a web business on network effects alone; significant, direct value for every single customer must be created. Before thinking about using viral marketing, socializing the application and network monetization, the provider must satisfy its very first customer.*

This threat was not pointed out in Sections 4.2.2 and 4.2.4. Actually, also this one is related to PW2 above. However, it makes sense that the provider must earn its first customers. Viral marketing cannot be used from the beginning.

4.2.5.3 Benefits for the SaaS Provider

Sääksjärvi et al. [**45**] list benefits and risks for the SaaS provider. The following benefits and risks, or disadvantages, are direct quotations (in *italics*). First, the benefit issues for the provider are listed.

PB1 *"SaaS enables economies of scale in production and distribution costs (one-to-many offering)"*

This supports PS2 and PS5 above.

PB2 *"The cash flows from SaaS are more predictable than in traditional software sales"*

This is true, because the probability that an existing customer will continue using the service is higher than the probability to get a new customer. However, it should be remembered that a customer may unsubscribe at any time and it is very difficult to predict that kind of behavior. The pay per use

model was discussed in CS3 above. If the contract term is e.g. one year, then the cash flows can be better foretold. In any case, the cash flows from the customer are predictable for the customer itself, but not that predictable for the provider, in my opinion.

PB3 *"SaaS expands the potential customer base"*

This supports PS1 and PO1 above.

PB4 *"The sales cycle of SaaS is shorter than that of traditional software sales"*

This supports PS6 above.

PB5 *"SaaS lowers version management and maintenance costs"*

This supports PS4 above.

PB6 *"If supplier succeeds in integrating good product and service into a SaaS offering, this creates a barrier to entry for competitors"*

This is related to PW2 above.

Next, the risk issues for the provider presented by Sääksjärvi et al. [**45**] are listed in Section 4.2.5.4.

4.2.5.4 Risks for the SaaS Provider

PD1 *"It is difficult to manage the complex network of suppliers, which is required for integrating the product and service businesses"*

This risk is probably obsolete. Today, cloud computing incorporates the IaaS and PaaS models that makes it easier to build a SaaS offering.

PD2 *"Moving to the SaaS model initially reduces the turnover as the revenue comes from service fees instead of license sales and consultation fees"*

This is true, but DO1 above stated that volume is essential in SaaS. PD2 supports PW1 above.

PD3 *"Performance and scalability issues are to be expected, depending on the technical solution used"*

As discussed earlier, the multi-tenant infrastructure of SaaS will solve the scalability issue. Performance-related issues were not discussed in Sections 4.1.1 through 4.1.4.

PD4 *"High initial investment in starting SaaS business (Building and maintaining the IT infrastructure required and buying 3rd party software)"*

This risk must be obsolete. Nowadays, there are many cloud infrastructure services, and thanks to them, the provider does not need to build its own IT infrastructure. Actually, it was stated in PO4 that the service provider can start a business with low risks, without up-front costs. What comes to buying 3rd party software, I do not know what it refers to.

PD5 *"The customization of SaaS applications typically requires extra costs"*

This supports CS4, where it was stated that SaaS often lacks a customization phase. See also CW1 and CW2 above.

PD6 *"Requires commitment to a more frequent release/upgrade cycle"*

This is true, but this can also be an advantage. A more frequent release/upgrade cycle enables agile development, as mentioned in PO6 above. In addition, development process is simple in SaaS, because there is only one version to maintain, as said in PS5.

4.2.6 Summary

I said this in Section 4.1.6, and I have to say this again for the provider part of the analysis; I find the lists, i.e. the SWOT analysis, by Järvi et al. very apt and extensive. My own findings support the statements by Järvi et al. although my examination remains weaker. This is because the business side is not that much present in the sources of Chapter 3. In one quality, PS3, I could provide an example when the statement is not true. Anyway, in typical cases, I believe that the statement by Järvi et al. holds up. I could not support many of the cases in the SWOT analysis by Järvi et al. with my own study done in Chapter 3. However, Section 4.2.5 supports and complements the analysis done in Sections 4.2.1 through 4.2.4. York has a little bit different approach while he lists top ten "dos" and "don'ts" instead of directly listing benefits and disadvantages. The advice by York is, however, closely related to the SWOT analysis presented in Sections 4.2.1 through 4.2.4. It is possible to find opportunities, strengths, threats, and weaknesses from the advice. The lists by York also provided some new qualities. Finally, the list of benefits and risks by Sääksjärvi et al. were presented. As it could be seen, almost all of the benefits in the list in Section 4.2.5.3 supported the SWOT analysis by Järvi et al. It seems that some of the risks listed in Section 4.2.5.4 have already become obsolete, maybe because half of the source articles used by Sääksjärvi et al. were from a decade ago. After these additions, I do not have more to add.

4.3 Pros and Cons of On-Premises Software

On-premises software means a traditional way to purchase software; an organization buys software licenses, buys servers to run backend of the application, and tells the IT department to maintain the application. After initialization, the application will be used from computers on the intranet of the organization. [**40**]

In other words, on-premises software is installed and run on computers that are located on the premises, in the building, of the organization using the software. On-premises software is sometimes referred to as "shrink-wrap" software.

Pros of on-premises software can be derived from the analysis done in Sections 4.1 and 4.2. The qualities that were presented as weaknesses and threats are now the pros. Cons of on-premises software can be derived from the strengths and opportunities listed in Sections 4.1 and 4.2, respectively. Next, the pros and cons of on-premises software are presented from the customer's and the provider's point of view separately. The following lists (i.e., Sections 4.3.1 and 4.3.2) are my own reasoning and primarily based on the findings in previous sections.

4.3.1 On-Premises Software – Customer's Point of View

4.3.1.1 Pros

CP1 There is more control over the service. It is possible to get the application tailored for customer's needs. It is possible to customize the application if customer's needs change.

CP2 Once purchased application remains unchanged. The customer does not need to worry about whether the application will develop into a direction that is unfavorable. Nor does he need to be afraid of that the possibility to use the product will end suddenly.

CP3 Contracts can be tailored for customer's needs. When there is the human touch in the sales process, it is possible to negotiate for the contract.

CP4 There is one-to-one communication between the provider and the customer. With on-premises software, it is usually possible to be in contact with the provider.

CP5 The application can be used without Internet connection. Unlike SaaS, on-premises software is typically functional without access to the Internet, because all necessary application components run on the premises.

CP6 Security; when data is stored on the premises, the customer can avoid a disaster such as Amazon Web Services crash (see CT2 in Section 4.1.4) if he takes good care of back-ups.

CP7 There are no legal issues related to physical location of data like in SaaS.

CP8 Local IT support exists. At least in medium and large-sized organizations, there is always the local IT that can help when a user has a problem. The support will be given in the localized language.

CP9 Application availability is better than in the SaaS model. With on-premises software, there is no similar risk related to service interruption as with SaaS applications. Unlike with SaaS, the user of shrink-wrap software does not need to be worried about system availability or reliability as long as his workstation is functional.

CP10 According to Fan et al. [**48**], there is a performance gap between the thin client interfaces of SaaS applications and the rich desktop of shrink-wrap software. In other words, the user of the rich desktop will probably get the best user experience.

CP11 The user of on-premises software will get the full functionality. Unlike on-premises applications, SaaS thin client applications may lack the full functionality [**48**]. This may be the case if an on-premises software has been transformed from an existing product to a SaaS application.

4.3.1.2 Cons

CC1 Local IT is more needed for installation and maintenance. Since on-premises software is run on the premises, the customer needs to handle installation and maintenance of the application. The natural way to do this is to utilize local IT.

CC2 On-premises applications cannot be accessed anywhere with any device. Typically, the user has to be in the building of the organization to use the application. However, some on-premises applications may be accessible

outside organization’s network if the user first establishes a VPN connection.

CC3 Costs are higher than in SaaS. On-premises software cannot benefit from the multi-tenant model of SaaS and thus on-premises applications cost more. In addition, the pay per use model is not utilized so up-front costs are usually high.

CC4 The introduction is slower than in SaaS. The application has to be installed on the premises and there might be a customization phase.

CC5 Risks of purchase are higher than in SaaS. There may not be a free evaluation period with on-premises software. In addition, starting investments are higher than in SaaS.

CC6 On-premises applications are not as scalable as SaaS applications. With an on-premises application, it is not possible to increase the number of users forever, because the architecture of the application will likely not allow it.

CC7 On-premises applications require updating. The customer needs to take care of installing updates. It might not be trivial, because updates may not always be installed on top of every existing application version. In other words, updates are usually compatible only with certain versions.

CC8 Supply is narrower. This may not be a problem yet, but in the future, when SaaS continues to grow, the number of on-premises software available will be likely decreased.

CC9 Purchasing and maintenance needs effort from the customer. Unlike in SaaS, on-premises applications require more complex purchasing, installation and updating, which takes valuable time from the customer.

CC10 Vendor lock-in is stronger than in SaaS. It is not as easy to stop using the unwanted product as it is in SaaS.

CC11 On-premises software lacks self-service. On the other hand, self-service is not always even wanted. Some companies prefer face-to-face meetings over faceless purchasing.

CC12 Data sharing and collaboration is more difficult than in SaaS. This is the result of CC2.

4.3.2 On-Premises Software – Provider's Point of View

4.3.2.1 Pros

PP1 Cash flow; unlike in SaaS, with on-premises software, delayed cash flow is not an issue. On-premises software usually do not offer as flexible pricing methods as SaaS. In addition, CAC relative to purchase price is not as high as in SaaS.

PP2 Competition is not that global, because applications are not on the Internet. Nor can customers be reached in the web.

PP3 Lock-in is stronger. CC10 listed above is a disadvantage for the customer, but a benefit for the provider.

PP4 Customers can be less worried about security and reliability. This is the same as CP6 above. See also CT2 in Section 4.1.4.

PP5 Copying a product is more difficult. Copycats cannot access a new product that easily, because it is not available for everyone on the Internet.

PP6 As stated by Fan et al. [**48**], the developer company of shrink-wrap software does not need to invest lots of money to guarantee availability.

4.3.2.2 Cons

PC1 There are less potential customers. Costs are higher, purchasing and maintenance is more complex, and local IT is needed more than in SaaS. The provider of on-premises software does not need to operate in the web. Therefore, the number of possible customers is lower than with SaaS applications.

PC2 Version control and maintenance is more complex. The provider is required to maintain several software versions at a time. In addition to standard versions, there might be tailored versions that need to be supported. Also Fan et al. [**48**] point out some disadvantages of on-premises software. It may be that a traditional on-premises software product needs to be periodically updated. E.g., anti-virus software would require constant updating to provide up-to-date security. With shrink-wrap software, it is necessary to track the installed version. Update procedure must be performed on every workstation the shrink-wrap software has been installed.

PC3 Development process is more complex. This is the result of PC2 above. The provider needs to be prepared for incompatibilities of different versions. Delivering updates is not as easy as in SaaS.

PC4 Possibly less scalable business model; unlike in SaaS, it is not that easy to grow the business with on-premises software, because all customers cannot be reached through the Internet.

PC5 Utilizing viral marketing is more difficult. Because on-premises software are not accessible in the web and there is no self-service, utilizing viral marketing is hard. User community cannot be easily utilized and integrated into the product.

PC6 The service provider has higher risks when starting a business. Development and maintenance costs with on-premises software are higher than in SaaS. It

is more difficult to get new customers, because the price of the application is high compared to the price of a SaaS application.

PC7 Possibly slower feedback cycle; more difficult upgradability slows down getting feedback on the new features. Agile development is more difficult than in SaaS.

5 SUMMARY AND CONCLUSIONS

The offerings of this thesis are finding the pros and cons of on-premises software presented in Section 4.3, the analysis of the benefits and drawbacks of SaaS done in Sections 4.1 and 4.2, and the analysis of the SaaS qualities by examining implementation cases in Chapter 3. Also presenting the implementation cases and a transformation case in Chapter 3 can be counted as an offering. These all are the results of my work.

5.1 Summary of Analysis

The SWOT analysis by Järvi et al. is extensive and reliable. Also other sources support their analysis as mentioned in Sections 4.1.6 and 4.2.6. My own examination in Chapter 3 supports the analysis, too. In practice, there are no contradictions between the lists by Järvi et al. and my findings. The overall analysis in Chapter 4 covers the viewpoints of the customer and the provider quite well. Both technical and business factors are present. However, the transformation aspect was not that clearly brought out in Chapter 4. Thus, Section 3.4 can be regarded as a value added case. Section 4.3 that analyses on-premises software is mostly my own reasoning and includes only few references. The reasoning is based on the analysis done in Sections 4.1 and 4.2. It is somewhat straightforward to derive the pros and cons of on-premises software when I first have found out the pros and cons of SaaS.

Based on the analysis done in Sections 4.1, 4.2, and 4.3, as well as Chapter 3, a good SaaS application from the customer's point of view can be summarized. It must be easy to purchase. Otherwise the customer will not go through the sales process and does not buy the offering. The application has to be easy to use. Ease of use is important especially in self-service SaaS where customer training is taken as online self-study. The application needs to be cheap. It has to be cheaper for the customer than a similar on-premises solution would be. Cost savings is one of the key benefits of SaaS. No one will buy an expensive SaaS, because prices are already low. And why should the customer change to SaaS from on-premises software if on-premises is more economical? Also the pricing model has to be tempting. It has to be reasonable. The customer is not willing to pay for extra. He only wants to pay for the use. The billing

can be based on the amount of usage or a period of time, like a month. Large up-front fees do not belong to SaaS and the customer knows that. He also demands to try the service for free. All decent SaaS offerings provide an evaluation period so that the customer can minimize risks. Almost everything in the SaaS solution should be automated. This includes purchase, trial, bug reporting, upgrading the application to the latest version, etc. Perhaps the customer still would appreciate a real support person in case there is a problem related to the service. The application has to provide configurability that is done by the customer. Configurability could include basic changes in access and security, user provisioning, simple workflow, and user preferences. Some layout related matters like colors and logos could be changed, possibly. The service must scale up and scale down according to use. The number of concurrent users is one example of scalability. The application infrastructure has to be able to serve all the users who want to do something with the application. In addition, the application must be secure and reliable. The customer wants to be ensured that his valuable data is safe. This means that outsiders cannot get access to it and the provider will not lose it. The SaaS offering should be somehow different than the thousand alternatives. Otherwise, the customer will never even consider purchasing the service. This surely is a challenge for the provider operating in the global market. Of course, the application shall be accessible independently of location and time. This was one of the key characteristics of SaaS. It is a notable benefit over on-premises software. The customer would want that there is no lock-in when he purchases a service. If there is no lock-in or only a weak lock-in, it is easier for the customer to change to another service. The SaaS application should also allow easy data sharing and collaboration. It is a new possibility of SaaS. Because the application is accessed typically with a web browser, it is practical and easy to share e.g. some files or discussion to your colleagues, family members, or friends.

Next, I will summarize a good SaaS application from the provider's point of view. The application must be multi-tenant, because it gives the provider many benefits. First, the provider can get cost savings in development and maintenance. Single software version is cheap to develop and there is only one version to be supported in upgrades. Reducing costs is one of the most important benefits of SaaS for the provider. The SaaS application has to be cheaper for the provider than an on-premises alternative. The

multi-tenant software model also enables scalability. The service has to be able to scale up and scale down according to the need. When the SaaS offering can serve more customers, it can make more money for the provider. Everything should be automated in the SaaS offering. Labor is expensive for the provider, but automation means efficiency and cost savings. Online trial, online purchase, feedback, upgrades, etc. shall be automated. Because the provider wants to save in costs, there is no customization in the application. However, there should be some kind of configurability so that customers are happy. The service has to create a lock-in. The provider wants to do that in order to be able to keep customers longer and get more revenue. Short feedback and upgrade cycles in the service allow agile development, which will have other benefits. The application has to utilize network. This means that user community should be somehow integrated into the application. Leveraging user community can be advantageous for the provider. A good SaaS offering has a memorable story to tell so that it will stay in people's minds and viral marketing can be exploited. The application or service should be difficult to copy, because there are competitors who will be very eager to copy every useful business idea once a new product appears in the market. The SaaS application should be marketed only online, because there are the potential customers. The service has to be developed to serve the long tail first. Then, it can possibly later serve other customer groups as well, but it does not work the other way around. The application shall be aimed for mass market, because volume is vital in SaaS business. The offering shall be profitable in the long term, of course. This must not be forgotten even though customer acquisition costs and support costs were high. A good SaaS application utilizes different billing methods creatively. Only the provider's creativity limits possibilities. A good SaaS offering has the business built into the product. This will support company's business. A good SaaS application creates direct value for every single customer.

In addition, when talking about what makes a good SaaS application, all the key characteristics of SaaS (see Section 2.3) apply if they were not mentioned in the summaries above.

5.2 Conclusions

5.2.1 Which Applications Fit the SaaS Model

On many counts, it has been stated that SaaS has to be simple. The purpose of use and features of a SaaS application need to be clearly brought out to ensure good self-service. A SaaS application must be a general-purpose application, to some extent. It will be hard to find a large market for more specific SaaS solutions, because there are not that many potential customers for complex and specific applications and, on the other hand, tailoring the application for customers' needs is against one of the SaaS principles. A provider may get a better compensation from large customers that demand something very specific, but volume is essential in the SaaS business. Therefore, I suggest that only simple, general-purpose applications will fit the SaaS model.

An application that allows building the business into the product fits the SaaS model, because that supports the company's business. For instance, CRM software and enterprise resource planning (ERP) software are suitable applications for SaaS. A SaaS application must create direct value for a customer. It has to produce clear value for the provider, too. Cheap applications fit SaaS, because an application has to be cheap in order to succeed in mass market. Also, SaaS customers have used to cheap service prices.

An application that requires frequent updating, such as an anti-virus or Internet security software, fits SaaS very well. Constant upgrades are vital for security software and it is natural and easy to deliver them when the application utilizes the SaaS model.

Then there is scalability. When an application needs huge scalability, it is worth considering SaaS. The architecture of the application must be designed for the cloud infrastructure. Scalability is required e.g. when there is a need to serve a vast amount of concurrent users. To give an example, ticket sales for a big rock concert would be one case. Another case could be e.g. performing very heavy computation with a mathematics software.

When there is a need for data sharing and collaboration, the SaaS model may be a good choice for the application. Office suites in the cloud, like Google Apps, are an example of this type of SaaS. Google Apps enables sharing of office documents and calendar with teams inside an organization.

Are there some kind of applications that do not fit SaaS? As mentioned above, very specific applications will not fit the SaaS business model. Nor will fit expensive applications. Also, applications that need to be used offline will surely not fit SaaS, because being online is fundamental to SaaS. Of course, it is possible to build applications that are meant to be used online, but also work when being offline. In this case, many of the benefits of SaaS are not exploitable, however.

5.2.2 When to Utilize SaaS

In which cases it is sensible to utilize SaaS? Small companies, especially start-ups, are a potential group, because they do not have a large IT department hosting on-premises software. Small and also medium-sized companies may not want to build their own IT infrastructure. It is expensive and needs continuous maintenance. It also needs facilities for servers. Servers require a space large enough, power, and cooling. On the one hand, you will have more control over everything, but on the other hand, IT infrastructure management and optimization can be cumbersome. If IT systems are not your company's core competence, I suggest that you should consider SaaS. Companies that buy their important software as SaaS offerings will experience ease of use what comes to purchase, installation, and maintenance. They also have a possibility to end the service with flexibility, which means that they do not have to pay for the service after they have unsubscribed. Naturally, this depends on the contract and contract term.

Integrating a SaaS application into on-premises systems, and particularly legacy systems, is challenging. This is the reason why SaaS providers do not usually offer integration projects. It may not be that difficult to integrate different SaaS applications together, because the applications are fresh and probably utilize some standards-based APIs. The situation is totally different with legacy systems that can be 20 years old, or even older. At that time, when these systems were built, software development was quite different than it is today. Nowadays, programmers who possess special skills

relevant to legacy systems are in demand. If organizations are already in difficulties in getting their legacy systems work on newer computers or platforms, how could it be possible to efficiently use SaaS in that kind of systems? In general, integration means tailoring, and tailoring is not usually included in SaaS.

Some industries still prefer on-premises software. Think about banking, finance, defense, and health care industries, for example. It is crucial to supervise and secure data on the premises in these industries. A bank cannot afford losing transactions from one day due to a cloud crash. As I pointed out in Section 4.1.4, cloud services are not "bulletproof". Service crashes as well as interruptions sometimes occur. Amazon's cloud crash was a disaster, but what would a similar case be for a bank? I cannot imagine that any bank would trust its business critical systems in SaaS. It will not happen anytime soon, at least. Amazon is regarded as one of the bests in its field and still they failed in preserving customers' data.

SaaS is an attractive model when there is a need to store large amounts of data. In the field of health care, medical imaging and 3D imaging in particular generates loads of image data. This data needs to be stored somewhere and the cloud would be one possible option. However, the risks seem to be too high, still. If there is a serious crash or some other accident and patient images are lost, it can and will result in unnecessary radiation for patients.

Trust is a key feature for a SaaS provider. Without trust there are no customers. The image and brand of a SaaS vendor has a great importance. This might be the reason why F-Secure has started to offer its online backup service (see Section 3.4.3). It is essential to trust that the data will be secure when a customer is thinking his backup options. F-Secure has a good reputation in the anti-virus and Internet security software industry, which has probably helped the company to expand its business.

Home users, consumers, are in a better position than enterprise customers, if you think the risks. Consumers do not have their business to lose when they utilize SaaS. A consumer might lose his personal photos, music, or some other personal files in the case of SaaS disaster. That would be very unfortunate, but the customer would survive

(although some photos may be priceless). Also importantly, a consumer can get access to state of the art technology thanks to SaaS and cloud computing.

What is the future of SaaS and software models in general? The tendency seems to be that SaaS is growing faster than on-premises software. Especially when completely new software is designed, SaaS is a strong option. Microsoft suggests that the future is a combination of on-premises software and SaaS, i.e. Software + Services (S+S) [**49**].

Microsoft is not the only company that has noticed the possibility to take the best of local software and Internet services and use this combination. In fact, F-Secure has already moved a part of the computation that is done in its security products to the cloud. There still is a locally installed client application that provides a GUI, but reputation of files, sites, and URLs are provided to F-Secure's applications by the real-time protection network that is implemented in the cloud. This moves the processing and memory intensive functions to the cloud and it has a positive effect on the performance of the client software. In addition, when the collective intelligence of client systems is combined, the real-time protection network is able to react to new threats a lot faster than before.

Another example of combining local software and cloud computing is mathematics software packages called MATLAB and Mathematica. Both software are capable of utilizing cloud computing to perform heavy computing. High performance computing is done in Amazon EC2. SaaS and cloud computing offer a great opportunity for parallel batch processing. AzureBlast (see Section 3.2) is a good example of this kind of parallel computing. When the amount of data to analyze is huge, it is reasonable to use the high-end capacity of cloud services to get the task done faster.

When to utilize software as a service? To answer the research question of this thesis accurately would require knowledge of the business case in question. It would require an understanding of how SaaS brings more value for the company when comparing to on-premises software. In the end, it is important that a SaaS customer and a SaaS provider can get economic value from SaaS.

REFERENCES

[1] Turner, M., Budgen, D., and Brereton, P., "Turning software into a service" *Computer*, vol. 36, p. 38–44, Oct. 2003.

[2] Chong, F. and Carraro, G. (2006, Apr.) Architecture Strategies for Catching the Long Tail. [Online]. http://msdn.microsoft.com/en-us/library/aa479069.aspx. Accessed 9.2.2010.

[3] Salesforce.com, inc. (2010) What is Software as a Service (SaaS) - salesforce.com. [Online]. http://www.salesforce.com/saas/. Accessed 28.2.2010.

[4] Mietzner, R., Leymann, F., and Papazoglou, P. M., "Defining Composite Configurable SaaS Application Packages Using SCA, Variability Descriptors and Multi-tenancy Patterns" in *The Third International Conference on Internet and Web Applications and Services*, 2008, p. 156–161.

[5] Mell, P. and Grance, T., "The NIST Definition of Cloud Computing Version 15," National Institute of Standards and Technology, Information Technology Laboratory, Technical Report, 2009.

[6] Armbrust, M., et al., "A View of Cloud Computing" *Communications of the ACM*, vol. 53, no. 4, p. 50–58, Apr. 2010.

[7] Mell, P. and Grance, T. (2009, Oct.) Presentation on Effectively and Securely Using the Cloud Computing Paradigm v26. [Online]. http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-computing-v26.ppt. Accessed 17.5.2010.

[8] Gillan, C. and McCarty, M., "ASPs Are for Real... But What's Right for You?" International Data Corporation White Paper, 1999.

[9] Software & Information Industry Association (SIIA), "Software as a Service: Strategic Backgrounder," Technical Report, 2001.

[10] Cheun, W. D., Lee, Y. J., Lee, W. J., and Kim, D. S., "A Quality Model for Evaluating Software-as-a-Service in Cloud Computing" in *The 7th ACIS International Conference on Software Engineering Research, Management and Applications*, 2009, p. 261–266.

[11] Cusumano, M., "The Changing Software Business: Moving from Products to Services" *Computer*, vol. 41, p. 20–27, Jan. 2008.

[12] CNET. (2010, Mar.) Live blog: Ballmer on the cloud, Beyond Binary - CNET News. [Online]. http://news.cnet.com/8301-13860_3-10463930-56.html. Accessed 14.8.2010.

[13] Google. (2011) Google App Engine - Google Code. [Online]. http://code.google.com/appengine/. Accessed 3.2.2011. Checked 6.2.2011.

[14] Microsoft. (2010) Windows Azure Platform. [Online]. http://www.microsoft.com/windowsazure/. Accessed 6.11.2010. Checked 3.4.2011.

[15] Chappell, D., "Introducing Windows Azure" Whitepaper, 2010.

[16] Chappell, D., "Introducing the Windows Azure Platform" Whitepaper, 2010.

[17] Amazon Web Services, "Overview of Amazon Web Services" Whitepaper, 2010.

[18] Amazon Web Services. (2010) Amazon Web Services. [Online]. http://aws.amazon.com/. Accessed 7.11.2010. Checked 28.12.2010.

[19] Amazon Web Services. (2010, Dec.) AWS Documentation library. [Online]. http://aws.amazon.com/documentation/. Accessed 29.12.2010.

[20] Google. (2010) Google Apps for Business | Official Website. [Online]. http://www.google.com/apps/intl/en/business/index.html. Accessed 10.10.2010. Checked 17.4.2011.

[21] Google. (2011) Software-as-a-service has built-in security advantages – Google Apps. [Online]. http://www.google.com/apps/intl/en/business/infrastructure_security.html. Accessed 23.4.2011.

[22] Google. (2010, Mar.) Official Google Enterprise Blog: Disaster Recovery by Google. [Online]. http://googleenterprise.blogspot.com/2010/03/disaster-recovery-by-google.html. Accessed 23.4.2011.

[23] BBC. (2010, May) BBC News - Google admits wi-fi data collection blunder. [Online]. http://news.bbc.co.uk/2/hi/technology/8684110.stm. Accessed 24.4.2011.

[24] Google. (2008, Oct.) Official Google Blog: What we learned from 1 million businesses in the cloud. [Online]. http://googleblog.blogspot.com/2008/10/what-we-learned-from-1-million.html. Accessed 24.4.2011.

[25] Lu, W., Jackson, J., and Barga, R., "AzureBlast: A Case Study of Developing Science Applications on the Cloud" in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, Chicago, 2010, p. 413–420.

[26] NCBI, U.S. National Library of Medicine. (2010, Jun.) Standalone BLAST Setup for Windows PC - BLAST Help - NCBI Bookshelf. [Online]. http://www.ncbi.nlm.nih.gov/books/NBK52637/. Accessed 26.4.2011.

[27] Maluf, A. D. and Okimura, T., "NASA Web-Accessible Open Software as a Service Framework" in *Aerospace conference, 2009 IEEE*, Big Sky, MT, 2009, p. 1–8.

[28] Komssi, M., Kauppinen, M., Heiskari, J., and Ropponen, M., "Transforming a Software Product Company into a Service Business: Case Study at F-Secure" in *33rd Annual IEEE International Computer Software and Applications Conference*, 2009, p. 61–66.

[29] F-Secure Corporation. (2008, Oct.) Internet Service Providers offer new unlimited F-Secure Online Backup. [Online]. http://www.f-secure.com/en_EMEA-Corp/pressroom/news/2008/fs_news_20081008_01_eng.html. Accessed 13.5.2011.

[30] F-Secure. (2010) F-Secure - Storage services - Online Backup - Overview. [Online]. http://www.f-secure.com/en/web/operators_global/storage-services/online-backup. Accessed 9.5.2011.

[31] F-Secure. (2011) F-Secure - Backup - Online Backup - Overview. [Online]. http://www.f-secure.com/en/web/home_global/backup/online-backup. Accessed 9.5.2011.

[32] F-Secure Corporation, "F-Secure Content Enabler, Version 4" White Paper, 2010.

[33] Microsoft. (2000, Jun.) Steve Ballmer Speech Transcript - Forum 2000. [Online]. http://www.microsoft.com/presspass/exec/steve/06-22f2k.mspx. Accessed 2.4.2011.

[34] Microsoft. (2000, Jun.) Microsoft Unveils Vision for Next Generation Internet: Company Introduces .NET Generation of Software. [Online].

http://www.microsoft.com/Presspass/press/2000/jun00/forumumbrellapr.mspx. Accessed 2.4.2011.

[35] Microsoft. (2002, Feb.) Microsoft Launches XML Web Services Revolution With Visual Studio .NET and .NET Framework. [Online]. http://www.microsoft.com/presspass/press/2002/feb02/02-13revolutionpr.mspx. Accessed 2.4.2011.

[36] Microsoft. (2010, Apr.) Microsoft Visual Studio 2010 and Microsoft .NET Framework 4 Available. [Online]. http://www.microsoft.com/Presspass/press/2010/apr10/04-11VS10PR.mspx. Accessed 2.4.2011.

[37] Microsoft. (2011) .NET Framework Conceptual Overview. [Online]. http://msdn.microsoft.com/en-us/library/zw4w595w.aspx. Accessed 3.4.2011.

[38] Markoff, J. (2001, Apr.) An Internet Critic Who Is Not Shy About Ruffling the Big Names in High Technology - New York Times. [Online]. http://www.nytimes.com/2001/04/09/business/internet-critic-who-not-shy-about-ruffling-big-names-high-technology.html. Accessed 3.4.2011.

[39] Cusumano, M., "Cloud Computing and SaaS as New Computing Platforms" *Communications of the ACM*, vol. 53, no. 4, p. 27–29, Apr. 2010.

[40] Järvi, A., Karttunen, J., Mäkilä, T., and Ipatti, J., *SaaS-käsikirja*. Turku, Finland, 2011.

[41] Sun, W., Zhang, K., Chen, S.-K., Zhang, X., and Liang, H., "Software as a Service: An Integration Perspective" *Lecture Notes in Computer Science*, vol. 4749, p. 558–569, 2007.

[42] Business Insider. (2011, Apr.) Amazon's Cloud Crash Disaster Permanently Destroyed Many Customers' Data. [Online]. http://www.businessinsider.com/amazon-lost-data-2011-4. Accessed 29.5.2011.

[43] Satake, K., "Fujitsu's Approach to SaaS in Japan—Fujitsu SaaS Platform—" *Fujitsu Scientific and Technical Journal*, vol. 45, no. 3, p. 265–274, Jul. 2009.

[44] Armbrust, M., et al., "Above the Clouds: A Berkeley View of Cloud Computing," University of California at Berkeley, Technical Report, 2009.

[45] Sääksjärvi, M., Lassila, A., and Nordström, H., "Evaluating the Software as a Service Business Model: From CPU Time-Sharing to Online Innovation Sharing" in *IADIS International Conference e-Society*, 2005, p. 177–182.

[46] F-Secure Corporation, "Annual Report 2010" Financial Publication, Mar. 2011.

[47] York, J., "Software-as-a-Service Success – The Top Ten Dos and Don'ts of SaaS" Chaotic Flow Whitepaper, 2008.

[48] Fan, M., Kumar, S., and Whinston, B. A., "Short-term and long-term competition between providers of shrink-wrap software and software as a service" *European Journal of Operational Research*, vol. 196, no. 2, p. 661–671, 2009.

[49] Microsoft. (2011) Software + Services (S+S). [Online]. http://msdn.microsoft.com/en-us/architecture/aa699384.aspx. Accessed 7.6.2011.