



University of Applied Sciences Darmstadt
University of Wisconsin - Platteville

The efficacy of Cloud Computing in Vertical Industries as measured by Profitability

In Partial Fulfillment of the Requirement for the Degree
Master of Science

By
Stefan Tobias Ruehl

Supervisors: Prof. Dr. Udo Bleimann (h_da)
Prof. Mike Rowe, Ph.D. (UWP)
Co-Supervisor: Prof. Dr. Christoph Wentzel (h_da)

Start Date: March 15, 2010
End Date: September 15, 2010



University of Applied Sciences Darmstadt

University of Wisconsin - Platteville

The efficacy of Cloud Computing in Vertical Industries as measured by Profitability

In Partial Fulfillment of the Requirement for the Degree
Master of Science

By
Stefan Tobias Ruehl

Supervisors: Prof. Dr. Udo Bleimann (h_da)
Prof. Mike Rowe, Ph.D. (UWP)
Marcus Zinn, M.Sc. (h_da)
Co-Supervisor: Prof. Dr. Christoph Wentzel (h_da)

in cooperation with

T · · Systems · · ·

Statement of Authorship

I hereby declare that this thesis is written on my own solely by using the sources quoted in the thesis body. All contents (including text, figures, and tables) which are taken from sources, both verbally and in terms of meanings are clearly marked and referenced.

The thesis has not been previously submitted to any other examination board, neither in this nor in a similar form.

Stefan T. Ruehl

Acknowledgment

It is a pleasure to thank those who made this thesis possible.

First of all I want to thank my advisor Marcus Zinn whose encouragement, guidance, support, and criticism from start to the very end allowed me to understand the objectives and challenges of a master's thesis.

Furthermore, I want to thank the folks from T-Systems who provided an excellent, supporting, innovative, and inspiring environment in which it was a pleasure to create this thesis. Thus, I want to give my special thanks to Stephan Verclas and Ralf Konrad but also to all my colleagues of the Innovation Center. I look forward to continue working with you guys.

Last but not least I want to thank Prof. Bleimann, Prof. Rowe, and Prof. Wentzel for supervising my thesis.

Cloud Computing is one of the most significant topics in IT-Management at the moment. The delivery of computing resources such as applications on demand represents a shift of paradigm. So far, the growing Cloud Computing application (Software-as-a-Service) market is still mostly limited to horizontal applications. This means they usually cover processes that are common to most companies regardless of the specific industry. If an operator gains the ability to provide industry specific SaaS-Applications, they can reach new markets with this new kind of applications. Thus, the goal of this thesis is to analyze the efficacy of such applications in vertical industries measured by profitability. Since vertical markets are naturally smaller, it is even more important that investments are as low as possible. Furthermore, it is necessary that a given application is flexible and adaptable to customers' requirements. This way it is possible to reach a higher number of customers. Thus, revenue can be increased.

The first part of this thesis proposes a platform, which purpose is to promote the aspects of flexibility and lower investments. This is achieved by utilizing an approach that combines the paradigm of Cloud Computing with the paradigm of Service-oriented Computing.

Starting from this platform the profitability of industry-specific SaaS-Applications is analyzed. In order to do that a mathematical model is created which captures the necessary inputs to calculate the profitability of such applications. Based on this mathematical model it is possible to make a general statement about the profitability. Furthermore, the proposed model can be used as a tool to analyze the profitability of given scenarios.

The thesis concludes by analyzing a specific example of the area of eHealth.

Cloud Computing ist im Moment eins der bedeutungsvollsten Themen im Bereich des IT-Managements. Die Bereitstellung von IT Ressourcen wie zum Beispiel bedarfsabhängige Applikationen stellt einen Paradigmenwechsel dar. Bis heute ist der wachsende Markt der Cloud Computing Applikationen (Software-as-a-Service) größtenteils dominiert von horizontalen Applikationen. Dies bedeutet, dass sie normalerweise industrieübergreifende Prozesse abbilden und deshalb von Unternehmen aus verschiedensten Industrien verwendet werden können. Sollte es einem Anbieter solcher Software gelingen, industriespezifische SaaS-Applikationen anzubieten, kann er mit dieser neuen Art von Applikationen in neue Märkte vordringen. Deshalb ist es das Ziel dieser Masterarbeit, die Wirksamkeit dieser Applikationen in vertikalen Industrien auf Basis der Profitabilität zu untersuchen. Da vertikale Märkte naturgemäß kleiner sind als horizontale ist es wichtig, dass die Investitionen so klein wie möglich gehalten werden. Außerdem ist es nötig, dass Applikationen über eine Flexibilität verfügen, die es erlaubt, sie an die Anforderungen verschiedenster Kunden anzupassen. Wenn dieses Ziel erreicht werden kann, ist es auch möglich, eine signifikante Anzahl an Kunden ansprechen zu können. Dies wiederum hat einem Umsatzanstieg zur Folge.

Der erste Teil der Masterarbeit beschäftigt sich mit der Konzeptionierung einer Plattform, die sowohl Flexibilität als auch niedrige Investitionskosten ermöglichen soll. Diese Ziele werden erreicht, indem das Paradigma Cloud Computing mit dem Paradigma Service-oriented Computing kombiniert wird.

Basierend auf dieser Plattform wird die Profitabilität von industriespezifischen SaaS-Applikationen analysiert. Konkret wird dafür ein mathematisches Modell entwickelt, das auf Basis von nötigen Eingabegrößen die Profitabilität eines Projektes bestimmt. Dieses mathematische Modell erlaubt es auch, eine generelle Aussage über die Profitabilität von industriespezifischen SaaS-Applikationen zu machen. Außerdem ist es möglich, das Modell anzuwenden, um Applikationsszenarien zu evaluieren und zu analysieren.

Um die Masterarbeit abzuschließen, wird dies anhand eines Beispiels aus dem eHealth Umfeld demonstriert.

Abstract	iii
Abstrakt (German Version)	iv
1 Introduction	1
1.1 Goals of this Thesis	2
1.2 Document Structure	3
2 Fundamentals	4
2.1 Cloud Computing	4
2.1.1 Vision of Computing as the 5th Utility	5
2.1.2 Characteristics	6
2.1.3 Service Models	7
2.1.4 Deployment Models	8
2.1.5 Economic Aspects	10
2.2 Service-Oriented Computing	11
2.2.1 Introduction	11
3 The Delivery Platform	13
3.1 Vision	13
3.2 Approach	16
3.3 Integration Platform	17
3.3.1 Infrastructure Components	17
3.3.2 Cloud Computing Foundation	20
3.3.3 Multi-tenancy	21
3.4 Reference Architecture of Applications	23
3.4.1 Reference Architecture Method	23
3.4.2 Reusability	30

3.5	Application Building	30
3.5.1	Flexibility Strategies	30
3.5.2	Flexibility	31
3.5.3	Customization of Applications	32
3.6	Deployment model	33
4	Economic Evaluation	35
4.1	Analysis of SaaS	35
4.1.1	SaaS versus Traditional License	36
4.1.2	Customers' View on SaaS	37
4.1.3	Operators' View	38
4.2	Abstract Target Market	39
4.2.1	Possible Applications	39
4.2.2	Target Customer Profile	39
4.3	Pricing Strategy	41
4.3.1	Pricing SaaS	42
4.3.2	Subscription Pricing based on Perpetual License	42
4.3.3	Other Factors for Subscription Pricing	44
4.4	Mathematical Model	45
4.4.1	Measure for Profitability	45
4.4.2	Net Present Value	46
4.4.3	Weighted Average Cost of Capital	47
4.4.4	Net Present Value - Example	48
4.4.5	Free Cash Flow - Analysis	49
4.4.6	Revenue	51
4.4.7	Expenses	53
4.4.8	Initial Investment	55
4.5	Analysis of Mathematical Model	55
5	Discussion of Results	57
5.1	Example: Patient Transition	57
5.1.1	Introduction of the Example	58
5.1.2	Demographic Predictions	59
5.1.3	Motivation	59
5.1.4	Creation of the Process Model	60
5.1.5	Flexibility	62
5.1.6	Determination of Initial Investment	63
5.1.7	Determination of other Inputs	65
5.1.8	Determination of Profitability	67
5.1.9	Sensitivity analysis	68
5.1.10	Shortcomings of the Example	70

5.2	Analysis of Approach	71
5.2.1	Chapter 3: The Delivery Platform	71
5.2.2	Chapter 4: Economic Evaluation	72
6	Summary and Conclusion	74
6.1	Summary	74
6.2	Conclusion	75
6.3	Future Research	76
6.3.1	Composite SaaS-Applications	76
6.3.2	Profitability Analysis	77
	Bibliography	78
A	Definitions of Cloud Computing	83
B	QUASOSS-Paper: “A Method for Engineering an Architecture for Software-as-a-Service Applications”	85
C	Feedback QUASOSS-Conference	92
	List of Figures	95
	List of Tables	96
	List of Abbreviations	97

CHAPTER 1

Introduction

Cloud Computing (CC) is probably the most discussed topic in IT-Management at the moment. In fact, Gartner has just recently published that they see CC as one of the most significant strategic trends in IT industry for the year 2010 and beyond [KP10]. According to another recent study by Ludwig Maximilian University, Munich more than half of the surveyed executives expect CC to become the dominant method of sourcing ICT services in the future [KBK10].

CC is the latest approach in utility computing, its goal is to evolve computing in a way that it will become the 5th utility after water, electricity, gas, and telephone [BYV⁺09].

It is commonly recognized that CC can be split into three different Service Models: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS). The first two deal with the offering of basic infrastructure like computing power, storage, and database management systems. SaaS on the other hand adds more value and allows offering entire applications in the cloud. Most of the currently promoted applications cover horizontal processes of a company such as ERP and CRP systems [Wit08]. This is due to the fact that horizontal processes are very similar or even the same in different companies from different industries. This results in big markets for this kind of applications.

However, this thesis will analyze the efficacy of these applications in vertical markets or industries. This means that the applications dealt with in this thesis cover processes that are specific to a particular industry and thus addresses a much smaller number of customers [Szy02].

The biggest challenge for this kind of applications is that the size of the market they aim to penetrate is smaller. In order to be profitable and exploit economies of scale applications need to be usable and attractive to a number of customers to provide sufficient revenue to amortize the investment and thus earn a profit. If an application offers the ability to adapt to different process variations and the different requirements of various customers, the number of customers can be increased. Due to the initial smaller market sizes in vertical applications this is especially important.

1.1 Goals of this Thesis

The goal of this thesis is to analyze the efficacy of Cloud Computing applications in vertical industries. In other words, the goal is to analyze under which circumstances industry-specific SaaS-Applications can be profitable.

In order to do that the first part of this thesis provides the conceptual design of an application delivery platform that allows for the develop, deploy, and deliver of applications from the cloud in a financially beneficial way. This is of high significance for the thesis since vertical markets are smaller than horizontal. The approach that is used to develop the platform starts by stating an vision of such a platform. Based on this vision it is possible to determine some abstract requirements that describe the platform. In order to fulfill these requirements technology paradigms are matched.

In order to gain a significant market share of a small market it is necessary that a given application can be adapted according to the demand of different customers. To achieve this goal the paradigm of Service-oriented Computing (SOC) is utilized. In order to reach a higher level of flexibility a SOC is employed as a well-suited architectural style for realizing SaaS-Applications [MMLP09, CC06b, GSH⁺07]. This is due to the fact that this architecture offers a flexible way for building composite applications out of existing building blocks. Furthermore, this approach allows reuse of individual sub-systems or components on an inter-applicative level. This ability contributes to the goal of minimizing development cost and decreasing time to market. However, the highlight of this part is the presentation of an architecture model that allows one to reach these goals.

Based on this conceptual design the second part of this thesis deals with an business evaluation of SaaS-Applications for vertical industries. In order to do this abstract target markets are analyzed. The point here is that it is not possible to create an application that is suitable for all customers of a given market with a limited budget. It is important, however, to create an application that can be suitable for a significant number of customers in order to amortize the investment. Furthermore, a possible pricing strategy will be presented. This strategy is a benchmarking technique that uses the prices of common perpetual licenses of competitors to determine the subscription price for a SaaS-Application. Based on the conceptual design and the business evaluations that were just mentioned a mathematical model is created which captures all necessary inputs to estimate the profitability of a given application. The approach that is used to create this model starts by discussing different techniques that can be used for profitability analysis. Based on this discussion a suitable technique is picked and a numerical measure for profitability is determined. Based on the results of the previous parts the inputs are chosen to create a equation that allows to calculate this measure for profitability.

To conclude, this part the model is analyzed, in order to gain a conclusion about the efficacy of industry-specific SaaS-Applications.

1.2 Document Structure

This document is divided into six chapters. Figure 1.1 gives an overview of these chapters and illustrates the flow of the thesis.

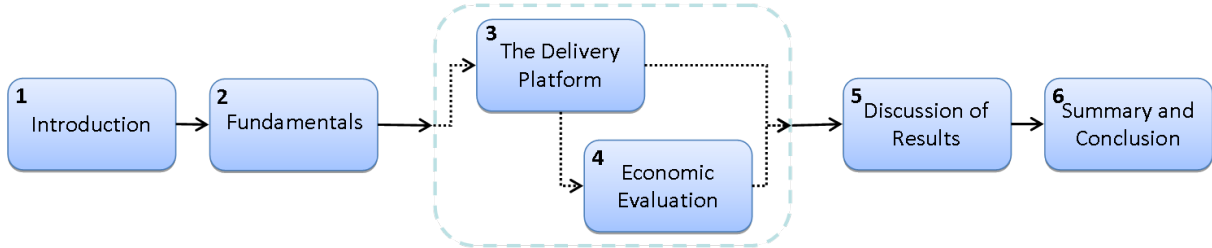


Figure 1.1: Structure of this Thesis.

Chapter 1: *Introduction*

This chapter introduces the reader to the topic and gives a brief overview of the entire thesis.

Chapter 2: *Fundamentals*

This chapter introduces the reader to the topics that are relevant for this thesis.

Chapter 3: *The Delivery Platform*

In this chapter a platform is designed whose goal is to enable the financially beneficial development, deployment, and delivery of industry specific applications. In order to do this, the chapter starts by stating a vision of such a platform and defines fundamental requirements that need to be fulfilled. Based on these requirements the platform is designed and an architecture for Software-as-a-Service-Applications is proposed.

Chapter 4: *Economic Evaluation*

Based on the platform design of chapter 3 this chapter deals with the economic evaluation of applications provided from the platform (this structure is illustrated by figure 1.1). To perform the economic evaluation the Software-as-a-Service business model is analyzed in detail before target markets and pricing is discussed. Based on these discussions a mathematical model is created which can be used to analyze the efficacy of SaaS-Applications for vertical industries.

Chapter 5: *Discussion of Results*

This chapter's purpose is to present the results of the thesis. In order to do this the results of chapters 3 and 4 are applied to a specific example (structure is illustrated by figure 1.1). This chapter also contains a summary and discussion of the limitations and shortcomings of this approach.

Chapter 6: *Summary and Conclusion*

This chapter concludes the thesis by summarizing the results and giving an outlook.

This chapter discusses the state-of-the-art concepts that are relevant for this thesis. In particular, it introduces the reader to the new area of Cloud Computing which deals with the continuation of the vision of computing resources as a utility. This chapter also discusses the different flavors of Cloud Computing as well as the economic reasons why Cloud Computing can be beneficial to both the operators offering it and the customers using it. For this thesis and the context of Cloud Computing operator refers to a company that provides Cloud Computing services. The introduction to Cloud Computing is especially important for this thesis since there is still confusion about the different terms in this area. Therefore, a common vocabulary for this thesis needs to be defined.

The second part of this chapter gives an introduction and overview of service-oriented computing as to assemble application components into a network of services that can be loosely coupled in order to create flexible applications.

Both topics are relevant for the following chapters.

2.1 Cloud Computing

Cloud Computing (CC) has recently become a buzzword that is used in academia as well as in industry. Fig. 2.1 shows, how often the term Cloud Computing was entered in Google search relative to the total search-volume across various regions of the world, and in various languages. Even though the term is already widely used, the definition of what CC really is differs a lot. Marketing departments especially have done their part in softening the definition by trying to put the cloud label on their already established products. Forrester Research Inc., for example, calls this process “*Cloud Washing*” [RML10].

Due to this ongoing confusion, it is necessary for this work to introduce a common definition that can be used for the entire thesis. This is done in the following section.

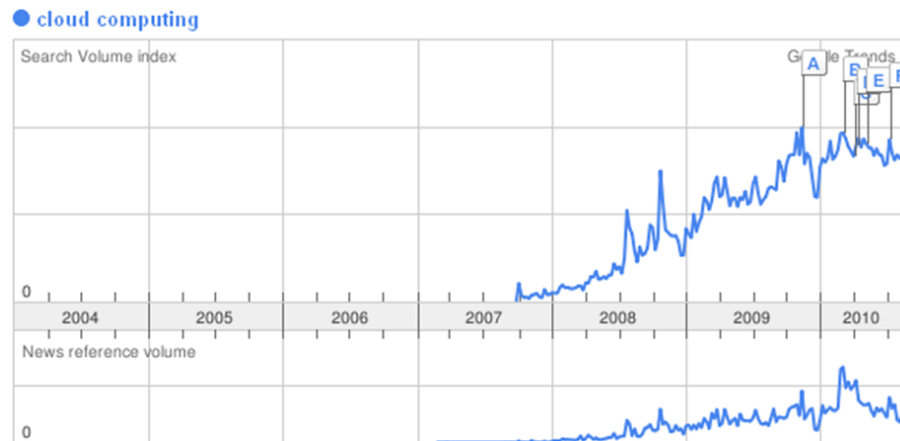


Figure 2.1: Cloud Computing on Google Trends [Goo10c].

2.1.1 Vision of Computing as the 5th Utility

Already in 1969, Leonard Kleinrock one of the chief scientists of the original Advanced Research Agency Network (ARPANET) who created the first packet switched network, was quoted to have said the following:

“As of now, computer networks are still in their infancy, but as they grow up and become sophisticated, we will probably see the spread of ‘computer utilities’, which, like present electric and telephone utilities, will service individual homes and offices across the country.” [Kle69]

Since the 1960s vision of computing resources offered as the 5th utility has been discussed continually. Similar to the traditional utilities such as water, electricity, gas, and telephony, this computing utility will provide a level of computing service that is necessary to satisfy the needs of a general community [BYV⁺09]. This 21st century’s model of computing anticipates a significant transformation of the computing industry [AFG⁺09, BYV⁺09]. The model promises to require service users to no longer have to invest in new infrastructure. As a side-effect it eliminates the risk for users running into problems when operating their existing already very complex infrastructure or adding new components to it. Instead, users pay operators only for the IT resources they actually consume based on their usage. This scenario poses many challenges for providers of IT resources and the technology that enables this vision.

Since Leonard Kleinrock’s visionary expression in the late 1960’s, IT has experienced a tremendous evolution. New technologies and paradigms have evolved that each contributes to the vision of computing power as a utility and its newest adaption CC. Probably the most important evolution undertook the Internet. It has evolved to an inexpensive platform that spans the world and enables devices to communicate without concern for where they are located. Having such a broadband infrastructure is a most basic requirement for utility computing.

Another paradigm that has emerged in the last couple of years is service computing. Its focus is on connecting business processes tightly with the IT services that are required to support them. Thus its goal is to optimize the way in which IT systems support business processes in order to increase their flexibility to dynamically adapt to a changing business environment. This change of paradigm has changed the understanding of how IT is used in companies.

The last significant evolution that contributes to the vision of utility computing is virtualization. Even though the term virtualization was first coined in the 1960s but by the 1990s had disappeared from most popular hardware platforms. However, it had a comeback in the beginning of the 2000s [Ros04]. The basic idea of it is the separation of applications from physical infrastructure. An Application in this context refers to packaging the application into a virtual machine (VM) that contains everything that is necessary to run the application including Operating System, Middlewares, Databases etc. This VM can be deployed and executed by any virtualized physical infrastructure. Thus, the same infrastructure can runs several VMs. This optimizes the usage of physical resources.

Based on these evolutions, CC is by far the most promising approach when dealing with the provisioning computing resources as a utility. However, there are two related concepts, Cluster and Grid Computing, that have appeared within the last decades. Cluster and Grid computing are often confused with CC, although, they bare significant differences. Cluster computing can be defined as a *“type of parallel and distributes system, which consists of a collection of inter-connected stand-alone computers working together as a single integrated computing resource”* [BYV⁺09]. Grid computing, on the other hand, can be defined as *“a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed ‘autonomous’ resources dynamically at runtime depending on their availability, capability, performance, cost, and users’ quality-of-service requirements”* [BYV⁺09]. The reason why CC is often confused with the previous two expressions is that CC possesses characteristics of both but adds strong support for virtualization. Thus, CC has emerged as the latest paradigm to promise reliable services delivered through next-generation data centers that are built on virtualized computing and storage technologies [BYV⁺09]. Thereby, it is not limited just to infrastructure like data storage and computational power but deals with much more up to the application layer.

So far there is no commonly unitized definition¹.

2.1.2 Characteristics

Based on the aforementioned definitions it is possible to define characteristics of CC. As a matter of fact almost every institution that published a definition for CC also published characteristics. The following section explains some of the most important ones in order to give a brief overview about the most significant aspects of CC.

Service-based An important characteristic of CC is that it distinguishes between provider and customer concerns. Implementation details of a service are hidden from the customer by a service interface that is well-defined. The service can be considered *ready-to-use* or *off-the-shelf* because it is designed to serve the needs of a set of customers [PF09]. The customer uses the service without the knowledge of how it actually works. The description of the service and how to use the service is defined in the service levels. These are defined in a non-technical manner

¹Referee to the appendix A (Pg. 83) for an overview of some of the most quoted definitions.

and describe IT outcomes rather than the technology itself and its capabilities.

Shared Pool of Resources The provider's resources are pooled in order to be usable for multiple services. This is done by assigning and reassigning physical and virtual resources dynamically to or from specific services or based on a multi-tenant model to a specific customer [MG09]. Thus, resources are shared by multiple customers in order to use maximized physical resource efficiency. One of the fundamental technologies that make this kind of pooling of resources possible is virtualization as mentioned in the previous section.

Since services share a pool of resources, operators can benefit from economies of scale [PF09].

Scalable and Elastic This characteristic is probably the most innovative one. It deals with the provisioning of capacities according to customer demand. In some cases this is even done automatically. Thus, CC generally has the ability to quickly scale up capacity in case of a high customer demand or scale down in case of a low customer demand. The ability to perform this kind of scaling is located in the underlying infrastructure and software platform [PF09] which can allocate or deallocate resources for the particular service from a resource pool.

To the customer the availability of resources often appears to be unlimited and can be purchased in a specific quantity at any time [MG09]. This of course has an impact on economic models.

Metered by Use Cloud systems automatically monitor the usage of the services they provide. Thus it is possible to control and optimize resource usage [MG09]. Tracking of service usage is also important for business models of cloud services since it is the foundation of providing pay-as-you-go plans, subscriptions or fixed plans. These plans are based on usage by a customer, which might be in terms of hours, data transfers and/or other use-based attributes [PF09].

2.1.3 Service Models

Over the past couple of months common agreement has formed that CC can be split into three service models. A service model is a scenario in which a Cloud Provider offers a certain amount of capability to service users. Each model contains a certain amount of service value [VRCL09]. Cloud providers typically focus on provisioning just one service model. However, there is no restriction in offering offer multiple types at the same time [SJN10, Pg. 13].

The following describes three commonly used models.

Infrastructure as a Service (IaaS) The infrastructure cloud provider manages a large pool of computing resources, such as storing and processing capacity. Customers use parts of this pool and are charged based on their usage. Usually IaaS refers especially to two different resources that are provisioned as a service [VRCL09, SK10, MG09]. This service is based on timeshare systems started in the 1960s.

Data Storage The provider manages a large set of storage devices and offers the service of storing customer data on it. As a result customers can reduce their expenditure for storage

infrastructure that customers can utilize for other purposes. In addition, customers do not need to be concerned with storage aspects such as data mirroring or archiving. Customers are billed based on the amount of storage they actually consume

Processing Power Providers manage a large set of servers. By using virtualization techniques they are able to split, assign and dynamically resize these resources. The cloud customers use a virtual machine which is executed in this virtualized environment. In this way the users have access to infrastructure that adjusts to their needs. This model can easily be confused for classic server hosting. However, it differs fundamentally since the customer is not limited to one physical machine but can use unlimited computing power.

An well-known example for this IaaS is Amazon's platform Amazon Elastic Compute Cloud (Amazon EC2)[Ama10].

Platform as a Service (PaaS) This service model adds more value to the capabilities of IaaS by adding one additional abstraction layer. Instead of supplying a virtualized infrastructure, PaaS provides a computing platform and/or solution stack as a service on which customers can deploy and execute their applications [VRCL09, SK10, MG09]. Thus, customers do not have to deal with the cost and complexity of buying and managing the underlying hardware and software layers. The sizing of the hardware resources of the PaaS platform is done in an elastic scalable way in order to generally provide as much computing resources as the customer demands. A well-known example is Google App Engine [Goo10b].

Software as a Service (SaaS) This is the service model with the highest capability and service value. Instead of provisioning an infrastructure, on which customers can build their applications, SaaS adds another abstraction layer and delivers entire applications to the customer [VRCL09, SK10, MG09, AFG⁺09]. These applications are commonly web-based and users access them by making use of a standard web-browser. SaaS providers usually offer these applications based on a pay-per-use fee. Customers, on the other hand, do not have to be concerned with the installation, operation, update etc. of the application or the infrastructure being used to host the application. SaaS applications profit greatly from an environment that adjusts to changing demands. According to these aspects, CC service models can be structured as displayed by figure 2.2.

One of the more famous example of an SaaS application is the Customer Relationship Management (CRM) System Salesforce [sal10a].

2.1.4 Deployment Models

Similar to the service models discussed in the previous section, there is common agreement that CC can be deployed in different ways [SK10, MG09]. The following section discusses different kinds of deployment models.

Which deployment model is used in a particular case is highly depends on the customer's needs.

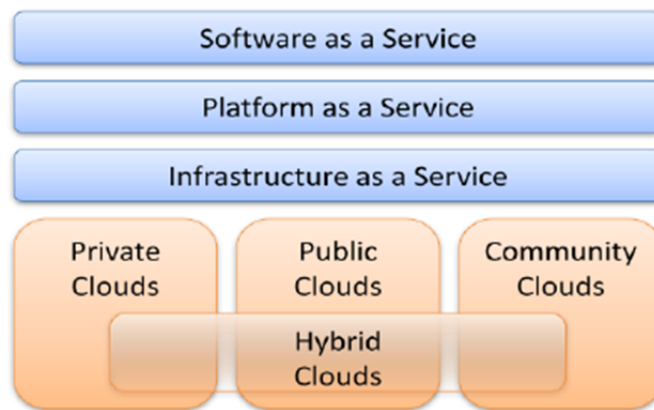


Figure 2.2: Cloud Computing Stack [SK10].

Public Cloud Using this deployment model a cloud infrastructure is made available to the general public or a large industry group and is owned by an organization selling cloud services [SK10, MG09, SJN10].

This model is suitable for customers with a need for a cheap price and a highly standardized production environment. Therefore, customers must accept low data security and data privacy levels. An example of this is TurboTax Online, which in the US can be used to prepare tax returns [Tur10].

Community Cloud This model allows sharing the infrastructure within a distinct group of organizations. This cloud is usually set up to the specific needs and requirements of this group [SK10, MG09]. This might include collaboration among the different stakeholders of the cloud [RML10]. It could be managed by the organizations themselves or by a third party.

This model is suitable for communities of organizations which wish to share a cloud and maybe have special requirements or needs.

Private Cloud This deployment model provides an exclusive cloud for every single customer. The cloud may be operated by the organization itself or a third party and may exist on premise or off premise [SK10, MG09, SJN10]. If the private cloud is operated by a third party off-premise, it is similar to classic outsourcing. However, cost advantages of private clouds are mainly driven by virtualization technologies.

This model is targeted at customers with high data- and security requirements, higher SLA requirements and/or specific legal requirements.

Hybrid Cloud This model is not a model in itself but actually a mixture of the above mentioned three deployment models. A hybrid cloud is independently managed but its data and applications can have the ability to burst outside of one model into another. An example of this is a cloud deployed as a private cloud that uses resources from a public cloud in cases of very high demand [SK10, MG09, SJN10].

This model is suitable for customers looking for a scalable, standardized and cost efficient solution

that also cover dedicated security requirements. The combination of both approaches leads to complex implementation architectures.

2.1.5 Economic Aspects

The reason why CC is so widely discussed at the moment is that it seems like it bears a lot of economic benefits for companies. This section describes and discusses these [SJN10].

Cost Reduction The desire to reduce costs is omnipresent in companies today. Thus it is one of the primary drivers for companies to rely on cloud infrastructures. Cloud infrastructures enable companies to pay only for the IT they actually need and when they need it, instead of continuously supporting an infrastructure that is sufficient to handle their worst case computing needs. CC promises to reduce cost for procurement and infrastructure maintenance for customer companies.

Especially relevant for this economic aspect are the two aforementioned characteristics (Section 2.1.2 Pg. 6) of CC *Metered by Use* and *Scalable and Elastic*.

Pay per Use Charging customers only for resources they actually consumed is a very important aspect. This aspect is one of the primary economic drivers of CC since it enables companies to decrease their upfront investments and shift it to operational expenditures.

Improved Time to Market This aspect is especially interesting for small and medium enterprises (SMEs) since they usually do not have the required financial liquidity to quickly acquire and setup an IT-infrastructure. CC, however, provides SMEs the ability to deploy their product on an infrastructure they can hire very fast and use it according to their needs. In this way, CC can help SMEs bridge the gap of financial liquidity, enabling them to be more competitive with large enterprises.

However, even larger enterprises can benefit from using CC in order to accelerate their time to market since even they need to be able to publish new capabilities with little overhead to remain competitive.

Turning CAPEX into OPEX For a company acquiring and setting up a local IT-Infrastructure CAPital EXpenditure (CAPEX) is required. Using a Cloud infrastructure transforms this CAPEX into OPERational EXpenditure (OPEX). OPEX is an on-going cost for running a product, business, or system. Transforming CAPEX to OPEX delays the upfront outlay of funds. From a business perspective there is a discussion regarding whether this transformation is financially beneficial or not [Gol09].

Return of Investment (ROI) ROI is a measure for investments by describing the ratio of money gained or lost on an investment relative to the amount of money invested. One could argue that this point is not relevant for CC since the transformation from CAPEX into OPEX eliminates the need for investments. However, the customer still needs to invest in order to make

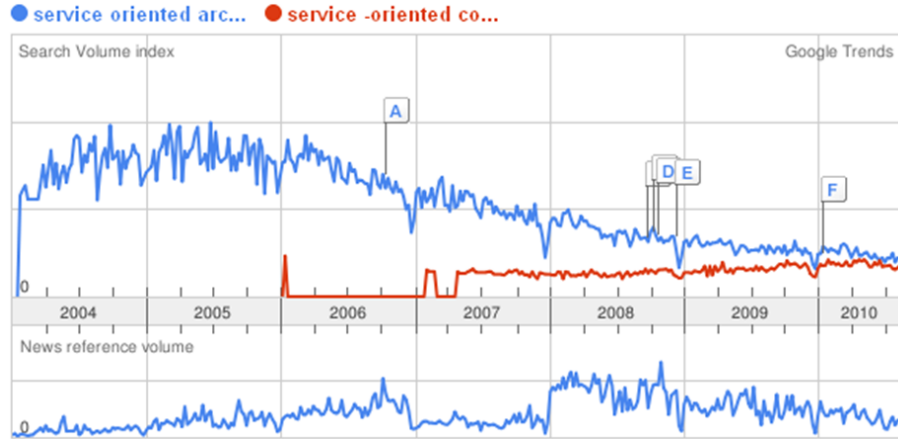


Figure 2.3: Service-Oriented Computing (red) and Service-Oriented Architecture (blue) on Google Trends [Goo10d].

use of the cloud. Thus some say that the investment could be cut by at least 90%. This, however, mainly depends on the actual project in which CC shall be used [Gol08].

Green ICT Green ICT or Green IT has been a hype topic for quite some time and its goal is to reduce the energy consumption of ICT systems as well as the carbon footprint. Of course for most companies the most appealing aspect of Green ICT is to reduce cost by reducing energy consumption. However, since CC deals a lot with trying to use resources in the most efficiently way (e.g. through virtualization), CC also provides the benefit of Green ICT. This can also be seen more and more with the operating of computing services from geographical regions that have more abundant non-carbon source energy available such as geothermal or wind.

2.2 Service-Oriented Computing

Another topic that is important to be familiar with for the next chapters of this thesis, is Service-Oriented Computing (SOC). SOC has already been discussed for a couple of years. Especially the related topic Service-Oriented Architecture (SOA) was a hype topic for a few years. Figure 2.3 shows² that the SOA hype has settled. However, it is a relevant topic for this thesis. As shown in figure 2.3 the term SOC has never been very popular since it is mostly used only in academics.

The following section gives an introduction to SOC and SOA.

2.2.1 Introduction

Traditional approaches to software development using Computer-aided software engineering (CASE) tools and modeling frameworks have proven to be appropriate and beneficial for building individual software components [HS05]. Once business needs change, applications have to be

²Similar to the figure in section 2.1 (Pg. 4) this figure also shows how often the term SOC and SOA were entered relative to the total search-volume across various regions of the world, and in various languages

adapted. Many of those old fashion applications, however, face significant problems in this case [GMKM04].

Service-Oriented Computing (SOC) is a paradigm that promises to solve that problem by using services. Services perform functions that range from very primitive things like answering a simple request to executing sophisticated business processes steps or even entire business processes [PTDL07]. A service hides the tasks that are performed for the user behind an interface. Furthermore, a service is supposed to be used in multiple contexts.

The goal of SOC is to easily assemble application components packaged in services into loosely coupled networks of services that can create dynamic business processes and agile applications [PTDL07].

The key to realizing this vision is the Service-Oriented Architecture (SOA). SOA is a logical way of designing a software system to provide services to end-user applications or other services distributed in a network. Services that are composed of other services in order to achieve more value are called composite services. A well structured SOA gives companies the ability to quickly adapt to changing business needs since they own a flexible infrastructure that allows them to provide independent, reusable application functions as services [PTDL07].

Web services are currently the most promising SOC-based technology. They primarily use open Internet-based standards like the Simple Object Access Protocol (SOAP) for transmitting data, the Web Service Definition Language (WSDL) for defining services, and the Business Process Execution Language for Web Services (BPEL) for orchestration services [PTDL07].

The Delivery Platform

Before it is possible to analyze the profitability of Cloud Computing (CC) applications for vertical industries, it is necessary to set up the environment in which these applications are managed. Special about vertical industries is that they are much smaller markets than horizontal industries [Szy02]. Thus, it is necessary that a platform that is used to build, deploy, and deliver this kind of Software-as-a-Service (SaaS) applications does that in a cost efficient way. Thus, this chapter's purpose is to describe such a platform.

It is commonly agreed that Information Technologies (IT) should never be an end in itself, but should always serve business goals by supporting the business [CVC03]. This is why the structure of this chapter represents this purpose of IT.

Based on this structure the chapter starts by presenting the vision of such a platform from a business perspective. This vision is then explained in a more detail way by defining a few abstract requirements. These requirements are still independent of IT paradigms.

After these requirements are defined the chapter continues by presenting different IT paradigms that can be utilized to fulfill the requirements. Based on these paradigms the integration platform is described and a reference architecture is proposed. Furthermore, the chapter describes how applications can be built that can serve vertical industries in a suitable way.

In order to conclude the chapter the deployment model is discussed, in which the integration platform and the SaaS-Applications can be deployed.

3.1 Vision

The vision of the platform can be summarized in the following definition:

A horizontal Software as a Service delivery platform that enables the financially beneficial development, deployment, and delivery of industry-specific applications.

In order to describe the vision in more detail, this section starts by providing high-level requirements¹ that need to be fulfilled by the platform. By discussing the requirements the primary focus is to emphasize the operator's benefits. These high-level requirements also serve as points of reference for the entire chapter. Future sections of this chapter refers to its specific requirements, discussing different characters or architectures of the platform.

The most important requirements are:

1. The platform shall promote the efficient use of available IT-Infrastructure.
2. The platform shall provide applications to customers in a service-based fashion.
3. Customer's usage of the applications offered by the platform shall be measurable.
4. The platform shall promote extensive reusability of individual components on an inter-application level.
5. The platform shall give applications that are deployed on it, the flexibility to be adaptable to customer's needs.
6. The platform shall allow customers to access and use the provided applications over the Internet by utilizing a standard web browser.

The requirements are designed to describe a SaaS delivery platform that supports operators by developing, deploying, and delivering SaaS applications to their customers in the most beneficial way.

The efficient use of IT-Infrastructure (Req. 1) is a major advantage for operators. The sharing of infrastructure by multiple customers, for example, helps operators use their IT-infrastructure more efficiently. Since this avoids execution of single instances of applications for every individual customer on a single physical machine, and, therefore, costs can be reduced. This allows operators to save on the cost or procurement, operation, and energy by reducing the actual physical machines that are required [CC06b, PF09, BYV⁺09].

The service-based provisioning of applications (Req. 2) implies a strict separation of concerns. Customer concerns are abstracted and uncoupled from provider concerns based on well-defined service interfaces. These interfaces hide the implementation details and complexity from the customer. The articulation of the service's purpose is described based on IT outcomes (availability, response time, performance versus price, and clear and predefined operational processes) and service levels, rather than technology and its capabilities [PF09]. This strict division between customer and operator concerns gives operators the ability to advertise concrete business solutions without having to deal with technical details. Customers, however, can focus on the actual functionality that is covered by the application and do not have to care about the technical implementation [PF09].

Measuring the customer's usage (Req. 3) is essential for operators in many ways. Primarily, this

¹Since these requirements are very abstract and describe aspects of the platform on a very high level. They do not follow and standard for the requirement creation like IEEE 830-1998.

data can be used to bill the customer [PF09]. Moreover by analyzing this data operators can gain conclusions about how profitable their offerings really are and how they need to adjust their business models in order to make them more profitable in the future. Furthermore, this data helps them to understand, control and optimize the usages of their infrastructure [PF09]. This is critical to ensure that necessary physical resources will be available to support the SLAs of their customers.

The ability of the platform to promote reuse on an interapplication level (Req. 4) can be extremely beneficial for operators since it allows them to shorten their development cycles [SS07]. Short development cycles and short time to market minimize the capital an operator has to invest in order to build a new offering. Reuse of already developed components across multiple offerings is a boost in development times. Short development times and low investments are especially important for the development of vertical applications since markets are naturally smaller than those of horizontal applications. Furthermore, a short development time enable operators to react in a fast and agile way to a change in an existing market or the emerging of a new one. This ability might give them the competitive advantage to dominate the particular market [Szy02]. In addition, reuse promotes better quality by using existing components that have known quality [BC95].

The flexibility of applications (Req. 5) to adapt to customer's needs is a crucial requirement for every application and every market. For vertical applications, however, there is an even higher importance due to the following fact. Vertical Markets are smaller than horizontals thus is ever more important to be able to address a significant market share. Since these are exactly the processes that shall be supported by applications deployed on the platform, it is crucial that the platform described in this thesis, promotes this kind of flexibility.

Giving customers the ability to access and use the provided application through a web browser (Req. 6) gives the operator a competitive advantage over the providers of common desktop applications. This is due to the fact that customers do not have to install and maintain any special software on their client computers. The only thing they need is a web browser that is standard on almost every client anyway [CPJ05].

One very important requirement that was not mentioned so far is the need for security and privacy. These two points are major reasons many companies and individuals are reluctant to make use of SaaS. However, most security problems related to SaaS web-applications are not unique for SaaS-Applications. But they are also known for web-applications. In this area they are addressed and there are techniques to tackle them [CPJ05]. Since the specifics of web-applications are not addressed in this thesis so will not security problems which are related to web-application. However, there are security concerns that are not related to web-application but due to the architecture that is going to be proposed. These concerns will of cause be addressed in the following sections.

3.2 Approach

In order to do the conceptual design of the platform according to the aforementioned requirements this section provides an overview of the approach that is utilized in this thesis.

The goal of this particular section is to show which paradigms and technologies are needed to build the platform according to the proposed requirements. The table 3.1 shows how Cloud Computing (CC) and Service-Oriented Computing (SOC) can be used to build web applications that satisfy the platform's requirements as they were proposed in the previous section. The first

Paradigm	Requirement					
	Efficient use of IT-Infrastructure (Req. 1)	Service-based (Req. 2)	Usage metering (Req. 3)	Promote Reusability (Req. 4)	Adaptation to Customer's needs (Req. 5)	Web Browser accessed (Req. 6)
Cloud Computing	✓	✓	✓	✗	✗	✗
Service-Oriented Computing	✗	✓	✓	✓	✓	✗
<i>Web based Application</i>	✗	✗	✗	✗	✗	✓
Result	✓	✓	✓	✓	✓	✓

Table 3.1: Illustrates how the used paradigms serve to fulfill the requirements.

paradigm, CC, can be utilized to fulfill the first three requirements. CC's characteristics *Shared Pool of Resources* [MG09, PF09] and *Scalable and Elastic* [MG09, PF09] as they were described in section 2.1.2 (Pg. 6) help operators utilize available IT-Infrastructure more efficiently (Req. 1). The requirement that applications on the platform are provided in a service-based fashion (Req. 2) is a part of the characteristics of CC as well. The CC's characteristic *Service-based* [PF09] (section 2.1.2 Pg. 6) provides this description. The last mentioned characteristic of CC *metered by use* [MG09, PF09] can be utilized to partially fulfill the requirement that customers' usage of the application needs to be metered. This can be done only partially since the CC infrastructure only provides data about the usage of the infrastructure but not of the exact usage of the PSP-Application that is executed when using it.

As illustrated by table 3.1, SOC helps satisfy the service-based provisioning (Req. 2) as well as promoting reusability (Req. 4) requirements. Both are due to the fact that a primary emphasis of SOC is to split an application into services that each provides business functionality [PTDL07, EHH⁺08a]. Each service hides the complexity and implementation details. The functionality that such a component provides through a service interface shall be as free as possible from the context of the rest of the application [SH05, EHH⁺08a, PTDL07]. Due to this fact, these components can easily be reused in another applications. The matter of reusability is discussed in more detail in section 3.4.2.

Another requirement that is fulfilled is the measuring of the customer's usage. This is due to the fact that usually a SOA-Suite also contains tools that allow for system monitoring [Gro09b]. Based on these tools it is possible to analyze the usage of the application of individual customers. Besides these two requirements SOC helps to satisfy another one. As table 3.1 indicates, SOC

also gives applications the flexibility that is necessary to adapt to different customer's needs (Req. 5). As mentioned in section 2.2 (Pg. 11) building an application in the SOC fashion involves the composition of services [EHH⁺08a, PTDL07]. Therefore, it is possible to change how services are composed for a specific customer and, thereby, adjust the application. The topic of customization to specific customer demands is discussed in more detail in section 3.5. Due to the usage of SOC the platform which is to be designed is called *Process and Service Platform* or short *PSP* from now on.

The last requirement that is open to be fulfilled is that the user shall be able to use the application through the utilization of a web browser. In order to fulfill this requirement it is necessary to equip the applications with a user interface that uses web technologies [CPJ05].

The following sections of this chapter show how the mentioned paradigms can be utilized to the PSP.

3.3 Integration Platform

This section deals with the technical integration platform of the PSP. Since the architecture paradigm that was chosen for the PSP is a SOA, a major part of the PSP is a SOA-Suite. This provides the infrastructure that is needed to fulfill the requirements for a service-based application provisioning (Req. 2), metering of user's usage of the application (Req. 3), promotion of reusability (Req. 4), and promotion of flexibility (Req. 5). The necessary components of a SOA-Suite are presented in the following sections. The chapter continues by presenting what needs are to be done in order to use CC to enable the sharing of IT resources. This sets up the environment to fulfill the requirements for an efficient use of IT resources (Req. 1), service-based usage (Req. 2), and metered usage of IT resources (Req. 3). This chapter concludes with a presentation of multi-tenancy as another technique to optimize the efficiency with which IT resources are used. This discussion contributes to requirement 1 (efficient usage of resources) as well.

3.3.1 Infrastructure Components

The biggest part of the integration platform is the SOA-Suite. Many vendors offer SOA-Suites. In this thesis, however, there is no recommendation of a specific product since products change frequently or even disappear. Instead, a generic vendor independent model is presented as it was defined by The Open Group [Pub09, Gro09b, Gro09a]. The Open Group is an industry consortium to set vendor- and technology-neutral open standards for computing infrastructure. In the following paragraphs the most important infrastructure components (IC) are described according to the definition by the Open Group [Pub09, Gro09b, Gro09a]. In this section IC refers to building blocks of the platform. Based on the presented integration platform it is possible to map specific products of vendors. The following list does not only contain ICs that are part of a SOA-Suite but also contains the supporting tasks that are necessary for PSP. The last two are such ICs.

Service Repository The reuse of services is one of the basic concepts of a SOA. In order to ensure that reuse can actually take place, it is necessary to clearly describe the individual service. These descriptions need to be available to all developers that might make use of the service. This is the purpose of a *service repository*: it allows for the maintenance and search a database of service descriptions.

PSP Purpose: Service descriptions of the individual PSP-Application components can be managed and searched for reuse.

Messaging Program A *messaging program* transports messages between other programs. One program submits a message to the messaging program, which delivers the message to a second program. In short, the message is sent by the first program, and received by the second. These programs are both consumers of the messaging service. In other words it is a bus system which connects individual programs.

Vendors of SOA-Suites often refer to messaging programs as the Enterprise Service Bus (ESB). The term, however, has many different meanings depending on the vendor. The term messaging program which is used here, on the other hand, describes a program that has only essential ESB functionality. Commercial ESB products should include this functionality, but often also have additional features. These include those of other building blocks that are described later in this section.

PSP Purpose: A *messaging program* is a necessary IC for the PSP since service interaction between different services is a basic requirement.

Activity Monitor An *activity monitor* is a program that interfaces to the messaging program and monitors the messaging activities. Its purpose is to collect data about the executed services.

PSP Purpose: The purpose of this IC is to measure activities that are performed in the application. Thus, it gathers the data that is necessary to analyze the user's activity and usage of the PSP-Application. This fulfills the requirement for metering of the user's usage of application (Req 3).

Policy Decision Points and Policy Enforcement Points “A *Policy Decision Point (PDP)* is a program that decides, in accordance with a policy, whether resource access requests should be granted. A *Policy Enforcement Point (PEP)* is a program that implements decisions to grant or deny resource access requests. PDPs and PEPs co-operate as part of an overall policy framework. Such a framework can be used to enforce policies of various kinds, including security and performance management. [...] When used in this way, they can control acceptance of messages for transport over the bus, and can control delivery of messages to services. PEPs can also be embedded within the services themselves. [Gro09b]”

PSP Purpose: Such a IC might be beneficial for the PSP in order to increase security. This IC might be especially important if a customer's system is hooked up to the PSP. In this case, access to this system must only be permitted to the owner of the system and not to all other tenants that share the PSP with him.

Data Translator A *data translator* is a IC that deals with conversion of data from one representation to another. It interfaces to a message bus and for example to a legacy system. It transforms the data that is provided by the system according to a specification into a format that can be handled by the message bus (messaging program) can handle.

PSP Purpose: The IC can be very useful in integrating the customer's legacy applications into their PSP-Applications.

Encryption Engine The *encryption engine* serves two purposes, the encryption and decryption of data, and it applies and verifies integrity checks based on encryption technology. When attaching this IC to the messaging program or sending and receiving data via the Internet, WS-Security is a commonly-accepted standard for the submission of data for encryption and decryption, and for application and verification of integrity checks.

PSP Purpose: Utilization of such a IC increases security for the PSP especially if external systems of the customer need to be integrated.

Event Processor “*The event processor detects input events with associated information, and generates output events with associated information. A single output event may correspond to multiple input events, and the information associated with an output event is derived from all the information associated with the corresponding input events. A common scenario is that the input events are messages on a message bus, and the output events are handled by analysis and presentation programs within an activity monitor. [Gro09a]*”

PSP Purpose: This IC can analyze the behavior of the PSP and notify the operator in case of an unpredicted event. Such an event could for example be an unpredicted peak in the customer's usage.

Composition Engine A composition engine is a program that executes scripts. These scripts control services and describe interactions between them. It is used to perform one or more of the services in a composition, as described in the model for scripted service composition. A script is written in a specific language that includes constructs for interacting with other services. The most common language for orchestration is the Web Services Business Process Execution Language (BPEL) defined by OASIS.

This interaction is often via messages, and composition engines are often integrated with messaging programs.

PSP Purpose: For the PSP a composition engine is used to build applications. This approach provides great flexibility to applications. Please refer to section 3.5 for further details. This flexibility directly contributes to fulfillment of the requirement for flexibility (Req. 5).

Service Registry A *Service Registry* is an organized collection of service descriptions maintained by a registry service that returns the descriptions that match specifications in submitted enquiries. The service registry has two purposes: first, the management of service descriptions that it maintains and, second, the enquiries and responses. In contrast to the *Service Repository*

the *Service Registry* only contains service description.

PSP Purpose: This IC allows for the management and search of services by PSP-Applications. Thus, it contributes to the fulfillment of the requirements for reuse and flexibility (Req. 4 and 5).

Service Components “A service component is a program that plays a part in performing a service. Service components include programs that implement service interfaces using underlying assets. Service component building blocks that implement service interfaces must conform to the same interface standards as the services. [Gro09b]”

PSP Purpose: For the PSP this IC is important since the PSP-Applications components actually need to be implemented and executed.

Model-Implementation Environment A model-implementation environment is a set of programs that support two things. First, it supports the creation of models for services or service compositions. Second, it enables the implementation of programs and scripts that perform the services and compositions directly from the models without substantial human intervention.

PSP Purpose: The PSP-Applications require tools for the implementation of a model and implementation environment.

Billing This IC is not part of the Open Group model. Nevertheless, it is necessary for the PSP since for some of the possible pricing strategies require data about the usage of the application (pricing strategies will be discussed in section 4.3 Pg. 41). Thus, this IC collects the necessary data from the Activity Monitor, matches the data with the business model of the application and forwards the data to the company-wide billing system.

PSP Purpose: This IC’s purpose is to calculate the price the customer is billed based on the usage and the business model of the application. Afterwards, the IC passes the data to the company-wide billing system.

Fulfillment and Assurance This IC is also not covered by the Open Group model. The IC gathers data about the overall performance of the application. This data is provided by the Activity Monitor and the CC infrastructure. The IC compares the data with Service Level Agreements (SLA) in order to find out if the SLAs were met.

PSP Purpose: This IC compares the actual performance of the platform with the SLAs in order to gather data about the health of the system. This data is important for operators since it allows, for example, the identification of bottlenecks and allows for the optimization of application performance. Furthermore, this is a possible way to find out if the proposed SLAs actually were fulfilled. Thus, the IC contributes to the fulfillment of the requirement for service-based provisioning (Req. 2).

3.3.2 Cloud Computing Foundation

In the previous section the basic infrastructure components of the PSP integration platform were described. These components would normally be installed on one physical machine in a data center.

According to requirement 1 the platform shall make use of available IT-Infrastructure efficiently another approach is necessary at this point. As discussed in section 3.1, this requirement can be

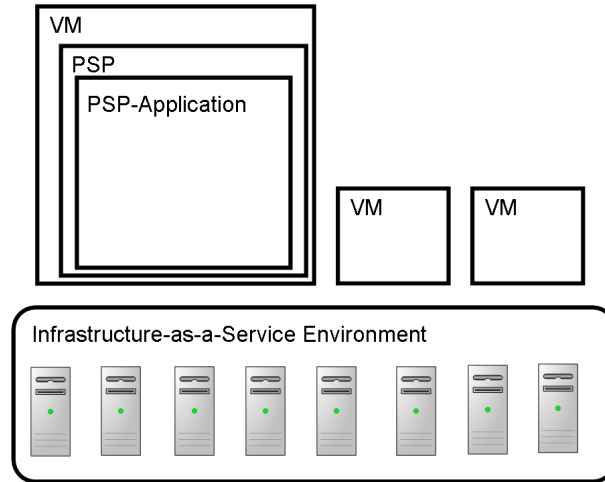


Figure 3.1: The PSP deployed in an Infrastructure-as-a-Service environment.

fulfilled by applying the Cloud Computing paradigm. This can be accomplished by installing the components into a virtual machine (VM) instead of a physical machine. This VM can be deployed in an Infrastructure-as-a-Service environment. Please refer to figure 3.1.

By doing so, the VM shares the available physical IT resources with other VMs. The PSP will be assigned computing resources according to the demand. Thus, requirement 1 is fulfilled.

The interesting thing about this point is that the SOA-Suite itself fulfills the definition of a Platform-as-a-Service infrastructure (section 2.1.3 PG. 7) for PSP-Applications.

3.3.3 Multi-tenancy

When designing a platform that is supposed to allow multiple customers to share available IT resources, CC is very beneficial. It is, however, limited to the sharing of resources by multiple VMs. By applying only CC every single customer or tenant would have their own VM. This would already provide great improvement in efficiency since it would not require a physical machine per tenant [BYV⁺09]. However, there is one more technique that promises to improve efficiency even further: multi-tenancy.

Multi-tenancy marks a significant change in how applications are developed, deployed, and delivered. The traditional and common way of deploying an application to serve the customer is that a single instance of the application serves a single customer. In this context a single customer might refer to a single user of an entire company. In the world of multi-tenant applications, on the other hand, a single instance of an application is used by several tenants [CC06b]. This refers to customers in the context of this thesis. Figure 3.2 illustrates that. The multi-tenancy approach makes it possible to use available resources more efficiently. Of course this model bears significant challenges since the customers that use the same instance of the same application might actually be competitive companies. This is why security issues need to have a very high priority. In this case security includes the separation of different customer's data both while

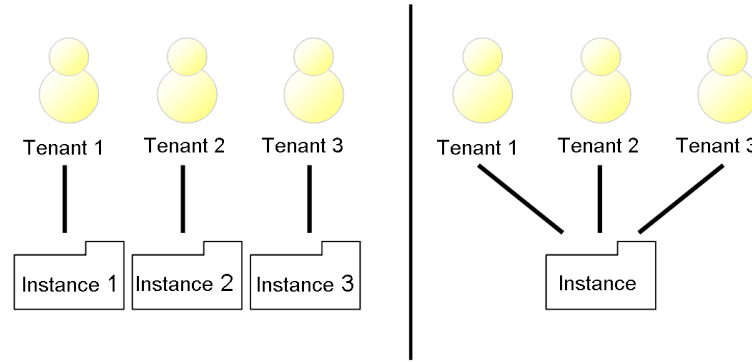


Figure 3.2: Contrast between Single-Instance and Multi-tenancy applications (adapted from [CC06b]).

stored and while processing [CC06b]. Furthermore, security and privacy concerns will require more complex architecture of the applications and will result in a more complex and longer development times [CC06b].

Another challenge when dealing with multi-tenant applications arises once the application needs to be adjusted to the needs of a specific customer. Naturally, an application is changed for one customer, the changes to the application exist for all tenants that use that instance. The only way to avoid that behavior is to make the multi-tenant application highly configurable on a tenant level instead of customizing the application for individual customers in the traditional single-instance way [CC06b].

Since sharing resources is one of the primary characteristics of CC it is a requirement for the PSP too (Req. 1). Multi-tenancy offers potential to help to fulfill this requirement even further if the PSP supports the development of multi-tenant applications.

One might argue that it is not actually necessary to support multi-tenancy since the IaaS infrastructure already promotes the sharing of resources and the single-instance applications might be deployed in individual VMs.

The interesting thing about this argument is that just recently SAP has made a design fault in the tenant architecture of their product Business-by-design. A couple of development years later SAP discovered that their product that was originally aimed at the Small and medium enterprises (SME) market was only profitable for customers with a rather high number of employees. This meant that it was no longer profitable to offer the application to small enterprises. This decreased their market. The result of this was that SAP added multi-tenancy to the product after the product had been developed for years. This bad design and architecture decision delayed the Business-by-design significantly [Som09, How10].

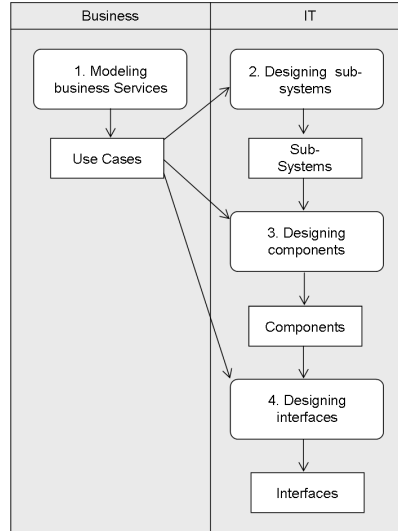


Figure 3.3: Overview of the method (adapted from [EHH⁺08b]).

3.4 Reference Architecture of Applications

The previous section presented is the integration platform as a part of the PSP. This chapter deals with the architecture of PSP-Applications.

In order to present this architecture this section proposes a method that can be used to design the applications in a SOA fashion. Thus, all PSP-Applications are designed in a common way. This promotes reuse and thereby contributes to the fulfillment of the requirement for reuse on an interapplication level (Req. 4).

After this method has been presented, the matter of reuse is discussed in more detail. The discussion of reuse leads to the conclusion that there are components that are usable almost any application. The matter of reuse of components is discussed in order to conclude this section.

3.4.1 Reference Architecture Method

This section introduces a method that can be used to design PSP-Applications. The reason why this method is proposed is that once a common method is applied to the development of many applications, these applications end up with the same structure [EHH⁺08a]. This promotes reuse of the components. Thus, this section contributes to the fulfillment of the requirement for reuse on an interapplication level (Req. 4 Pg. 14).

The method was adopted from the paper “A Method for engineering a true service-oriented architecture” [EHH⁺08b] and the book “Quasar Enterprise: Anwendungslandschaften service-orientiert gestalten” [EHH⁺08a]. This method, as presented by the two sources, discusses an approach to design an enterprise IT architecture based on a “true” Service-Oriented Architecture (SOA) [EHH⁺08b]. This approach can be seen as an instantiation of several enterprise architecture frameworks like *The Open Group Architecture Framework* (TOGAF) [Gro10], the *Integrated Architecture Framework* (IAF), the *Zachmann Framework* [Zac99], or the *Department of Defense Architecture Framework* (DoDAF) [oD10]. These frameworks provide terminology

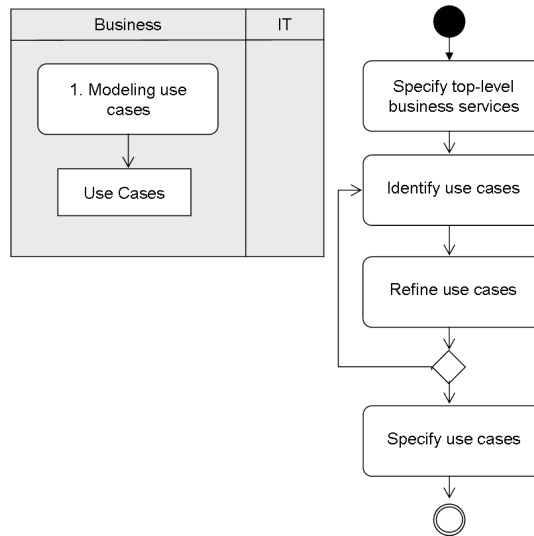


Figure 3.4: Overview of the overall reference architecture method [EHH⁺08b].

like meta models and abstract road maps. They do not provide concrete methods. This is where instantiations like the one presented can be used.

Based on this instantiation a model was developed to create the architecture of SaaS applications. This was done by adapting the method from the enterprise IT landscape level to the level of applications. Based on such an application architecture it is possible to create applications that can be adjusted to customers' demand very easily by composing applications from defined components.

Figure 3.3 illustrates an overview of the overall method. This overview figure indicates the different steps of the method broken down into method chunks. Each individual chunk will be discussed and explained in the following section.

One of the most important things about SOA and the mentioned frameworks is that they separate *Business* and *IT* aspects [EHH⁺08a]. This is indicated by the two swim lanes in figure 3.3.

The version of the method that is presented in the following sections was reviewed by several experts. It was attempted to publish this method as a research paper at the 2nd International Workshop on the Quality of Service-Oriented Software Systems (QUASOSS). This was done in co-operation with the Co-Autors Marcus Zinn, M.Sc., Dr. Stephan Verclas, and Prof. Dr. Bernhard Humm (one of the authors of the method on which this one is based [EHH⁺08a, EHH⁺08b]). The paper that was handed in to the conference can be found in B (Pg. 85). Unfortunately, the paper was not accepted, however very extensive feedback was given. This feedback can be found in chapter C (Pg. 92). The feedback the was related to the method has been implemented in the following section. A next attempted to publish the method is planned.

Step 1: Modeling Use Cases

Figure 3.4 shows an overview of this step. The focus of this step is to stepwise refine Unified Modeling Language (UML) use cases that the application shall provide starting from the business services. “A business service is the output of a service provider towards a service consumer:

goods, information, or activities. A business service is based on a service contract. It specifies in- and outgoing information and goods and the basic course of action.” [EHH⁺08b].

Applying this definition to the application level as it is discussed in this paper can be done as described in the following: a given application’s purpose is to cover one or more business services of a company. Since this paper discusses SaaS-Applications, the business services that are covered are those the customer wishes to outsource. Every individual business service that is covered is realized based on a series of actions. Due to the facts that this paper discusses the architecture of applications and UML use cases are widely accepted mechanism for analyze applications they will be used for this step. Use cases can be defined on different levels of granularity. The usual form, a hierarchy, is that coarse-grained services contain fine-grained services. Thus, the following method’s step will create a hierarchy of use cases starting from the top-level business services. Each business services for itself includes a number of use cases which need to be performed. These use cases contain each for itself again a number of lower-level use cases and so forth (functional decomposition). This structure can be displayed by utilizing a UML use case “includes”-hierarchy.

Since this step is located completely on the business swim lane the business services are not influenced by IT concerns. A possible input for this step is the business services the customer wishes to outsource and the requirement documents describing the functionality of how the business service needs to be fulfilled. At this point, however, it is important to keep in mind that the application is supposed to serve multiple customers. This means that it might be possible that different customers have different requirements. This is were the flexibility of the application needs to be able to cover these possibilities.

The goal of this method is to create the mentioned use case hierarchy by following a defined method (fig. 3.4).

1. *Specify business services:* This first sub-step’s focus is to specify the most coarse-grained functionality the application shall provide. These business services are the ones a customer wishes to outsource.
2. *Identify use cases:* Use cases that need to be performed within a business service are candidates for the next finer-grained level of use cases.
3. *Refine use cases:* A use case is refined if the following condition holds.
 - (a) The entire purpose the coarse-grained of use case is covered by the finer-grained use case (additional fine-grained use cases need to be added).

Sub-steps 2 and 3 are executed until either a suitable level is reached or elementary use cases are found.

4. *Specify use cases:* A suitable way would be to apply use case diagrams and use case descriptions as they are defined in the UML.

The result of this method’s step can be illustrated in a tree showing every top-level business service on the first level and the included use case it consists of on the next. Therefore, this

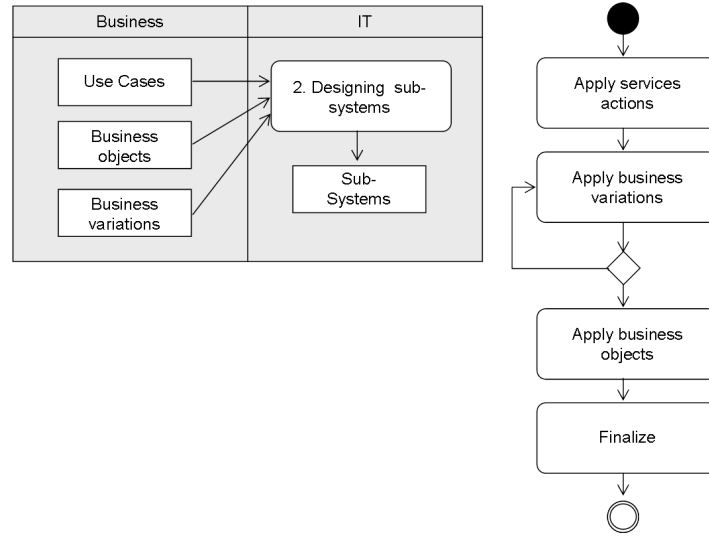


Figure 3.5: Reference architecture: designing domains (adapted from [EHH⁺08b]).

graph might have many levels. The business services and use cases should be documented by using a UML use case diagram. Each individual use case should be documented by using use case descriptions.

Step 2: Designing Domains

The second step of the method represents the passage from the business to the IT swim lane. Figure 3.5 gives an overview of this step. Building an application based on the SOA paradigm means that an application is structured according to the business [EHH⁺08a]. This is why at this point inputs from the business are used to design subsystems that structure application services. Figure 3.4 refines the overview method in figure 3.3 by identifying three step inputs:

1. *Use cases*: at this point only the top-level use cases are used. These are the ones that are right below the business services.
2. *Business objects* are the most important top-level items that are handled by the application.
3. *Business variation* reflects different ways of executing a specific process step. In case multiple customers have different approaches to the same process step, these would be variations. The specific customer can then decide to use variation A or B in their application according to their business needs.

Based on these inputs the sub-systems are designed. Sub-systems represent in the context of this architecture the highest level (top-level) structure of the application. In this step of the method, sub-systems are designed as illustrated in figure 3.5.

1. *Apply use cases*: Initial sub-systems come from the top-level service actions.

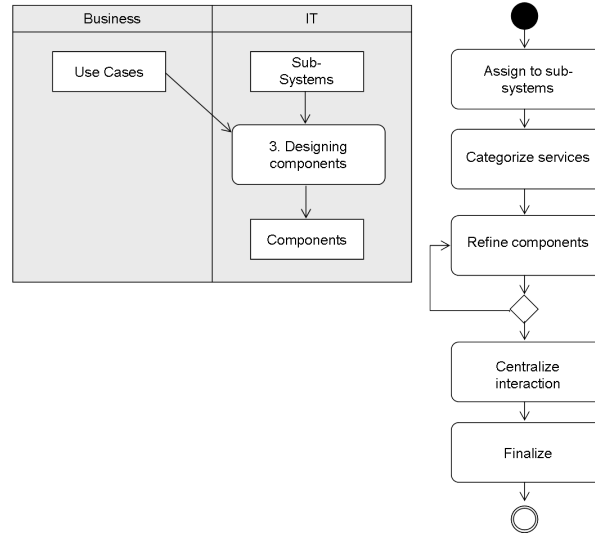


Figure 3.6: Reference architecture: designing components (adapted from [EHH⁺08b]).

2. *Apply business variations:* Sub-systems are split according to the characteristics of one or more business variations. If the variation between two different strategic approaches is too big to be handled by configuration of the same sub-system, it needs to be split in two. This decision needs to be done after careful consideration and only with insight into the industry [Die06].
3. *Apply business objects:* Every business object needs to be analyzed in which sub-system it is generated, modified, or deleted. If an object is created, modified, or deleted in more than one of the existing sub-systems, the object will get its own sub-system called “*ObjectName Manager*”.
4. *Finalize:* Sub-systems are given meaningful names. The sub-system model can be illustrated by using an appropriate graphical representation such as UML packages.

The output of this step is the definition of sub-systems. These definitions will be the input for the next step of the method.

Step 3: Designing Components

The third step of the method is located entirely on the IT swim lane. Figure 3.6 gives an overview of this step. This step presents a method to define components. A component is defined as a container that contains a large portion of business logic and it exports and imports interfaces. Thereby, it is part of a certain sub-system. A sub-system may contain many components. A component, however, may belong to one subsystem only. An interface groups operations of a component and specifies the protocol used. An operation is described by its signature, its semantics and non-functional properties.

The components are created based on the sub-systems that were defined in the previous step. Furthermore, other inputs for this step are the use cases as they were defined in step 1. The

most high-level ones were already used in the previous step. This time, however, the lower-level use cases are applied.

Components are created according to the process illustrated in figure 3.6.

1. *Assign to sub-systems*: Use cases from the hierarchy created in step 1 are assigned to components. Due to the split of sub-systems according to business variations it is possible that a single use case will be offered by multiple sub-systems.
2. *Categorize service*: It would be possible to categorize the components from a technical [Zin07] or from a business perspective [EHH⁺08b]. The difference between both views is only minimal. For this paper, however, the following business perspective is chosen:
 - (a) *Data*: Managing business objects.
 - (b) *Function*: Providing algorithmic business logic.
 - (c) *Process*: Providing flow-oriented business logic.
 - (d) *Interaction*: Allowing the user to interact with the application.

The use cases are categorized according to those four categories. Components are designed that have exactly one category, implement one use case and belong to one subsystem.

3. *Refine components*: The components that have been constructed so far are split according to the following rules:
 - (a) Business logic that changes at a different pace shall be separated.
 - (b) Transaction data shall be separated from static data.
 - (c) Components shall not have cyclic dependencies.
 - (d) Components of different categories shall have dependencies according to a layering
 $interaction \rightsquigarrow process \rightsquigarrow function \rightsquigarrow data$.
4. *Centralize interaction*: Group interaction components from different subsystems into new, dedicated subsystems .
5. *Finalize*: The components are named appropriately and checked for completeness with respect to the covered services.

Step 4: Designing Interfaces

Figure 3.7 gives an overview of this step. The focus of this step is on the application services. After the components were defined, their interfaces and operations need to be constructed. This can be done by executing the following process:

1. *Use service actions for operations*: The actions of the use cases become operations.
2. *Apply rules*: If the following rules do not apply to the defined operations, the operations need to be adapted accordingly.

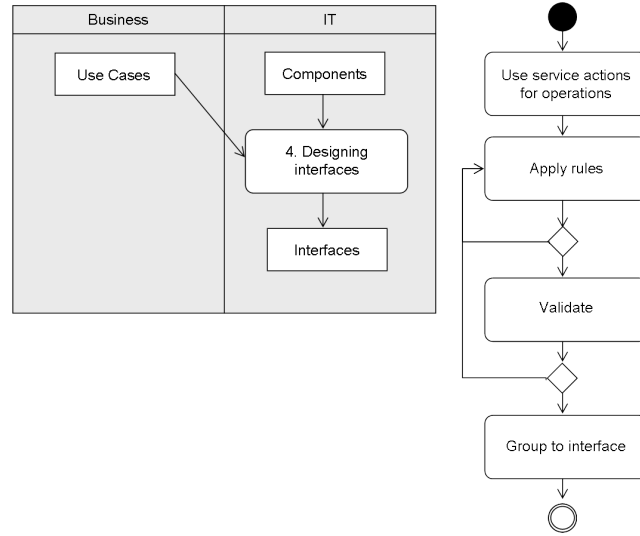


Figure 3.7: Reference architecture: designing interfaces (adapted from [EHH⁺08b]).

- (a) *Business-oriented*: Every operation shall provide business logic only and must not reveal implementation details.
 - (b) *Coarse-grained*: Operations shall comprise as much business logic as possible.
 - (c) *Idempotent*: Multiple invocations of an operation with the same parameters shall have the same effect as a single invocation.
 - (d) *Context free*: Operations shall have minimal knowledge on the context in which they are invoked (session context, transaction context, batch/online).
3. *Validate*: A check for completeness with respect to the covered services is performed. This is done by creating a UML sequence diagram for each application service, using the components and their operations.
 4. *Group to interface*: The operations constructed are grouped to interfaces according to:
 - (a) user groups.
 - (b) access mode (read/write).

This is the end of the method. Once this method was applied to an application the architecture or structure of an application should be defined down to the service. At this point decisions need to be made on how these services can be implemented. This could be done by developing the required systems from scratch or by buying third party systems which can even fulfill the tasks of the components of even domains.

In chapter 5.1 (Pg. 57) the described method is demonstrated by applying it to a specific example.

3.4.2 Reusability

If the method that was just presented is applied to an application, the result is a very clear architecture that aligns the application with the business [PTDL07, EHH⁺08a, HS05, Pel03, SH05]. Once this is accomplished it is possible to reuse individual components for other applications since they serve a specific business purpose that certainly can reemerge in another application.

Naturally, reuse of such components is very likely especially for applications that serve a similar overall business purpose [EHH⁺08a].

However, another degree of reuse offers components that service a requirement that every application has. Such a component that would be a multi-tenancy aware would be the database communication layer. Such a component could be used to offer a interface to a database that gives the application developer the illusion that there is a single database for every application. In order to do that, this component would have to encapsulate the complexity of matching an user-neutral requests with the identity of the user and translates this to access a database that stores the data for multiple users. For years database abstraction layers are standard tools in software development (Open Database Connectivity (ODBC) is just one example).

Such a component would be beneficial for every application since it hides complexity and simplifies development.

3.5 Application Building

Since the last section presented an approach for how the applications can be structured in a SOA way, this chapter deals with the building applications based on this architecture. The major goal at this point is that the applications that are offered to customers need to be extremely flexible. Due to this fact this section starts by discussing the term *flexibility* and distinguishes between different flexibility strategies.

Based on this discussion the approaches for flexibility in PSP-Applications are described. This section mainly addresses to the requirement for the delivery of flexible applications which can be adjusted to customer needs (Req. 5).

3.5.1 Flexibility Strategies

Flexibility concepts are of significant interest to a wide range of research areas like manufacturing, economics, strategic management, and Information Systems (IS) [GS06]. The term *flexibility*, however, is extremely imprecise. A study about flexibility in manufacturing, for example, revealed that it is possible to distinguish eleven different kinds of flexibility [SS90].

This is why it is necessary to discuss the term a little further before talking about flexibility of PSP-Applications.

Especially for IS, two general types of flexibility [HMH96, GS06] that are important for this thesis: *Flexibility in the pattern of use* (Flexibility-to-Use) and *flexibility for further changes* (Flexibility-to-Change) can be distinguished between .

Both can be described as followed.

Flexibility-to-Use This kind of flexibility is related to manufacturing machine flexibility [SS90, GS06]. For IS this represents the number of of users' requirements the IS can cover without requiring a major change of the IS.

System functionality This describes the different features the system provides to a user.

Scope of an IS database This is the number of items that can be processed by a system. An example would be the number of product categories that that can be purchased through the catalog of an electronic procurement system.

User interface This refers to the different features and methods an IS provides to a user to interact with the system. This also includes the different channels a user can use to interact with the system (e.g. desktop computer or mobile device).

Processing capacity Refers to the number of users that can use the IS simultaneously.

Flexibility-to-Change This kind of flexibility describes an applications extensibility and the effort necessary to implement a major change. The possibilities for this range from systems that cannot be changed at all to systems to systems that promote changes by including a large set of opportunities. For example does the system modular, allowing multiple components to be added to or removed from it.

In order to distinguish the two different kinds of flexibility it is necessary to define what is meant by a *major change*. A major change of the IS can be defined as adjustments and modifications that require a fresh system setup, re-installation or re-testing [GS06]. Changes that cause only minor disruptions like changing pre-installed parameters are not considered a major change thus belong to flexibility-to-use [GS06].

3.5.2 Flexibility

Since the PSP's primary purpose is to serve as an SaaS delivery platform especially for vertical industries, it is extremely important that these SaaS applications can be customized according to customers' demand (Req.: 5).

Since vertical industries have naturally smaller markets than those of horizontal applications it is much more important that the PSP-Applications can be adjusted to a customer's demand. At this point the flexibility of the application mainly determines how many customers can be reached.

The important theme at this point is that PSP-Applications are always designed to be multi-tenant applications (please refer to section 3.3.3 (Pg. 21) for more details). Due to this fact it is not desirable to provide individual instances of applications to customers that are customized by using flexibility-to-change strategies. As a matter of fact for PSP applications and all other multi-tenant SaaS applications it is necessary to build enormous flexibility-to-use. This included

every aspect of flexibility-to-use that was mentioned in the previous section. PSP-Applications need the following:

- to cover the functionality that a profitable share of a market desires (System functionality).
- to possess a database that is as flexible as necessary to cover all cases that are relevant for the customer group (Scope of an IS database).
- needs to have a customizable user interface. Examples would be the customization of interfaces with the customers' company logo (User interface). Furthermore, it is necessary to provide the channels of interaction that the targeted customer group requires.
- to be able to handle as many users as the target customer group has. Regarding this point multi-tenancy is especially important for the PSP (this was described in section 3.3.3 Pg. 21) (Processing capacity).

Within the available flexibility-to-use the applications are customized according to user's demand on a tenant-specific level.

It is desirable that the flexibility-to-change strategy allows to changing the application in a easy way without having to test the entire application. Choosing the SOA paradigm as technical architecture for the PSP promotes changes of PSP-Applications [PTDL07]. However, due to multi-tenancy there are points for which one need to be aware. If a change to the system is made, the change affects all tenants [MMLP09]. That means if the change is supposed to be made available only to a subset of all customers it is necessary to create the change in a way that the change must be activated through configuration.

An example of a well-known system that promotes flexibility of SaaS applications is Salesforce.com, a CRM System. Users can customize Salesforce for their needs by adding custom user-defined fields to their customers [sal10a]. In this way they have the ability to store the data they actually need for of their customers. Furthermore, users of Salesforce have the opportunity to extend the standard application by their self-developed applications. This is another tool to add features to Salesforce on a specific vertical- or function-level (Finance, Human Resources, etc).

3.5.3 Customization of Applications

As discussed in previous sections the PSP uses a SOA to build applications. A SOA is well-suited as an architectural style for realizing SaaS applications [MMLP09, CC06b, GSH⁺07]. This is due to the fact that this architecture offers a flexible way to for building composite applications out of existing building blocks. For the PSP the term *building blocks* refers to the domains and components that were designed according to the reference presented architecture method². These components can be accessed through interfaces which are called services. The individual services of the application domains and components can be composed to produce complex applications. The SOA-Suite as it was described in section 3.3.1 provides components an mechanisms to allow composition which is also called orchestration [Pel03]. The *messaging program* and the

²The method was presented in section 3.4.1 Pg. 23.

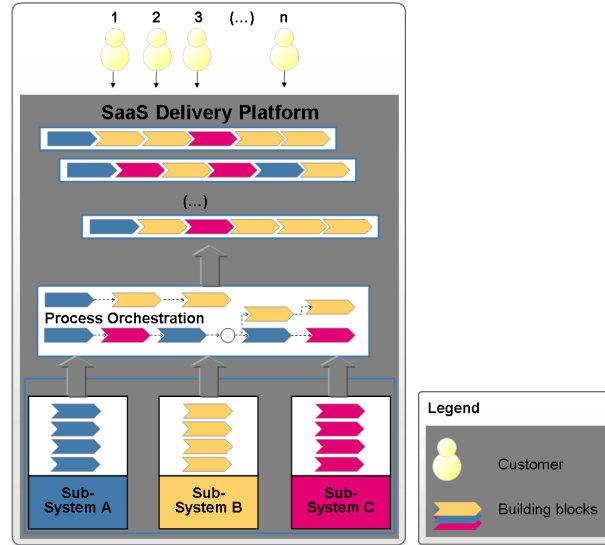


Figure 3.8: PSP Applications can be changed through Orchestration.

composition engine are the primary components that provide this functionality.

Figure 3.8 illustrates how applications can be composed for individual customers. As illustrated by the figure a process is orchestrated for a specific customer. The customer has to describe the the process the PSP-Application is to cover. Then it is possible for the operator to compose existing components or domains into an application that fulfills customer's needs.

When doing so it is not necessary that all components that exist are actually used by this specific customer. On the other hand however it is possible that a customer wants to integrate their own systems into the process. This is possible due to the ability of the SOA-Suite for to integrate of third-party systems.

The process model that was designed needs to be stored for a specific tenant and is only accessible to that customer.

This approach allows a higher degree of customization in contrast to monolithic SaaS applications that are deployed according to the multi-tenant model and configurable via meta-data [MMLP09].

3.6 Deployment model

In the previous chapter (section 2.1.4 Pg. 8) the different deployment models of Cloud Computing (CC) were discussed. The delivery model chosen for a particular service is determined by the non-functional requirements of its users. The ability to fulfill needs for data security and data privacy as well as the price is mainly influenced by the chosen deployment model.

Due to this fact it is important to anticipate customers' needs when determining an appropriate deployment model for the PSP and the PSP-Applications. One of the main goals for the PSP is to deliver industry specific applications for a reasonable price. Thus, deploying the PSP and its applications in private cloud for every individual customer would be contradictorily to this goal because of the high infrastructure cost. The public cloud deployment model, on the one hand, would provide the PSP with the cheapest possible infrastructure. However this model is

associated with the more potential security and privacy risks. This indicated that the community cloud model might be a suitable model for the PSP. Applications developed and deployed based

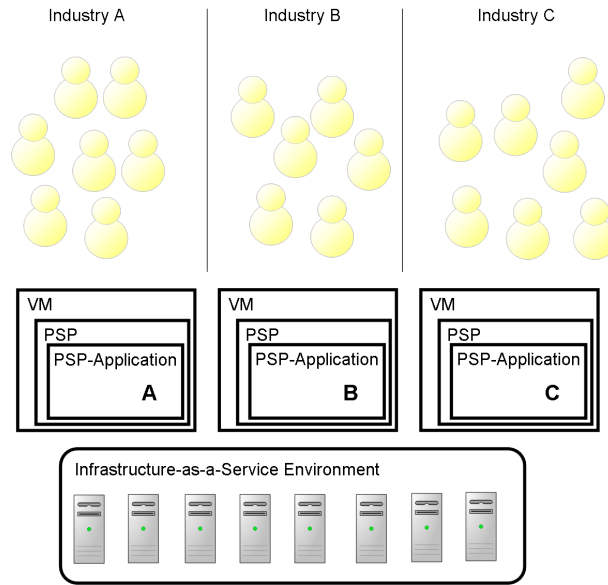


Figure 3.9: PSP based on the deployment model community cloud.

on the PSP can be used by multiple tenants at the same time without interference between the different tenants. Due to this fact a single instance of the application can serve multiple tenants at the same time (refer to section 3.3 Pg. 17 for further details). In order to execute this application an instance of the PSP is necessary. This PSP instance is shared by multiple tenants indirectly through the usage of the application. As a matter of fact a single instance of the PSP would actually be able to execute multiple applications that each in itself serves a particular vertical industry.

However, by separating different applications into their own PSP instance packaged in their own Virtual Machine, security issues can be improved without increasing the price significantly. This is due to the fact that a PSP-Application is never used by a single tenant since this would not be profitable to the operator of the application. Furthermore, since the packaged VMs are deployed in an IaaS infrastructure, they will only be provisioned with the resources they actually need. Figure 3.9 illustrates the described deployment.

If the PSP is deployed in this way that has just been indicated, it fulfills the definition of a community cloud deployment according to the proposed definition that was presented in section 2.1.4 (Pg. 8). In this case the distinct group of organizations (the community) sharing the infrastructure is the different customers that use the same application. This means the customers that belong to the same industry.

Should a single customer/tenant require a higher security, it would be possible to deploy a single instance of the application and the PSP in a private cloud for this particular customer. This, however, would increase the price the customer needs to pay. This, again, contributes to the goal to bill the customer only what they need.

In the previous chapter the conceptual design was made for a platform that allows one to develop, deploy, and deliver industry specific applications in as financially beneficial way. This platform was called Process and Service Platform (PSP).

Based on this conceptual design this chapter deals with the business analysis of the platform. In order to do this a mathematical model is created. This model allows one to determine under which circumstances PSP-Applications can be profitable. Input for this chapter is the previous chapter 3. Figure 1.1 (Pg. 3) illustrates this chapter's purpose in the context of the entire thesis. This chapter structures as follows: In section 2.1.3 (Pg. 7) the concept of Software-as-a-Service was discussed briefly as a possible service model of Cloud Computing (CC). Since this chapter's purpose is to analyze the profitability of PSP-Applications, it is necessary to start by discussing SaaS in more detail. A special focus is set on the impact on the customer's and the operator's business. Based on this and the input from the previous chapter it is possible to analyze the target market of PSP-Applications and afterwards propose a strategy for pricing new applications. These discussions are used to create a mathematical model which estimates profitability. The chapter concludes by analyzing the proposed model in case-independent way.

4.1 Analysis of SaaS

SaaS represents a dramatic change of paradigm how the business with software works and how IT is managed within companies [CC06a]. This change and its impact on software vendors and customers will be discussed in this section.

In order to do that, this section starts with a comparison of SaaS versus the traditional perpetual license model. Based on this comparison the section will continue by discussing benefits of providing SaaS-Applications instead of traditional software. In order to conclude this section the impact of SaaS for customers will be discussed.

4.1.1 SaaS versus Traditional License

The most significant differences between a common perpetual licenses and the SaaS model can be illustrated based on three dimensions - how it is licensed, where it is located, and how it is managed. Please review figure 4.1. These dimensions can be used to analyze SaaS-products. In

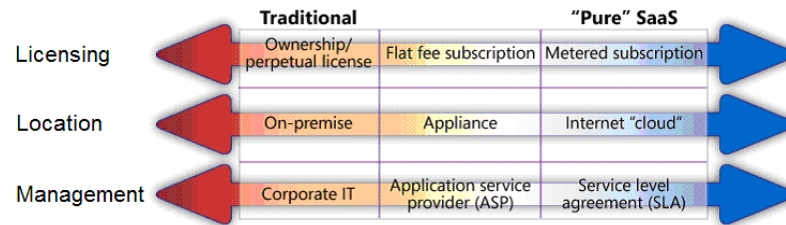


Figure 4.1: Difference between Traditional Licensing and SaaS [CC06a].

the “pure” form of SaaS, operators host an application on their own infrastructure and provide access to multiple users over the Internet in exchange for a fee. Each of the dimensions can be divided into three segments which describe different flavors of traditional licensing model on one end and the “pure” SaaS model on the other. The middle segment is an additional option which combines aspects of both models [CC06a].

Licensing In the traditional model of software sales, the customer buys a license for each user or site, or in case of custom built software the software is owned outright. This license is valid perennally and requires customers to make a upfront investment. In the SaaS model, on the other hand, the customer is only billed based on the usage of the application. The usage might be defined by parameters such as number of transactions, event, minute, click, operations or any other measurable attribute. This is also called *pay-as-you-use*. In between these two is the time-based subscription model. Applying this model means that the customer pays a flat fee per user and also pays a fixed fee for the particular period of time.

Location In the traditional model customers install and operate the software themselves using their own infrastructure. In the SaaS model, however, the operator installs and operates the application. In between these two there is the appliance model. In this model the software vendor supplies customers with a hardware/software component as a black box, which is installed at the customers’ site [CC06a].

Management In the traditional model, a company’s IT department is responsible for the provisioning of IT services to users. Thus, these departments are required to be familiar with all aspects that are necessary to accomplish this task. This includes the knowledge of networks, servers, and application platforms as well as providing support and troubleshooting. In the SaaS model, on the other hand, applications are completely managed by the operator. Service-level

agreements (SLAs) govern the quality, availability, and support commitments that the provider makes to the subscriber [CC06a].

4.1.2 Customers' View on SaaS

If a customer decides to use software that is provisioned using the SaaS model, there is a significant change in how the customer deals with this application. Since customers pay a fee based on their usage instead of buying a perpetual license, the initial investment a customer makes can be minimized. The potential for minimization, depends however on the level of customization that is necessary to make the application fulfill the customers' requirements. However, this price would probably be lower than for the common license since a common license would also require customization. In case customers are satisfied with a standard version of the SaaS-application there would be no initial investment [CC06a].

With no significant initial investment to amortize, an enterprise that uses SaaS-Applications can simply quit using the application in case it produces disappointing business results. In this case the company would realize a smaller loss than using traditional software. Thus, SaaS lowers the risk of software acquisition [CC06a].

Since the application does not need to run on internal infrastructure, SaaS helps the customer save infrastructure resources. But SaaS does not just have impact on the infrastructure that is necessary to run the application since most SaaS-Applications use web-based technologies.

These facts limit the technical responsibilities of customers to establishing a connection between the client computer and operators servers, and providing a web browser which is standard on most computers anyway. This saves SaaS customers cost for hardware as well as labor [CC06a, Leo09]. Furthermore, the job of keeping the application running, which usually requires jobs like installing patches, managing upgrades, performance monitoring, high availability insurance, and so forth, is handled by the provider of the SaaS-Application [CC06a]. Thus, the internal IT departments do not have to deal with these "overhead" activities. This allows IT departments to focus more on high-level activities that support the business goals of their company, instead of being primarily reactive and operations-focused [CC06a].

Besides these advantages SaaS also bears challenges. These challenges relate to SaaS being leased software which the customer does not own and that it runs on an operator site instead of within the company. These aspects mark the loss of control over the application. Thus, the decision to use SaaS within a company instead of traditional software boils down to the tension between control and cost [CC06a].

To make the decision whether to use SaaS or not a company needs to consider a number of points. Some of them are the following:

Political *"Sometimes, the decision can be short-circuited by resistance from within an organization, if important people insist that certain functionality remain internal, under the control of IT; other considerations therefore become unimportant. Test-drive deployments [...] might sometimes help convince risk-averse managers to approve pilot projects."* [CC06a]

Technical Since SaaS is designed to serve multiple customers there are limitations in flexibility. If a customer requires functionality that cannot be delivered by the SaaS-Application, it might make no sense to subscribe to this application [CC06a]. Another technical factor that needs to be considered is the type and amount of data that is transmitted to and from the application on a regular basis [CC06a]. An SaaS-Application that handles high amounts of data might not be usable because of insufficient bandwidth.

Financial The total cost of ownership (TCO) of an SaaS-Application needs to be compared to that of an equivalent on-premise application. In most cases the initial acquisition cost of SaaS is lower than that of on-premise applications. The long-term cost, on the other hand, might be a different story depending on the specific SaaS-Application and how it is billed [CC06a].

Legal *“Some industries are subject to regulatory law in different parts of the world, which imposes various reporting and recordkeeping requirements that your potential SaaS solution candidates cannot satisfy. Consider the regulatory environments in all the different jurisdictions in which your organization operates and how they affect your application needs.”* [CC06a]

In addition, technical and financial considerations can also have legal ramifications. An example for that are data security and privacy standards that need to met due to legal regulations.

4.1.3 Operators' View

Similar to the impact of SaaS for customers, SaaS certainly has impact on the operators' business. Their business changes from software vendor to SaaS provider [Son08]. This fact changes the level service that providers offer to the customer. Instead of being pure software forges they have to take over some of the tasks which were previously performed by the internal IT departments of their customers. These tasks include hosting the application, support, maintenance, backups, version updates and so forth.

Even if the responsibilities of the operator increase, there are mayor benefits and opportunities for cost saving involved.

For an SaaS-Application there is only one (or a few) productive instances of an application running. Those instances are under the control of the operator. Due to this fact it is the operator's responsibility to update the productive instances to newer versions. Since this is done directly without having to distribute updates to customers, it is possible to update the application in more frequent cycles with less effort [Son08]. This allows operators to react faster on security problems or feature requests. This helps to enable constant innovation through simultaneous release to all customers. Besides the obvious benefits for the customer this also decreases cost the operators has to spend for support and maintain since they have to support only one, the most current version, instead of multiple ones [Son08].

Besides these mentioned opportunities for cost savings there is one more very important opportunity. This opportunity is called *economies of scale* [CC06b]. Economies of scale refers to the cost advantages that a business obtains due to expansion. Economies of scale is a long run concept and refers to reductions in unit cost as the size of a facility and the usage levels of other inputs

increase [Nor79].

Although SaaS-Applications can not be referred to as manufacturing business, the model of economies of scale does still apply to it, and has implications. For example, an operator can provide SaaS-Applications installed on a server farm of five servers that supports 50 medium-size customers. This means that each customer is only responsible for a tenth of the cost of a server. In the traditional model each customer might have to use a single self-maintained server. This represents a significant opportunity for savings which gives operators the ability to set their prices to be more attractive to customers than traditional software vendors. A pricing strategy will be discussed in section 4.3 (Pg. 41).

4.2 Abstract Target Market

Based on the discussions of the previous section about SaaS in general, this section analyzes where PSP-Applications can actually be applied. Thus, this section starts by discussing for which applications the PSP can be most beneficial. Based on this discussion the section concludes by analyzing which customers can actually benefit from PSP-Application.

4.2.1 Possible Applications

As discussed in the beginning of this chapter, SaaS stands for the delivery of standard software to multiple customers. In fact, to be profitable an SaaS-Application has to be sold to many customers since only then do economies of scale result.

Considering this, the question arises as to which applications can actually be implemented based on the PSP. Due to the used components (section 3.3 Pg. 17) of the PSP and how applications are structured (section 3.4 Pg. 23) it is rather obvious that PSP-Applications should cover an entire or at least a part of a process. This is also due to the fact that when customers decide to use a particular PSP-Application, they outsource the IT that is necessary to fulfill these processes or parts of them.

Based on these considerations it makes sense to limit PSP-Applications to process-driven applications. This is reasonable especially with respect to the flexibility strategy that was proposed for PSP-Applications (section 3.5 Pg. 30). The reason is that changing the applications structure actually means changing the process which the application covers. Thus, applications are adapted to the process the customer has in place or wishes to use.

4.2.2 Target Customer Profile

When developing an application it is a fundamental requirement that there are actually customers that wish to use it and are willing to pay for it. However, an PSP-Application will never be able to cover all process variations that are possible for a specific process it tries to cover. Thus, it will not be possible to build an application that is usable for every customer that has a certain process in place and wishes to outsource it.

In order to illustrate this point please refer to figure 4.2. In this figure there is an PSP-Application

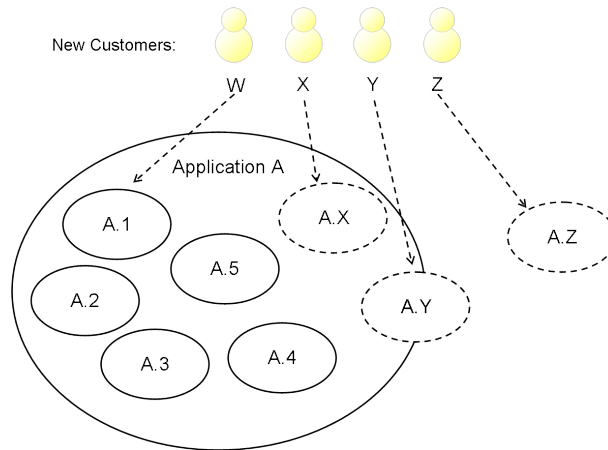


Figure 4.2: Illustrates the possible applications as they might be needed by customers.

called A (this is indicated by a big ellipse). Based on the components that were developed to realize application A, it was possible to define five different variations of the same application. A.1 - A.5 indicate these applications. Each of these applications implements the same process (process A which is covered by application A) but each is different from the others. However, each variation of the application is based on the same components which belong to application A. However, not every component that is part of the application must be used in every variation. If a new customer wants to use the application there are 4 different scenarios which require effort from the operator. These are illustrated in figure 4.2 with 4 “new customers”.

Customer W This represents a new customer who needs an application that covers an process that has already been provided for a different customer.

Implication Customer W can be served without any investments. It is only necessary to supply customer W with an application that implements the variation A.1.

Customer X This represents a scenario in which a new customer wishes an application that can be covered by the existing application but it has never been done before.

Implication In order to serve this customer it is necessary to create a new variation (this is indicated by the ellipse A.X). Once this variation is created it is possible to serve this customer. Investments for this are minimal since the basic idea of the PSP is promoting variations that can easily be generated (Section 3.2 Pg. 16).

Customer Y In this scenario a new customer wishes an application that can not be produced entirely from the existing application. Only parts of the desired application can be realized based on the existing application. A specific example might be that a customer requires a feature that is not part of the application yet.

Implication Based on the given application it is not possible to satisfy the customer. Thus, there are only three alternatives:

1. The customer is willing and able to change the requirements and adapt to the application that can be delivered.

2. The operator extends application A to implement variation A. Y can be created. This might be beneficial for the operator since the extension that was made for customer Y might also be of use to other paying customers.
3. It is not possible to satisfy the customer. Thus, it is not possible to acquire this customer.

What an operator does in this position has to be well-considered. First, it is necessary to analyze what exactly the extension is and what it would cost to implement. Second, the benefit of the extension needs to be considered according to the following questions. Is it possible to sell the extension to multiple customers as an add-on or is there a significant number of potential customers for whom this extension would be a “deal-winner”?

Customer Z In this scenario a new customer wishes an application that can not at all be covered by the existing application.

Implication Similar to the previous scenario. Two of the three alternatives are the same: the customer is not willing or able to adapt to a deliverable application, and it is not possible to satisfy the customer. Thus, it is not possible to sell the product. The third alternative, the extension of the application, is rather unlikely for this scenario since the variation that is required by the customer is completely outside of application A. This means that the entire application that allow to realize A.Z would have to be developed. This is probably not beneficial or profitable for the operator.

Using the four scenarios that have just been discussed above allows one to make a statement about customers. The most ideal customer for a PSP-Application is one who has a process in place or wishes to outsource one that can be delivered based on an existing PSP-Application variation. It would also be profitable if the needed PSP-Application can be generated without extension or investment in the application. Besides this it is also possible to extend an existing PSP-Application. But this should only be done if it is beneficial for the long term goals of the operator such as gaining further market penetration and thereby generating bigger revenues.

4.3 Pricing Strategy

This section will discuss the pricing strategy of PSP-Applications. Since from a customers' view these applications are SaaS-Applications, the considerations that are presented in the following apply to all kinds of applications that are proposed for delivery according to the SaaS service model.

In order to discuss the pricing strategy this section starts by talking about different levels of metering usage. This is followed by a comparison of the traditional licensing model and the SaaS one. The section concludes by discussing different commitment factors of SaaS that were not covered by the previous discussion.

4.3.1 Pricing SaaS

In section 2.1.2 (Pg. 6) the different characteristics of CC were introduced. One of the characteristics was that the actual usage of the provided service is metered. The customer is billed based on this metered usage. This method is called pay-as-you-use and was also mentioned in section 4.1.1.

Applying this to SaaS-Applications, or especially for this section, with PSP-Applications the metering can be done in different ways. Possible ways to charge the customer in the pay-as-you-use fashion could be per transaction, event, minute, click, operations or any other measurable attribute. When billing is based on such attributes it most closely follows the CC characteristic. However, it is a fact that most SaaS vendors charge a fixed subscription fee per user and specific period of time [Ret10, Wri07].

Salesforce.com for example charges between \$5 and \$250 per user and month [sal10b]. Customers that pay this monthly fee can use the application as much as they want.

One might argue that such a subscription fee could also be considered a pay-as-you-use model since the customer can sign up users based on the demand. The typical example is a company that signs up an intern for a limited period of a few month. The degree of flexibility mainly depends of the subscription period. One month, as used by salesforce.com, certainly gives customers flexibility. Google, on the other hand, charges customers an annual fee of \$50 per user for their business-grade version of Google Apps [Goo10a]. A subscription period of an entire year certainly shrinks the customer's flexibility.

However, offering an application based a subscription has advantages for the customer as well as for the operator. The customer has more cost control since the "pure" pay-as-you-use model is not necessarily cheaper. The operator, on the other hand, can plan the revenue streams with a higher certainty [Ret10].

Due to these considerations the pricing model for PSP-Applications will be based a subscription fee per user and month.

In addition, new customers will be charged an initial integration fee. This is necessary since PSP-Applications are highly customized to user's requirements. Please refer to section 3.5 (Pg. 30) for further details about how PSP-Application flexibility and how they are built. The exact price of this initial fee mainly depends on the effort that is necessary to build an application. If a customer, is willing to adapt to a standard version that application the fee might actually not be necessary. If the customers, however, require many changes and maybe even the integration of their own system, the fee might be higher. However, when determining the price it is necessary to find a reasonable price in order to maintain the advantage of low cost for the customer.

4.3.2 Subscription Pricing based on Perpetual License

When determining the subscription price for an SaaS product it is necessary to consider the competitors' prices [Wri07].

Doing this is rather simple if the competitors also offer SaaS-Applications. However, the purpose of the PSP it to deliver industry specific SaaS-Applications. So far most of the SaaS-Applications

target for horizontal processes, it is very likely that the market that a new PSP-Application aims to penetrate is dominated by applications following the common software model using a perpetual license.

Due to this fact, it is necessary to analyze the price a customer would have to pay to a competitor when determining the actual subscription price of a PSP-Application.

The two core elements a customer has to pay for a perpetual license are the perpetual license price itself and the associated annual maintenance [Wri07].

The easiest approach to determine the subscription price is by dividing the perpetual license and maintenance by the subscription term, generally in month, and probably applying a carrying cost or interest rate to account for the time value of money [Wri07].

However, this very simplistic approach is arguable since the subscription price, customers would have to pay becomes very high if they only subscribe for a short period of time. Furthermore, the price for the entire subscription term would be little different for the customer than buying the perpetual license and maintenance [Wri07].

Due to this fact it is obvious that the subscription term is not the appropriate divisor. Another approach is demonstrated by the following equation [Wri07]. Please review table 4.1 for further details about the parameters.

$$\begin{aligned}
 SaaS_{month} &= \frac{PL}{Payback} + \frac{PL \times \%Maintenance}{12} \times (1 - SaaS_{discount}) \\
 &= PL \times \frac{1}{Payback} + PL \times \frac{\%Maintenance}{12} \times (1 - SaaS_{discount}) \\
 &= PL \times \left(\frac{1}{Payback} + \frac{\%Maintenance}{12} \right) \times (1 - SaaS_{discount})
 \end{aligned}$$

The divisor the formula uses is the payback term. The payback or internal payback can be

Para.	Description
$SaaS_{month}$	Monthly subscription price per unit
PL	Perpetual license price per unit
$Payback$	Desired internal payback against perpetual license revenue
$\%Maintenance$	Annual relevant maintenance rate for perpetual license
$SaaS_{discount}$	Discount factor related to commitment to SaaS

Table 4.1: Description of parameters to calculate subscription price.

defined as “the number of months from contract onset that management is willing to risk until full value of an equivalent traditional sale would be received” [Wri07]. Thus, the equation captures the value of the perpetual license over a certain period of time and adds the value of monthly maintenance. In addition, the equation adds one more multiplicand that allows a discount to the customer. Discounts are not uncommon to perpetual licenses as well as for subscription based pricing. List prices can be calculated by setting the $SaaS_{discount} = 0$.

As mentioned earlier, the goal of the formula is to create a price that is attractive for both the vendor and the customer by recouping the perpetual license over a certain period of time.

Since the structure of the formula estimating the payback period has major impact on the

subscription price it also has impact on profitability of the entire product. Due to this importance it is necessary to spend further thoughts on calculating the payback period.

A very important thing that needs to be considered is that it is rather unlikely that every customer is going to be subscribed for the entire payback period. Actually, it is very likely that a certain percentage of all customers default after a certain time. Thus, it is necessary to decrease the payback based on the probability that a customer is going to unsubscribe and the estimated average usage time until they do so. The following equation captures this [Wri07].

$$Payback = term + renewal - \%defaulters \times (term + renewal - timeToDefault)$$

Table 4.2 describes the different parameters which are applied in this formula. Based on the two

Para.	Description
<i>Payback</i>	Desired internal payback against perpetual license revenue
<i>term</i>	Subscription term in month
<i>renewal</i>	Expected renewal month
<i>%defaulters</i>	Percentage of customers who preach or cancel the contract
<i>timeToDefault</i>	Expected average number of month to default for the group

Table 4.2: Description of parameters to calculate payback period.

equations that were presented in this section it is possible to determine a subscription price by analyzing competitors' perpetual licenses. However, when applying these equations some aspects of SaaS are not considered. These aspects are dealt with in the next section.

4.3.3 Other Factors for Subscription Pricing

Section 4.1.2 discussed the benefits customers gain from changing their IT strategy from buying perpetual licensed and running applications on their own infrastructure to SaaS [Leo09].

So far the benefits customers gain from SaaS, have not been considered when determining the price based on the competitors' prices. However, they need to be considered when designing subscription prices. If these potential benefits are actually used to increase the price for a subscription or not, mainly depends on the pricing strategy of the operator. If operators try to penetrate an existing market by using aggressive pricing as a tool, they might try to keep their prices below those of competitors [Wri07].

The price a customer pays for the product is an important factor if the entire product is going to be profitable for the operator. Profitability of PSP-Applications is will be discussed in the following section.

4.4 Mathematical Model

This section analyzes the profitability of SaaS-Applications or to be more precise: the profitability of PSP-Applications. Profitability can be defined as:

“The efficiency of a company or industry at generating earnings.” [mon10]

Profitability is analyzed by creating a mathematical model. Inputs for this model is everything that has been presented to this point. The purpose of everything that has been presented to this point was to setup the environment in which this model can be created.

A model can be defined as:

“Graphical, mathematical (symbolic), physical, or verbal representation or simplified version of a concept, phenomenon, relationship, structure, system, or an aspect of the real world. The objectives of a model include

- 1. to facilitate understanding by eliminating unnecessary components,*
- 2. to aid in decision making by simulating ‘what if’ scenarios,*
- 3. to explain, control, and predict events on the basis of past observations.*

Since most objects and phenomenon are very complicated (have numerous parts) and much too complex (parts have dense interconnections) to be comprehended in their entirety, a model contains only those features that are of primary importance to the model maker’s purpose. [...]” [Bus10]

The purpose of this section is to create a mathematical model which allows one to understand the impact of different inputs of a PSP-Application’s on profitability. Once the model is created it can be used for decision making. This will be done based on a specific example in the next chapter (section 5.1 Pg. 57).

However, in order to create the model this section starts by determining a measure for profitability. Starting from this point inputs are added to the model to expand the model the in a way that considers the most important factors.

In order to create this section Ms. Stefanie Kreusel was consulted as an expert on profitability analysis. She holds an MBA and is currently working for T-Systems International GmbH in the area of Corporate Performance Controlling. She also reviewed this section and checked for correctness of the used methods.

4.4.1 Measure for Profitability

The first step for creating a mathematical model which allows one to analyze profitability of PSP-Applications is to determine a measure for profitability.

If operators decide to offer a new PSP-Application, they have to invest in the development of this new application. Investment is the commitment of money or capital in order to gain profitable returns. The classic financial appraisal techniques are often considered the standard

tools to perform a profitability analysis. They deliver meaningful results based on a limited and understandable collection of equations and input parameters [Bru09]. Due to this fact they will be used to analyze profitability of PSP-Applications.

In general, financial appraisal techniques can be categorized into the following two different groups [Bru09].

Non-discounted Cash Flow Analysis are very applicable. However, this limits their correctness and meaningfulness to for reality.

Discounted Cash Flow Analysis are much more flexible and effective. Values that are used for the calculation are not considered in a time-independent single value manner but as a series of cash flows following the concepts of the time value of money.

Due to the complexity of PSP-Applications' profitability, the discounted cash flow analysis is the appropriate tool to do the analysis. As a matter of fact this kind of method is nowadays common whenever these kinds of calculations need to be performed [Bru09]. This is due to the fact that spreadsheet tools allow one to apply this rather complex calculations in an easy way by gaining higher reliability. Non-discounted methods should only be used in cases where approximated value is used as inputs for the method.

The decision to pick a discounted cash flow analysis method limits the number of appropriate methods [Bru09]. Since the Net Present Value (NPV) is the most common discounted cash flow analysis method which is used in many companies, this method is used as the primary measure for profitability in this thesis.

4.4.2 Net Present Value

As discussed in the previous section the Net Present Value (NPV) is used as primary measure for profitability in this thesis.

NPV compares the projected cash flows to the costs of an investment. It uses the time value of money to evaluate these future cash flows at a particular point in time. Time value of money means that money available today is worth more than the money available in the future. An example: if one owns €38.56 and invests it today at a rate of 10 percent, the investment will yield €100 after a period of 10 years. When investing the same amount (€38.56) in 10 years the lost profit is €61.45. This fact is considered in the NPV calculation [Bru09].

Applying this means that the value of a cash flow is determined by the time it appears. This is expressed by the present value (PV). The PV is the value on a given date of a future payment or series of future payments, discounted to reflect the time value of money.

$$PV_t = FCF_t (1 + WACC)^{-t} = \frac{FCF_t}{(1 + WACC)^t}$$

Based on this the NPV can be seen as the sum of PV over a certain period of time. Table 4.3

Para.	Description
NPV	Net present value
PV	Present value
T	Time of the cash flow (period under observation)
FCF_t	Free cash flow at time t
C_0	Initial investment (has to be a negative since cash outflow)
$WACC$	Weighted average cost of capital (Required rate of return)

Table 4.3: Parameters used for NPV calculation.

describes the variables used in the equation.

$$NPV = \sum_{t=0}^T PV_t = \sum_{t=0}^T \frac{FCF_t}{(1 + WACC)^t}$$

Once an operator decides to offer a new application to customers, an upfront investment needs to be done in order to finance the development. Thus, in year zero there is a negative free cash flow. Based on the introduced formula this can be expressed as follows:

$$PV_0 = \frac{C_0}{(1 + WACC)^0} = C_0$$

Thus, the final formula that will be used to calculate the NPV as measure for the profitability of PSP applications looks like this:

$$NPV = C_0 + \sum_{t=1}^T \frac{FCF_t}{(1 + WACC)^t}$$

Once the NPV is calculated for a particular project, the result has to be interpreted according to the following rules:

NPV > 0 The investment would add value to the company. Thus, it is profitable.

NPV < 0 The investment would subtract value from the firm. Thus, it is not profitable.

NPV = 0 The investment would neither gain nor lose value for the company. Thus, the proposed project repays original investment plus the required rate of return

Based on this it is possible to make a statement about the profitability of a particular application.

4.4.3 Weighted Average Cost of Capital

Most of the variables used for the NPV analysis are rather intuitive. In order to demonstrate their meaning they will be applied to an example in the next section. The weighted average cost of capital (WACC), however, needs further explanation.

As the name already implies WACC refers to the cost for the capital a company pays in order to

be able to invest a certain amount of money. The WACC is part of the NPV calculation since the cost of capital needs to be paid by the project in which the capital was invested. Thus, the WACC can also be seen as the minimum rate of return a project has to achieve.

In order to calculate the WACC it is necessary to analyze from where a company receives its money for an investment. Usually companies use a mixture of capital that is owned by themselves (own capital) and money they borrow from a third-party (outside capital). Both forms of capital are used based on a ratio, e.g. 30 % outside capital and 70 % own capital. Using this ratio it is possible to calculate the actual cost of capital (WACC) based on the mixture of different sources of capital [T-S10].

However, for this thesis it is assumed that the WACC is given by the company/operator who invests in the project. Thus, the acquisition of capital and ratio between own and outside capital is only covered hidden by the WACC.

4.4.4 Net Present Value - Example

Since the previous section was rather theoretical this section will apply the theory to an example in order to demonstrate the NPV. In order to do that the NPV is calculated in the following example: a project has an initial investment (C_0) of €50,000 (which includes machinery, and employee training costs). The Free Cash Flows of year 1-3 is €25,000 annually. The required rate of return (WACC) is 10%, and there are no cash flows expected after year 3.

Applying these inputs to the equation the following calculation is performed. The first step is to calculate the individual present values of each year.

$$\begin{aligned}
 PV_0 &= C_0 & &= -50,000 \\
 PV_1 &= \frac{25,000}{(1 + 0.10)^1} & &= 22,727.27 \\
 PV_2 &= \frac{25,000}{(1 + 0.10)^2} & &= 20,661.16 \\
 PV_3 &= \frac{25,000}{(1 + 0.10)^3} & &= 18,782.87
 \end{aligned}$$

As visible in the calculations the present values shrink as time passes. This is due to the time value of money. Money invested today is worth more than money invested in a year. Based on these individual present values it is possible to calculate the NPV. As explained in the previous section the NPV is the sum of all present values.

$$\begin{aligned}
 NPV &= \sum_{t=0}^3 PV_t \\
 &= -50,000 + 22,727.27 + 20,661.16 + 18,782.87 \\
 &= \underline{\underline{12,171.30}}
 \end{aligned}$$

As calculated the project will have a NPV of €12,171.30 after three years. Since the NPV is greater than zero ($NPV > 0$) the investment into the project would be profitable for the

company. In addition to the 10% required return the company would make an additional profit of more than twelve thousand Euros.

4.4.5 Free Cash Flow - Analysis

The major input that is necessary to determine the NPV is the Free Cash Flow (FCF). The FCF represents the cash a company generates after laying out the money required for tax. The money is available to all investors: dept holders and equity holders [KGW⁺05]. Free cash flow is important since it allows companies to make acquisitions, develop new products, pay dividends, make acquisitions, and reduce debt. This all contributes to the goal to enhance shareholder equity.

In order to determine the FCF the Entity-Approach is used. In this approach the FCF is calculated independently of the cost for different types of capital (equity and dept). Once the FCF has been calculated it will be discounted, while determining the NPV, by applying the WACC at this point the cost of capital will be considered [CHS09, T-S10].

The alternative to this used approach is the Equity-Approach. By applying this approach the cost of capital, both equity and dept, is considered while determining the FCF.

However, the Entity-Approach was taken since it is the less complex approach [T-S10, KGW⁺05]. This section discusses how the FCF can be determined based on the Entity-Approach in more detail. Figure 4.3 gives an overview of that method.

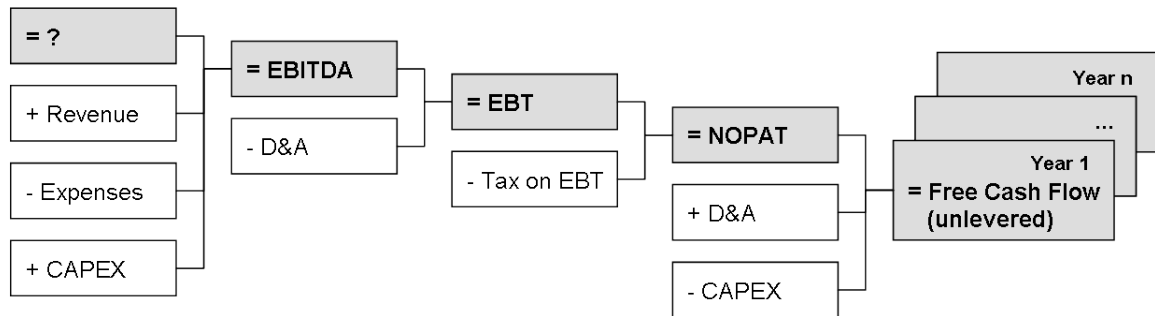


Figure 4.3: Overview of the entire Free Cash Flow Analysis.

Earnings before Interest, Tax, Depreciation, and Amortization (EBITDA)

As the name already implies EBITDA refers to the earnings of a company before Interest, Tax, and Depreciation and were accounted. It can be determined based on the following equation:

$$EBITDA_t = Revenue_t - Expenses_t + CAPEX_t$$

In order to determine EBITDA the expenses need to be subtracted from the generated revenue and CAPEX are added. This is due to the fact that CAPEX shall not be considered when determining Tax and Depreciation and Amortization. This is due to the fact that CAPEX are investments. Tax and Depreciation and Amortization, however, is only determined based on

operational cash flows [CHS09, T-S10]. Thus, CAPEX are added (since they are included in the Expenses).

How the revenue (section 4.4.6 Pg. 51), expenses, and CAPEX (section 4.4.7 Pg. 53) are determined is discussed in the following sections.

Earnings before Tax (EBT)

As implied by the name the EBT is a measure for the earnings before tax. This means that Deprecation and Amortization (D&A) are subtracted from the EBITDA.

$$EBT_t = EBITDA_t - D\&A_t$$

Straight-line depreciation is the simplest and most-often used method. When applying this method companies estimate the salvage value of the asset at the end of the period during which it will be used. This period is called useful life (UL). After the useful life period the asset has no value. For this model the D&A are calculated by the following equation.

$$D\&A_t = \begin{cases} \frac{\sum_{n=0}^t CAPEX_n}{UL} & \text{if } t < UL \\ \frac{\sum_{n=t-UL+1}^t CAPEX_n}{UL} & \text{if } t \geq UL \end{cases}$$

In this calculation the interest is not considered. This is due to the fact that the entity approach is used. The cost of capital is considered once the WACC is used to determine the NPV.

Net Operated Profit after Tax (NOPAT)

As the name already implies, NOPAT refers to the Net Operating Profit after Tax. It is determined by subtracting tax that has to be paid on profits from the EBT. Since there are different tax systems all over the world [KL05], only a very simple one is used for this method. This model simply states that a given percentage of profits (TaxRate) must be paid as tax.

$$NOPAT_t = \begin{cases} EBT_t & \text{if } EBT_t \leq 0 \\ EBT_t - EBT_t \times TaxRate & \text{if } EBT_t > 0 \end{cases}$$

Free Cash Flow

After all the steps that were just presented the Free Cash Flow can finally be determined by the following equation.

$$FCF_t = NOPAT_t + D\&A_t - CAPEX_t$$

D&A are added to the NOPAT again since this value is only used for the books but does not represent a cash flow [CHS09, T-S10]. CAPEX need to be subtracted again since they represent a cash flow expense that was only removed for the determination of tax and D&A [CHS09, T-S10]. In order to insert CAPEX into the calculation they need to be subtracted since they were originally part of the expenses.

Para.	Description
R	Total revenue
RNC	Revenue from new customers
RLC	Revenue from loyal customers
P	Market penetration
S	Market size
IIF	Initial installation fee
SP	Subscription price
SPT	Subscription price per tenant
AUT	Average number of users per tenant

Table 4.4: Parameters for Revenue calculation.

4.4.6 Revenue

Revenue is the income a company receives from customers in exchange for goods and/or services. As discussed in section 4.1.1 customers do not buy a perpetual license but rather rent a service in order to obtain use of a PSP-Application. Fundamentally, this changes the way in which a SaaS or PSP operator gains revenue fundamentally. As discussed in section 4.3 the incoming revenue streams a PSP operator gains come from the following two sources.

Subscription Fee This refers to the fee customers pays per user per month. This is the primary source of revenue for the operator.

Installation Fee This refers to the fee customers pay when they first start to use the application. This fee pays for the customization of the application to customer's needs, provisioning of the application, and creation of users.

Since the installation fee is only paid by new customers it is necessary to distinguish between revenue generated by new and loyal customers when calculating the total revenue. This can be shown by the following equation:

$$R_t = RLC_t + RNC_t$$

Table 4.4 gives an overview of the abbreviations used. Determining the revenue that is generated by loyal customers is determined by multiplying the number of customers of the past year by the cost of the subscription a customer (tenant) has to pay per year.

$$RLC_t = P_{t-1} \times S \times 12 \times SPT$$

$$SPT = AUT \times SP$$

As seen in the equation the number of customers is based on the market penetration in the previous year and the total market size. The subscription price per tenant (SPT) is calculated based on the subscription price per user (SP) multiplied by the average number of users (AUT) a customer (tenant) needs.

Determining the revenue that is generated by new customers is harder than those of loyal customers. Revenue that is generated by new customers comes from the installation fee and from

the subscription fee.

$$RNC_t = RevenueFromInstallationFee + RevenueFromSubscription$$

The first step for both sources of revenue is to determine the number of new customers. This can be done by subtracting this years' market penetration from the previous years and multiplying it with the total market size.

$$NumberOfNewCustomer = (P_t - P_{t-1}) \times S$$

Multiplying the number of new customers in the given year by the installation fee, results in the revenue that is generated through the installation fee.

$$\begin{aligned} RevenueFromInstallationFee &= NumberOfNewCustomer \times IIF \\ &= (P_t - P_{t-1}) \times S \times IIF \end{aligned}$$

Calculating the revenue that is gained from subscription can be determined based on the number of new customers multiplied by the subscription price a customer (tenant) has to pay for the subscription of the first year. The months the new customers have to pay in their first year can be estimated based on the assumption that the month in which new customers start using the application is evenly distributed over the entire year. If this is assumed, the average time a new customer is subscribed in the first year is 6 month.

$$\begin{aligned} RevenueFromSubscription &= NumberOfNewCustomer \times SPT \times 6 \\ &= (P_t - P_{t-1}) \times S \times SPT \times 6 \end{aligned}$$

Based on these two sources it is possible to determine the revenue that is achieved through a new customer.

$$\begin{aligned} RNC_t &= ((P_t - P_{t-1}) \times S \times SPT \times 6) + ((P_t - P_{t-1}) \times S \times IIF) \\ &= (P_t - P_{t-1}) \times S \times (6 \times SPT + IIF) \end{aligned}$$

Combining this with the revenue that is produced by loyal customers the total revenue can be determined.

$$\begin{aligned} R_t &= RLC_t + RNC_t \\ &= (P_{t-1} \times S \times 12 \times SPT) + ((P_t - P_{t-1}) \times S \times (6 \times SPT + IIF)) \\ &= S \times ((P_{t-1} \times 12 \times SPT) + ((P_t - P_{t-1}) \times (6 \times SPT + IIF))) \end{aligned}$$

Using this equation the revenue can be determined.

4.4.7 Expenses

Besides the revenue that can be achieved by offering a PSP-Application to customers, there are also expenses involved. Expenses are the outflow of money to another person or group to pay for an item or service. In case of a PSP-Application there is a wide variety of different expenses that need to be taken into account when analyzing profitability. The following equation gives an overview (table 4.5 explains the used abbreviations):

$$Exp_t = AO_t + DC_t + PC_t$$

Application Operation (AO) refers to the expenses that are necessary to execute and provide the application to customers. At this point the design of the last chapter has direct impact on cost. In section 3.6 it was stated that the PSP will be hosted in an Infrastructure-as-a-Service infrastructure because operators only pay the infrastructure they actually need to satisfy their customers.

Based on this it is possible to determine the total cost of operation based on the number of customers multiplied by the cost of operation per customer and number of years of operation. However, similar to the calculation of the revenue it is necessary to distinguish between the operation cost resulting from new customers and those costs resulting by loyal customers. This is due to the fact that loyal customer raise cost for operation for the entire year and new customers only for half a year (in average). Expressing all this as equations results in the following (table 4.5 explains the used abbreviations.):

$$\begin{aligned} AO_t &= AONC_t + AOLC_t \\ AOLC_t &= P_{t-1} \times S \times 12 \times OC \\ AONC_t &= (P_t - P_{t-1}) \times S \times 6 \times OC \end{aligned}$$

The cost for application operation can be calculated based on the complete equation:

$$AO_t = (P_{t-1} \times S \times 12 \times OC) + ((P_t - P_{t-1}) \times S \times 6 \times OC)$$

Development Cost (DC) refers to the expenses that are necessary to develop or continue developing the PSP-Application. It, thereby, includes the capital expenditures (CAPEX). The term CAPEX refers to expenditures for creating future benefits. This includes capital invested to acquire or upgrade physical assets. Physical assets might include property, industrial buildings or equipment. For the PSP this refers to the cost for acquisition, installation, licenses and training. Furthermore the cost of implementation and project management of a new application needs to be added. The following equation illustrates how development cost can be determined:

$$DC_t = CAPEX_t + Mgmt_t + Dev_t - Reuse_t$$

Para.	Description
<i>Exp</i>	Total expense of a given year
<i>AO</i>	Total Application operation cost
<i>AONC</i>	Application operation cost for new customers
<i>AOLC</i>	Application operation cost for loyal customers
<i>DC</i>	Development cost
<i>PC</i>	Promotional cost
<i>OC</i>	Operational cost per customer and month
<i>CAPEX</i>	Cost for acquisition, installation, licenses and training of the PSP.
<i>Mgmt</i>	Project management involved with developing a new PSP-Application.
<i>Dev</i>	Cost of development for a new PSP-Application.
<i>Reuse</i>	Cost savings due to reuse of already developed components of subsystems.
<i>P</i>	Market penetration
<i>S</i>	Market size

Table 4.5: Parameters for Expense calculation.

In the last chapter reuse was one of the main requirements for the PSP (section 3.1 Pg. 13). This is due to the fact that reuse of already existing components of sub-systems shortens investments in development times. Since the conceptual design of the PSP is an input for this chapter this needs also to be considered as well as determining the investment that is necessary to develop the new PSP-Application. Due to this fact, there is a variable called *Reuse* in the equation that is subtracted. This variable represents the money that is saved, due to reuse of already existing components.

Promotional Cost (PC) refers to other cost that are necessary for the promotion of a PSP-Application. These cover the costs for sales and marketing.

$$PC_t = Marketing_t + Sales_t$$

Marketing can be described as the actions that are necessary to create customer interest in products or services. Sales, on the other hand, refer to the activities that are involved with selling a particular product to a customer.

Both are necessary in order to bring a new product, in this case a PSP-Application, to the customer. Only if customers are aware of a certain product they will buy it.

Is it possible to create the equation for calculating expenses based on the cost for application operation, development, and promotion.

$$\begin{aligned}
 Exp_t &= AO_t + DC_t + PC_t \\
 &= ((P_{t-1} \times S \times 12 \times OC) + ((P_t - P_{t-1}) \times S \times 6 \times OC)) \\
 &\quad + (CAPEX_t + Mgmt_t + Dev_t - Reuse_t) \\
 &\quad + (Marketing_t + Sales_t)
 \end{aligned}$$

4.4.8 Initial Investment

This refers to the investments, the outflows of money, which are made before the application can be offered to the first customer. In fact the determination is the determination of the Free Cash Flow in point of time 0.

$$C_0 = FCF_0$$

However, since the initial investment is the investment an operators has to pay before they can start to introduce the product to the market the Free Cash Flow analysis can be simplified in many aspects.

Revenue Since the product can not be introduced to the market yet, there is no revenue.

Implication: $R_0 = 0$

Expenses Since at this point there is no market penetration yet it is not necessary to operate the application on a productive system. Furthermore, it is assumed that while the application is still in production there are not expenses necessary for promotion.

Implication: $Exp_0 = DC_0 = CAPEX_0 + Mgmt_0 + Dev_0 - Reuse_0$

Tax Since there is no revenue yet it is not possible to gain a profit. Since only on profits tax need to be payed there is no Tax to pay for the initial investment.

Implication: $NPAT_0 = EBIT_0$

Due to this simplifications the following alteration for the Free Cash Flow analysis can be used to determine the initial investment:

$$\begin{aligned} C_0 &= CAPEX_0 + Mgmt_0 + Dev_0 - Reuse_0 + CAPEX_0 - D\&A_0 + D\&A_0 - CAPEX_0 \\ &= CAPEX_0 + Mgmt_0 + Dev_0 - Reuse_0 \end{aligned}$$

4.5 Analysis of Mathematical Model

The goal of the method was just created in the previous sections as tool for planning whether or not a given scenario can be realized as a PSP-Application in a profitable way. Since in the last section the entire method was presented step by step with lots of explanations this section's purpose is to give a short overview of it and try to make a general statement about the efficacy of industry-specific SaaS-Application.

In order to gain an overview of the entire method please refer to table 4.6. The table lists all the different equations that were just presented. On the left side is the value that is to be determined. On the right side are the inputs that are necessary to determine the value. To all values on the left a color code was assigned. This color code was also assigned to the same variable on the right. This way only those variables are still of black font which do not appear in the left column of the table. Thus, they are inputs for the entire profitability analysis.

Based on this overview it is obvious that the model requires a total of 19 inputs in order to determine the profitability of a project for period of observation of one year. These inputs would

Value	Equation
Net Present Value	$\underline{NPV} = C_0 + \sum_{t=1}^T \frac{FCF_t}{(1+WACC)^t}$
Initial Investment	$\underline{C_0} = CAPEX_0 + Mgmt_0 + Dev_0 - Reuse_0$
Free Cash Flow	$FCF_t = CAPEX_t + NOPAT_t + D\&A_t$
Net Operated Profit after Tax	$NOPAT_t = \begin{cases} EBT_t & \text{if } EBT_t \leq 0 \\ EBT_t - EBT_t \times TaxRate & \text{if } EBT_t > 0 \end{cases}$
Earnings before Tax	$EBT_t = EBITDA_t - D\&A_t$
Depreciation and Amortization	$D\&A_t = \begin{cases} \frac{\sum_{n=0}^t CAPEX_n}{UL} & \text{if } t < UL \\ \frac{\sum_{n=t-UL+1}^t CAPEX_n}{UL} & \text{if } t \geq UL \end{cases}$
Earnings before Interest, Tax, Depreciation, and Amortization	$EBITDA_t = R_t - Exp_t + CAPEX_t$
Revenue	$R_t = (P_{t-1} \times S \times 12 \times SPT) + ((P_t - P_{t-1}) \times S \times (6 \times SPT + IIF))$
Expenses	$Exp_t = (P_{t-1} \times S \times 12 \times OC) + ((P_t - P_{t-1}) \times S \times 6 \times OC) + (CAPEX_t + Mgmt_t + Dev_t - Reuse_t) + (Marketing_t + Sales_t)$

Table 4.6: Overview of the entire mathematical model.

be: $T, WACC, TaxRate, S, AUT, SP, IIF, Mgmt_0, Mgmt_1, Dev_0, Dev_1, Reuse_0, Reuse_1, Marketing_1, Sales_1, CAPEX_0, CAPEX_1, P_1, OC_t$. However, analyzing profitability is usually done over multiple years since most projects don't expect to break even within the first year. Assuming the period of observation is set to 3 year that would even increase the total number of inputs.

Due to this very high number of inputs it is not possible to define a specific environment settings in which industry-specific SaaS-Applications can be profitable. This point will be discussed in more detail in section 5.2.2 (Pg. 72). It is, however, possible to make a very general statement about profitability of such applications. This statement is:

Industry-specific SaaS-Applications can be profitable if the application attracts enough customers to cover the investments and the annual cost the application causes.

This is certainly a very general statement, it is backed up by the structure of the mathematical model. If a specific application scenario can be profitable need to be determined for a specific case. In order to do that the proposed mathematical model can be used as a tool. It only need to be feed with the required inputs.

Discussion of Results

In the previous chapter the platform, that was proposed in chapter 3, was analyzed from a business perspective. In order to do analyze the platform in an object manner that a mathematical model was created which captures important inputs to estimate the profitability of PSP-Applications. This chapter's purpose is to discuss the results of the previous chapters. Since both, chapter 3 and 4 are rather theoretical, this chapter applies their results to a specific example. This is achieved through an example possible application. This application is explained and analyzed as to whether it could be realized as a PSP-Application in a profitable way. In order to do this the mathematical model from the previous section is used. In order to determine the cost for the initial investment that is necessary to build the application the architecture model of section 3.4 is used. After the example has been demonstrated, the entire approach of the thesis is summarized and discussed.

The chapter is divided into two parts. The first part applies the results of previous chapters to an example. The example is an application for the eHealth sector which covers the process of transferring a patient between different institutions. The second part of this chapter is analyzes the entire approach that was presented in this thesis. Furthermore, shortcomings of this approach are discussed.

5.1 Example: Patient Transition

This section's purpose is to apply everything that has been presented in this thesis so far to a specific example. In order to do that, this section starts by introducing the example and explaining why the application would be a relevant application for an operator. After that, the application is designed according to the architecture that was proposed in section 3.4 (Pg. 23). Based on the designed architecture, the cost for the development is estimated. The cost of development represents one of the inputs that are necessary to determine profitability. Based on

this input and others given by a market analysis the profitability of the example is estimated using the model that was created in section 4.4 (Pg. 45).

Please notice that the primary purpose of this section is to demonstrate the results of this thesis. Due to this fact it is not possible to conduct an entire market analysis. Weak assumptions are marked whenever it was necessary to make them furthermore they are discusses in section 5.1.10 (Pg. 70).

5.1.1 Introduction of the Example

The process that is covered by the example application is a process from the area of eHealth. eHealth refers to the healthcare practice supported by electronic processes and communication [Mea01]. The application targets for the German market. Due to this the application has a special focus of satisfying the requirement of this specific market.

The example process that this section deals with is the process that is executed when a patient has to transit from one healthcare institution to another. This transit can appear between different institutions:

Hospital This covers all kinds of hospitals (e.g. normal, rehabilitational, psychotherapy, or pure night or day hospitals).

Nursing home According to German law (SGB XI) three types of nursing homes can be distinguished: “Altenwohnheim” (the primary purpose is habitation; “home for the elderly”), “Altenheim” (housekeeping is done by the personnel), and “Pflegeheim” (stationary care 24/7 is required).

Out-patient facility This covers nursing services by private companies as well as public welfare.

Doctor’s office This includes all kinds of doctors.

Every time a patient is transferred between institutions data needs to be transferred as well. In Germany this is done by sending a discharge letter (called “Arztbrief”) from the sending entity to the institution that will perform the subsequent treatment. This letter covers the among other following data:

- Personal information about the patient - First and last name, date of birth, address etc.
- Duration of treatment.
- Medical diagnosis - possibly with International Statistical Classification of Diseases and Related Health Problems 10th Revision categorization (ICD-10).
- Treatments that were conducted.
- Medical findings.
- Medical history.

The letter addresses primarily the physician who is continuing the treatment of the patient. In addition, the letter might address additional physicians that are involved in the treatment and the patient himself.

5.1.2 Demographic Predictions

According to the Organization for Economic Co-operation and Development (OECD), Germany belongs to the OECD countries with the highest fraction of old people. According to prognoses conducted by the German government, this trend is going to continue in future. Elderly people require naturally more medical attention than younger people. Thus, the cost pressure is expected to raise in future. IT, however, can help to optimize processes in healthcare and, thereby, not just help to save money but also to improve quality [PDH10, T-S10].

These facts contribute to the importance of the area of eHealth in general and the presented example in particular.

5.1.3 Motivation

Currently, the mentioned letter is filled in by a doctor and sent by snail mail to the institution that receives the patient. An application as it is developed in this section would offer several benefits for the healthcare institutions [PDH10]:

- Improved interaction and communication between different institutions.
- Facilitate of communication between different institutions.
- Standardize of communication between different institutions.
- Save of time for the communication.
- Prevent of loss of data.
- Improve treatment of patients due to more detailed documentation.

For many years there were no standards for communication area of eHealth, neither for communication between different institutions or for communication within the same institution [PDH10, T-S10]. Introducing such standards is the goal of the Health Level Seven (HL7) non-profit organization. Furthermore, in the past there has not been a standardized version of a discharge letter. However, the Verband der Hersteller von IT-Lösungen für das Gesundheitswesen e.V. (VHitG)¹ has defined the VHitG-Arztbrief as a standardized version of such a document. The utilization of such standards in the development of the example application that is described in this section is highly recommended. Since experts predict that these standards will evolve to be common in the eHealth industry, it is obvious that implementing this with the example application will allow to communication with other systems in future [PDH10]. Thus, the Patient Transition application can have the ability to transfer data faster and directly from and into a

¹Might be translated as 'co-operational non-profit organization of Software vendors in the eHealth area'.

Hospital Management System (HMS). Due to a standardized form, data will also have a higher quality.

However, experts recommend building such an application with and without this communication with HMS in order to allow customers to start using the application as easy as possible [PDH10]. Customers can start with the version without networking their systems and once they become familiar with the application, the systems can be connected. This reduces the acceptance threshold of consumers. These different versions of the application will be used to show how variations can be handled with the approach that is presented in this thesis.

5.1.4 Creation of the Process Model

The goal of this section is to design the application's architecture using the method that is presented in section 3.4 (Pg. 23). By applying this method the application is designed in a service-oriented way. This allows for the ease adjustment of the application to customers' needs and enables the reuse of components on an inter-application level.

The following paragraphs illustrate how the method can be used to create the example application's components.

Step 1: Modeling Business Services

The primary purpose of this step is to analyze the given application from a business perspective. This is done by stepwise refining use cases starting with a business service. The refinement is done following the method's step 1 that is illustrated in figure 3.4 (Pg. 24). Applying this step

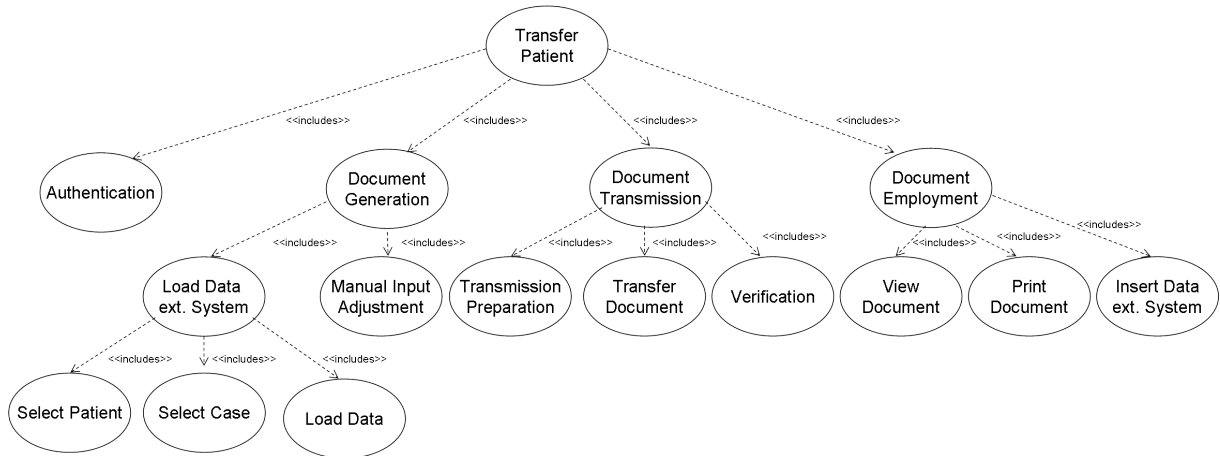


Figure 5.1: Illustrates the result of applying step 1 to the example.

to the example produces the tree shaped use case structure that is illustrated by figure 5.1. The top-level business service is *Transfer patient*. In the given example there is only one business service since the application serves only the purpose to transferring patients from one health care institution to another. This is the business service that customers can outsource by using this application. Another example, however, might require to fulfill two or more business services. The top-level use cases that need to be performed to fulfill the registration process are:

Authentication This top-level use case ensures that only authorized users can access the system.

Generation This top-level use case covers all steps that are required to generate the document in the institution that the patient is going to leave.

Transmission This top-level use case covers the use cases that need to be executed in order to transmit the document from the sending institution to the receiving institution.

Employment This top-level use case includes all steps that are required to receive the document and employ it within the institution that receives the patient.

Each individual top-level service contains a certain number of lower-level use cases. In the following the method these are defined iteratively.

Step 2: Design Domains

This second step deals with the creation of the sub-systems of the application. Figure 3.5 (Pg. 26) gives an overview of the entire method's step. The basic idea of this step is to create the sub-systems of the application based on the *use cases*, *business objects*, and *business variations*. For the given example these terms refer to the following:

Use cases The top-level use cases of the previous step become sub-systems. In the given example these are *Authentication*, *Document Generation*, *Document Transmission*, and *Document Employment*.

Business objects This refers to objects that are used in more than one subsystem. In the given example the transfer document is a business object that is used in the sub-systems *Generation*, *Transmission*, and *Employment*. Furthermore, the institution is another business object which associates the users' with other institutions. This object is used in the *Authentication* (to grant access) and the *Transmission* (to determine addressee) sub-system.

Business variations In the given example there is not business variation since the process is highly standardized and all steps are executed the same way by all customers.

The result of this step is illustrated by figure 5.2. As visible in this figure the results of this sub-

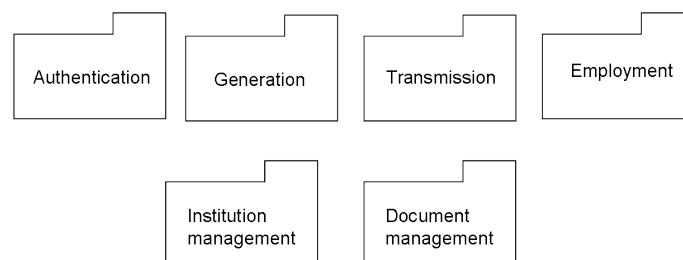


Figure 5.2: Illustrates the result of applying step 2 to the example.

system are the definition of four sub-systems *Authentication*, *Document Generation*, *Document Transmission*, and *Document Employment* which derive from the use cases. The *Document Manager* and *Institution Manager* sub-systems are created due to the business objects. The rule that results in this creation states that if a certain business object is used in more than one sub-system, it becomes its own sub-system.

Step 3: Design Components

This step's purpose is to design the individual components of the example application. This is done according to the method's step illustrated by figure 3.6 (Pg. 27). In order to create the components of the application the lower-level use cases are transformed. These components are categorized according to the defined categories. To conclude the step the components are assigned to a particular sub-system. The result of this step is illustrated by figure 5.3. As the

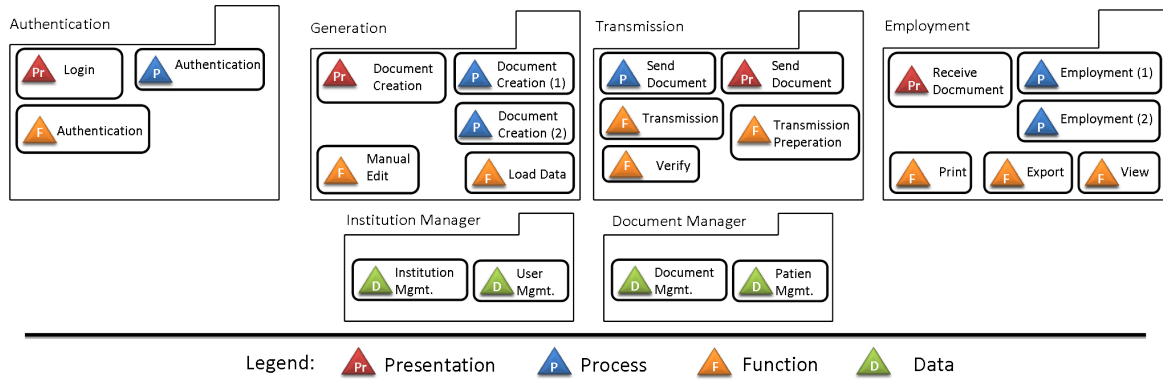


Figure 5.3: Illustrates the result of applying step 3 to the example.

figure illustrates there are no additional sub-systems added since it is not necessary to execute step 4 of this method's step. The purpose of each individual component is discussed in section 5.1.6 (Pg. 63). Furthermore, this section estimates the initial investment that is necessary to realize the application.

As illustrated by figure 5.3 there are two process components in the sub-systems *Generation* and *Employment*. This is due to the variation that is handled by the application and is discussed in the following section.

5.1.5 Flexibility

As mentioned in section 5.1.3 (Pg. 59) the application must cover a certain variation in the process. In order to have a low acceptance threshold, the application must be available with and without interaction to the institutions' own systems. To realize this in the sub-systems, each sub-system contains two process components. *Document Creation (1)* and *Employment (1)* describe the processes of creating and employing the document without interaction to an external system. This means that the functional components *Load Data* and *Export* are not used. The process components *Document Creation (2)* and *Employment (2)*, on the other hand, use the two functional components and, thereby, cover the process including the interaction with

an institution's own IT-Systems. In order to create the applications based on this the process

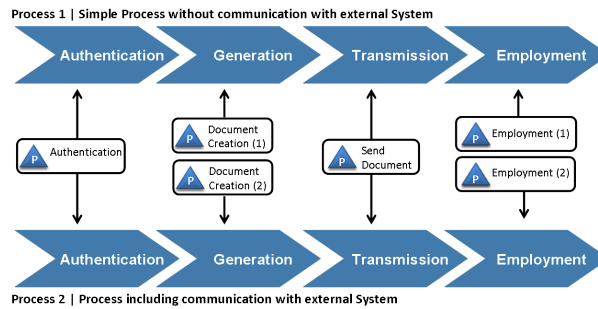


Figure 5.4: Illustrates how the process variation of the application can be realized.

components are orchestrated to a process. Depending on the process variation that is to be covered, this can be done by using one component or the other. This is illustrated by figure 5.4.

5.1.6 Determination of Initial Investment

The goal of this section is to determine the cost that is necessary to develop the example application. In order to do this an expert was consulted. This expert is Mr Marco Kuhnert. He holds a degree in Computer Science and is a project manager of T-Systems in the Systems Integration department.

Development Cost of subsystems

The following lists all the sub-systems and describes their functionality. Based on this a price estimate is estimated.

Authentication This sub-system encapsulates functionality that is necessary to give users that have permission access to the system.

Estimated investment: €2,000 - This price is based on the assumption that Lightweight Directory Access Protocol (LDAP) can be used as Commercial off-the-shelf (COTS) Software.

Generation This sub-system encapsulates the functionality that is necessary to generate the document. The first step is that the user shall have the ability to choose a patient and a specific case associated with this patient. The data that is necessary for this is gathered from the institutions' own IT-Systems by the application. Based on this selection, the basic document is created and if needed manually edited. In the variation case without access to the external system, there is no prepositioned form. Instead the user is required to fill in everything manually.

Estimated investment: €25,000

Transmission This sub-systems' purpose is to encapsulate the functionality that is necessary to transmit a specific document that was created in the previous sub-system. The process of transmission covers the following: an addressee needs to be chosen by sender and the transmission

needs to be triggered. Before this can actually be done, it is necessary that the patient agrees to the transmission of data. This is required by German law (BDSG). In order to do this the patient signs a paper form that is archived by the institution that will initiate the transmission. Once the transmission has been started the system logs the transmission to verify that it actually took place. To do this a time stamp is stored when the transmission was triggered. Another time stamp is stored as soon as the next sub-system starts using the document.

Estimated investment: €20,000

Employment This sub-system encapsulates the functionality that is necessary to view different documents that were received. The user shall be allowed to browse through these documents. Furthermore, the user shall have the capability of printing these documents if necessary. In addition, it shall also be possible to insert a specific document directly into the institution's own IT-Systems via HL-7 Communication protocols.

Estimated investment: €20,000

Document Manager This component covers the Data components for the document and the patient. These encapsulate the Create, Read, Update, and Delete (CRUD) operations for these two business objects. It is important that at this point the standard of the VHitG-Arztbrief is implemented as mentioned in section 5.1.3 (Pg. 59).

Estimated investment: €25,000

Institution Manager This component covers the Data components for institutions and their associated users. Similar to the *Document Manger* this sub-system also encapsulates the CRUD operation for these two business objects.

Estimated investment: €15,000

Determination of Initial Investment

The initial investment is defined as $C_0 = Mgmt_0 + Dev_0 - Reuse_0 + CAPEX_0$ in section 4.4.8 (Pg. 55). In this section the prices are assigned to the specific parameters of the equation that is used to estimate the initial investment. Based on the previous section and the estimates by Marco Kuhnert it is possible to do that as follows.

Cost of development Based on the previous section it is possible to assume the cost of development is €107,000. In addition, another €20,000 will be added for an administrative application that allows the operator to administrate the application. Furthermore, this amount is used to allow the users to customize their own instance of the application. Thus, the **cost of development is assumed as €127,000.**

Cost savings through reuse (Reuse) It is assumed that this is the first application an operator implements in the eHealth area. Thus, this input is **set to 0.**

Project Management The cost for **project management** is assumed as **€30,000**. This includes a cost for the creation of user manuals, project documentation, and the conduction of technology study before the project is started.

CAPEX The investments for acquisition and installation of the PSP as well as training etc. is assumed as **€10,000**. This is based on the expert's opinion.

5.1.7 Determination of other Inputs

The goal of this section is to determine what other inputs that are necessary to analyze profitability. These inputs are explained in section 4.4 (Pg. 45). This section explains the different groups of inputs in the following order: *Static inputs*, *Operational Cost*, *Market Analysis*, *Pricing*, *Promotional Cost*, and *Annual Development Cost*. Every group is briefly explained, background information is given and the individual inputs are assumed.

Static Inputs

This section deals with inputs that do not depend on the application that is analyzed but inputs that are given through the environment.

Period under Observation (T) Due to the rather small size of the application this project should have a positive payback within a few years. Thus, the profitability will be analyzed over a period of **3 years**.

Weighted Average Cost of Capital (WACC) This value is determined by the operator. It depends on the mix of capital and the cost related to the capital. In order to be realistic the WACC is assumed as **10%** [T-S10].

Tax Rate In order to be realistic the tax rate is **set to 19%** [T-S10].

Useful life of Assets (UL) CAPEX will depreciate over **3 years**.

Operational Cost

This section deals with the input that determines the cost of operation.

Operational Cost (OC) This input refers to the cost of operation that is caused by one tenant and month. Unfortunately, there is no data available to determine this value. Thus, an it is **assumed as €20**.

Market Analysis

Figure 5.1 gives an overview of the market. According to experts in the field it is possible to reach a market penetration of 90%. The timeframe for this, however, is rather uncertain. Experts consider a market penetration of 50% within 5 years as realistic [PDH10].

Type of Institution	Number
Hospitals	2,083
Rehabilitational Institutions	1,239
Nursing Homes	11,029
Doctors' Offices	125,754
Home-care Companies	1,227
Nursing service	11,529
Hospice	158
Psychotherapy Offices	32,623
Sum	185,642

Table 5.1: Description of Market [T-S10, PDH10].

Market Size (S) Based on table 5.1 the total market size can be **assumed as 185,642 institutions**.

Market Penetration (P) Based on the available data the following market penetrations are assumed: $P_1 = 3\%$, $P_2 = 10\%$, $P_3 = 20\%$.

Pricing

Pricing is a very hard topic for the given application. In section 4.3 (Pg. 41) a benchmarking technique is presented that allows one to determine a subscription price which is based on the perpetual license price of competitors. However, in this market there are currently no competitors. Thus, it is hard to determine an appropriate price [PDH10]. One way is to analyze the cost the current process. The paper-based version requires institutions to extract data from an HMS, fill in a form, send this form, receive it, and input data into another system. Based on this a transaction-based pricing strategy would probably another acceptable alternative. It is estimated that in Germany there is an annual amount of five million documents sent [PDH10]. An acceptable price for one of the transactions would be €5 [PDH10].

Average number of users per Customer(AUT) This input is **assumed 2.3**. The number is assumed quite low because of the high ratio of doctors' offices as illustrated by table 5.1 (Pg. 66). These offices usually need only one or two users. Hospitals, on the other hand, have a much higher need for users of the application. One would assume that this type of system would first be adopted by the heavy users as they have the most to gain by its use. This would also support a transaction based pricing strategy to produce a better market penetration. However, since the input can only be assumed and it is not possible to back it up with data the input value will be analyzed further in section 5.1.9 (Pg. 68).

Subscription Price (SP) Since the model cannot cover such a pricing strategy a subscription price must be estimated. Assuming that a user of the system (which might represent an entire doctor's office) performs about five transfers of a patient a week. This makes twenty a month and results in a subscription fee of €100. However, since there is no data available about the average transfers per user and month, the subscription price is **assumed as €50 a month and user**.

Initial installation fee (IIF) This fee should cover the creation of users and some sales cost. This includes a introduction to the system for the personal that has to work with the system. Thus, the IIF is **assumed as €250**. However, since this value is only based on assumptions it is analyzed in section 5.1.9 (Pg. 68).

Promotional Cost

This refers to costs that have to be paid annually but which are not covered by any of the other sections.

Marketing Cost (Marketing) It is assumed that in the first year 0.5 full time employee (FTE) is needed for marketing tasks like the creation of advertising documents. In the second year only 0.25 are needed and in year three there are no marketing tasks are needed. Assuming that a Marketing FTE costs a company €150,000 this results in the following cost: $Marketing_1 = \text{€75,000}$, $Marketing_2 = \text{€37,500}$, and $Marketing_3 = \text{€0}$.

Sales Cost (Sales) For sales related tasks it is assumed that in the first and second year 1 FTE is required. In the third year only 0.5 FTE is required. Similar to the cost of a Marketing FTE the annual cost is assumed €150,000. This results in the following cost: $Sales_1 = \text{€150,000}$, $Sales_2 = \text{€150,000}$, and $Sales_3 = \text{€75,000}$.

Annual Development Cost

This refers to the annual cost related to the continuing product improvement and maintenance.

CAPEX It is assumed that CAPEX appears as followed: $CAPEX_1 = \text{€10,000}$, $CAPEX_2 = \text{€8,000}$, and $CAPEX_3 = \text{€8,000}$. This includes Licensing, Training, Modeling tools, etc.

Mgmt The annual cost for project management is assumed as €1,000 and appears as followed: $Mgmt_1 = \text{€1,000}$, $Mgmt_2 = \text{€1,000}$, and $Mgmt_3 = \text{€1,000}$.

Dev The annual cost for further development and maintenance is assumed as €10,000. Thus, this results in the follwoing cost: $Dev_1 = \text{€10,000}$, $Dev_2 = \text{€10,000}$, and $Dev_3 = \text{€10,000}$.

Reuse No reuse from other PSP-Applications is assumed, since is the first application for an operator in the eHealth area. Therefore, this results in: $Reuse_1 = \text{€0}$, $Reuse_2 = \text{€0}$, and $Reuse_3 = \text{€0}$.

5.1.8 Determination of Profitability

Based on the inputs that where estimated in the previous section it is possible to determine the profitability of the application. Using the mathematical model that was proposed in 4.4 (Pg. 45) the profitability of the application was analyzed. The result can be seen in table 5.2. As visible in the table the investment that is necessary to develop the application is €167,000. This was determined in section 5.1.6 (Pg. 63). However, the given example is able to pay back this investment in the first year already. Furthermore it make almost profit of almost €3 Million. In

Year	NPV
0	€-167,000.00
1	€2,999,371.96
2	€14,231,340.69
3	€36,305,202.89

Table 5.2: Profitability of Example Application.

year two and three the profit is even higher.

In order to illustrate how the NPV evolves from year 0 to 3 please review figure 5.5.

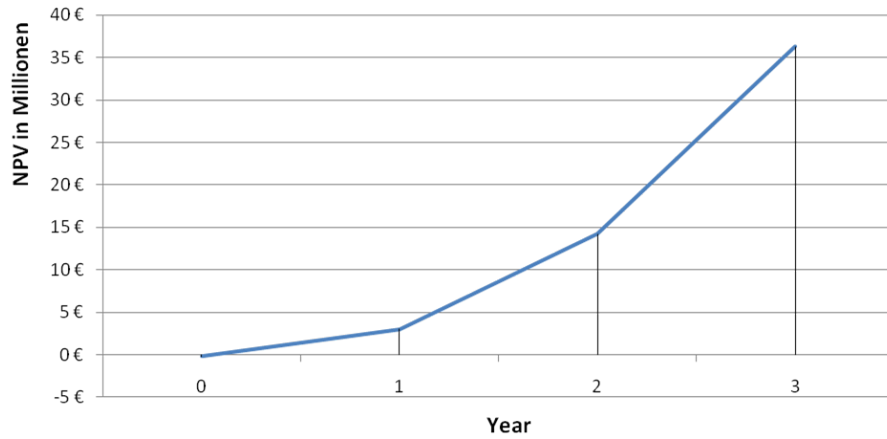


Figure 5.5: Illustrates the NPV of the example application over 3 year.

5.1.9 Sensitivity analysis

In the previous section the proposed mathematical model is used to determine the profitability. However, the model can also be used to perform a sensitivity analysis (SA). SA refers to a study of how the variation of inputs of a mathematical model affects the variation in the output of the model. In other words, it is possible to determine how the profitability of a given scenario changes if inputs are altered [Sal08]. This technique is especially interesting for input values that are estimated based on weak data.

However, when performing an SA based on the mathematical model of this thesis it is important to keep in mind that some of the input values are highly correlative. For example, a simple increase of the subscription price will improve the profitability. In reality, however, it is very likely that an increase of the subscription price will also result in a lower market penetration. Thus, it might be necessary to adjust not just one value but a set of values.

However, in order to analyze the given example of this section and, thereby, demonstrate the technique of SA this section continues by giving three examples.

Initial Investment

The first example examines which effect the initial investment of the application has on the profitability. In order to do that the mathematical model is fed with different values for the

initial investment. The values are €50,000; €100,000; ... ; €400,000. The result of this SA is illustrated by figure 5.6. The figure shows a three dimensional coordinate system which axes are

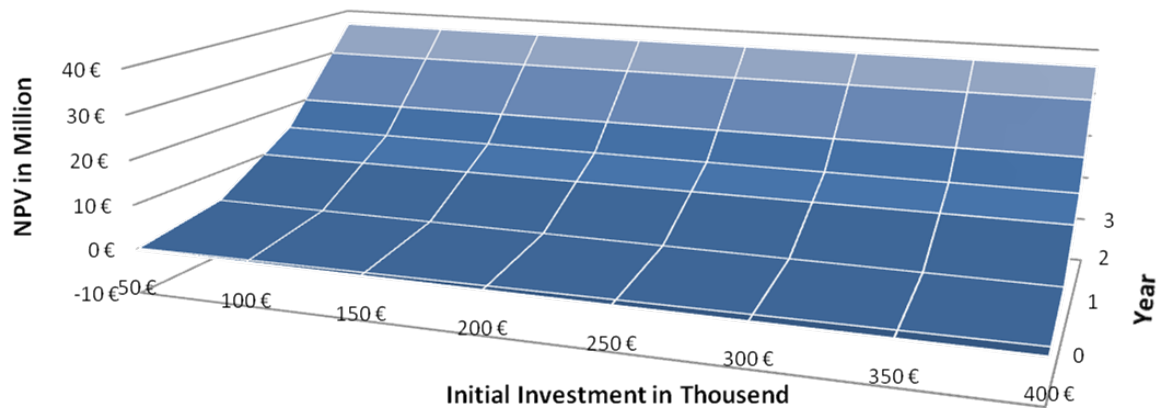


Figure 5.6: Illustrates the impact of initial investment on NPV.

the Initial investment, time, and the profitability. The profitability is illustrated as a plain in this coordinate system. As visible in the figure a higher initial investment increases the time until the application is profitable. However, this increase is only minor.

Pricing: Initial Installation Fee

The second example's purpose is to analyze the effect of the initial installation fee (IIF) customers have to pay when first subscribing as a customer. Since it is only little information available to determine how the application can be priced this is an interesting question. Especially since it might be possible that a more detailed market analysis reveals that an IIF might prevent the application from succeeding in the given market. Figure 5.7 illustrates the result of the SA. As

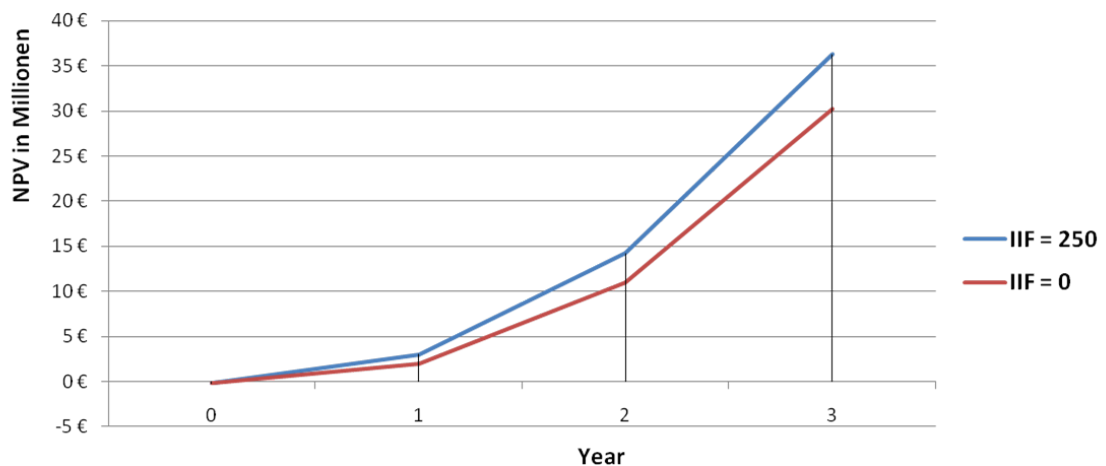


Figure 5.7: Illustrates the impact off the initial installation fee on the NPV.

illustrated by the figure the profitability decreases if the IIF is set to zero. This was expected but it is also possible to determine the decrease in profit due to this change. The profit that is achieved after a period of three years including the IIF was 36,290,202.89. This profit decreases

to a little more than €30 Million (€30,265,799.2) if the IIF is not charged. This is a decrease of 17%.

Average Number of Users per Tenant

The third and final example of a SA refers to the average number of users per tenant (AUT). In section 5.1.7 (Pg. 65) this input value is assumed as 2.3. This value, however, has only very weak backup through data. Because of this figure 5.8 illustrates a worst case scenario in which every customer only pays for one user. As visible this has huge impact on the profitability but

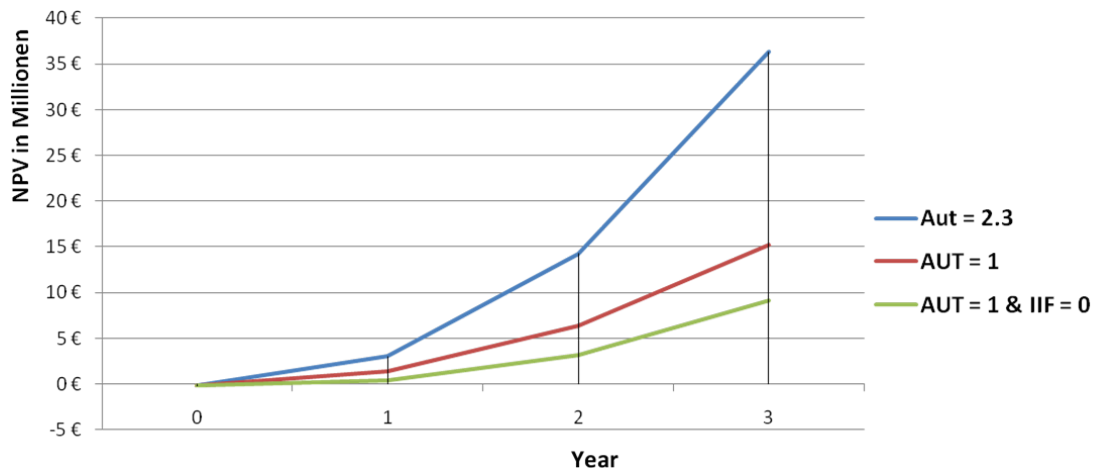


Figure 5.8: Illustrates the impact of the average number of customers on the NPV.

is still positive. This profit decreases to a little more than €15 Million (€15,187,083.64) if the AUT is set to 1. This, however, is a very unrealistic scenario because it would imply that also every bigger institution like a hospital would only pay for a single user. Furthermore, the figure illustrates the scenario in which AUT is set to 1 and the initial installation fee is set to 0. Even then the given application is still profitable (NPV after year 3 is €9,162,679.94).

5.1.10 Shortcomings of the Example

Due to the fact that the primary purpose of this section is to apply the results of this thesis, some shortcomings have to be accepted. These are listed in the following structure:

Simple example The given example application is simple from a software development point of view. However, it was picked because of the ideal fit for the SaaS-Model and is an extremely good example for a vertical industry application. A more complex example (from a software development point of view) was used for the creation of the paper that is presented in chapter B (Pg. 85).

Variation This example left little opportunity to present the models ability of the approach to handle variations of a process. However, in a more complex example this could have been done more extensively. As an example, appendix B (Pg. 85) presents another type of variation.

Availability of data The data available is quite substantial. Especially, according to the interview with an expert in the area of eHealth [PDH10]. However, the following two points need to be mentioned:

Pricing Strategy For the given example a transaction-based pricing strategy might have been more appropriate. To determine if this is really, a complete marketing research study would be needed and this is out of scope of this thesis.

Marketing Strategy In order to penetrate the eHealth market it would be necessary to create a full-blown marketing strategy. This strategy would have to use all the available techniques and tools that marketing has to offer. The development of such a strategy is beyond the scope of this thesis, thus, the values were estimated.

Privacy/Security The specific privacy and security concerns of an eHealth-System were not considered while creating the example. This is due to the fact that the section only presents a rough overview of the application. In case this application would be developed, a more detailed analysis of the application's security and privacy requirements would need to be performed.

Market Penetration Another shortcoming is that probably not every institution in Germany will start using the application at the same time. However, as the application was designed it is necessary that both institutions (sending and receiving institution) are using the application. This is rather unrealistic. A solution to this problem would be that the operator prints the document and sends it via snail mail to the receiving institution that is not yet a customer. However, due to complexity this was not considered in this thesis.

5.2 Analysis of Approach

This section's purpose is to analyze the entire approach of this thesis. In order to do this the approach is briefly summarized and shortcomings are addressed and discussed. This analysis is structured according to the presented chapters.

5.2.1 Chapter 3: The Delivery Platform

In chapter 3 (Pg. 13) a platform is designed whose purpose is to allow the financially beneficial development, deployment, and delivery of industry specific SaaS-Applications.

Shortcoming of IaaS Foundation The first shortcoming of this chapter is that it states that resources could be used more economically by simply deploying the proposed platform in an Infrastructure-as-a-Service (IaaS) environment. This, however, is a very simplistic approach. This approach only works as long as customers' usage of the application does not exceed the number of transaction the platform can handle. Another approach would be to dynamically distribute the application over several VMs in an IaaS environment according to demand.

However, the simplistic approach was selected for presentation to narrow the scope of this thesis.

Shortcoming of Reference Architecture Part of this chapter is the creation of a method that can be used to design applications in a service-oriented way. This method is created based on a proven method to create service-oriented application landscapes. However, this method has not been applied yet in a real project since it was created in this research thesis. So far it has only been applied to two example applications. Thus, it has not been proven that this method actually works well in reality. The two examples, however, provide evidence that the method would work. Developing an entire application using the method would exceed the scope of this thesis.

Shortcoming of Application Customization The approach that is presented in this thesis to create highly flexible SaaS-Applications by means of using a service-oriented application architecture. Due to this approach applications can be created by changing how the different service-oriented components are orchestrated. This was stated in the chapter as a very simplistic approach. It is not analyzed under which circumstances this change of composition can actually be performed. Furthermore, it would be possible to apply techniques from the area of software product line engineering to the approach of this thesis. However, all of this would exceed the scope of this thesis.

5.2.2 Chapter 4: Economic Evaluation

In chapter 4 (Pg. 35) the designed platform is analyzed from a business perspective. The result of this chapter is a mathematical model that captures the most significant inputs that are necessary to determine the profitability of an industry-specific SaaS-Application.

Shortcomings of Model The first shortcoming that should be mentioned is that the model does not cover customers that quit their subscription. This has the effect that it can handle only scenarios which have a rising or constant market penetration. This problem is due to how the revenue is calculated. In this equation new customers pay an initial installation fee and a subscription fee (in average for 6 months) in their first year. If the market penetration is falling, this means that there is a negative number of new customers which results in the effect that the initial installation fee is paid back to the customers when they leave. This does not comply with reality if customers leave, in that they would not get their installation fee back.

This was done due to complexity reasons. If quitting customers were to be considered by the model, it would require estimating the number of quitting customers, which adds more complexity to the model. Since it is assumed that the model would only be applied to scenarios where there seems to be a promising market which allows penetration, this shortcoming was accepted.

An additional shortcoming of the model is that it only provides the ability to cover one pricing model. The model that is covered allows one to charge the customer an initial installation fee and a subscription fee. This subscription fee is paid per month by the user of the system. However, as discussed in section 4.3 (Pg. 41) there are other pricing strategies possible. Some of them might even be more suitable for different kinds of applications. However, in order to make the profitability model reasonable simple the pricing strategies are limited to only the one

that is described in this thesis. This model was taken since it is the most common one for SaaS-Applications [Ret10, Wri07].

Another, more general shortcoming is that in order to analyze the profitability of a future project it is necessary to estimate some of the input values. This might especially be a problem when there is only weak data available about the market which the application aims to penetrate. But even if there is sufficient data available, estimations such as the market penetration can never be made with a guarantee since events might occur that are unforeseeable. However, this is a common problem to every area where estimates of the future are made. The only way to increase the certainty of estimates is by gathering as much data as possible.

Shortcomings of Model Analysis At the end of this chapter the mathematical model is analyzed. Due to the complexity of the model, and its 21 required inputs (in case that the period of observation is one years) it is not possible to make a specific statement under which specific circumstances industry-specific SaaS-Applications can be profitable. The original plan for the analysis of the model was to visualize the model by plotting it in a multi-dimensional coordinate system. This would allow one to analyze the entire model in a simple drawing. Based on this figure it would be possible to make specific statements under which specific circumstances industry-specific SaaS-Applications can be profitable. However, visualizing the model that way would require the use of as many dimensions as there are input variables. This would require a total number of dimensions that is higher or equal 21. It is not possible to visualize more than three (in some cases four) dimensions at once [PDF10]. One of them would be the result of the model of NPV. Thus only two inputs are left. The only way to cover more inputs with this would be to assume values until only two inputs are left, draw it and make another diagram with 2 other values. However, due to the complexity of the model this approach would end up with a very high number of diagrams, which in the end have no meaning because they are too confusing. Thus, it was not possible to follow the planned approach. Another approach would be to abstract the model to a point where it is possible to visualize it. If this approach had been chosen, the model would have lost its meaning for the specific area it deals with. If the only inputs that are left would have been e.g. the initial investment, revenue, and annual expenses the model would not describe the area anymore but pretty much any product that is possible. Another approach that would be possible is to make specific statements based on data. This data could be generated by determining the possible minimum and maximum of every input and feeding the model with each possible combination. This way lots of data would be created probably covering every possible case. In the end those cases would have a positive NPV (would be profitable) where enough revenue can be generated to cover all expenses. However, it would not be possible to make more specific statements about when it can be profitable because, even if there is a big market and it is possible to gain a high penetration, the subscription price might be too low to cover all expenses. Thus, it appears that there are far too many inputs to consider to make a more specific statement than the one that was made in section 4.5 (Pg. 55).

Summary and Conclusion

The purpose of this chapter is to provide a conclusion of the thesis. In order to do this it will start by summarizing the entire thesis and point out some of the highlights. Based on this summary the thesis will conclude by pointing out the results that were gained in this thesis. The final section of this thesis lists some loose ends that might serve as a starting point for future research.

6.1 Summary

The goal of this thesis was to analyze the efficacy of Cloud Computing applications in vertical industries. In other words, the purpose was to analyze under which circumstances industry-specific SaaS-Applications can be financially beneficial for an operator.

In order to answer this question the thesis started by providing a platform that allows for the development, deployment, and delivery of industry specific applications as economically as possible. This platform combines the paradigm of Service-Oriented Computing with the paradigm of Cloud Computing. The approach that was presented creates applications based on service-oriented building blocks. In order to create these building blocks, a architecture method was presented in section 3.4 (Pg. 23). This method was adapted from a method for creating application landscapes in a service-oriented way¹. Figure 3.3 (Pg. 23) gives an overview of the entire method. The building blocks that were created using this method can be combined in order to create applications that actually serve customers needs. These composite applications provide very high flexibility in that they can be adjusted to customers' needs. Furthermore it is possible to reuse individual components on an inter-application level. This allows operators so decrease

¹It was attempted to publish this method in an ACM Conference. The paper that was submitted can be found in appendix B (Pg. 85). This first attempted was not successful. The reasoning for this can be found in appendix C (Pg. 92). However, the paper was considered interesting and worth publishing because of that a future attempt is planned.

development cost, thereby decreasing time to market. In order to optimize the operational cost of the platform and the applications that are delivered to the customer using it, the platform was built based on a Cloud Computing infrastructure using a technique called multi-tenancy. Both allow to more efficient use of available resources. This results in cheaper operational cost for applications. The goal is cost related improvement so as to make serving smaller markets more economically beneficial.

Based on this platform the thesis continued by analyzing industry-specific applications from a more business-oriented view. In order to do this a more detailed discussion of the business model of Software-as-a-Service was provided. At this point the specific focus was set to the implications of CC-Applications to, both customers' and operators' business. Based on this discussion customer-types were identified who would be most attractive to applications delivered as SaaS-Applications on the proposed platform. Furthermore, a pricing strategy was presented. This strategy is proposed as a benchmarking technique with a goal of determining a subscription price based on the prices of competitors' perpetual licenses. All this was used as input for the creation of a mathematical model. This model captures all necessary inputs to determine the profitability of a industry-specific application. By applying this model it is possible to estimate the profitability of a given application. An overview of the model can be gained by reviewing the table 4.6 (Pg. 56).

In order to illustrate the use of this model a specific example was given. This example represents an application from the area of eHealth. The application was created by applying the aforementioned architecture method. Based on the resulting architecture it was possible to estimate the cost of development. Using this and other inputs from market analysis it was possible to estimate the application's profitability by utilizing the aforementioned mathematical model.

6.2 Conclusion

The main result of this thesis are the conceptual design of a platform and a mathematical model:

Platform As presented in chapter 3, by designing the platform it was possible to create an environment that promotes the development, deployment, and delivery of industry-specific applications in a financially beneficial way. This goal was achieved by combining the paradigm of Cloud Computing with the paradigm of Service-oriented Architecture. This is illustrated by figure 6.1.

Mathematical model Building on the platform in chapter 3, the creation of a mathematical model was described. Table 6.1 gives an overview of the entire model. Based on this model it was possible to make a very general statement about the profitability of industry-specific SaaS-Application. This statement is that *industry-specific SaaS-Applications can be profitable if the application attracts enough customers to cover the investments and the annual cost the application causes*. In addition to this rather intuitive statement, the mathematical model can serve as a tool to evaluate whether a given industry specific application can be profitably developed and deliver or not.

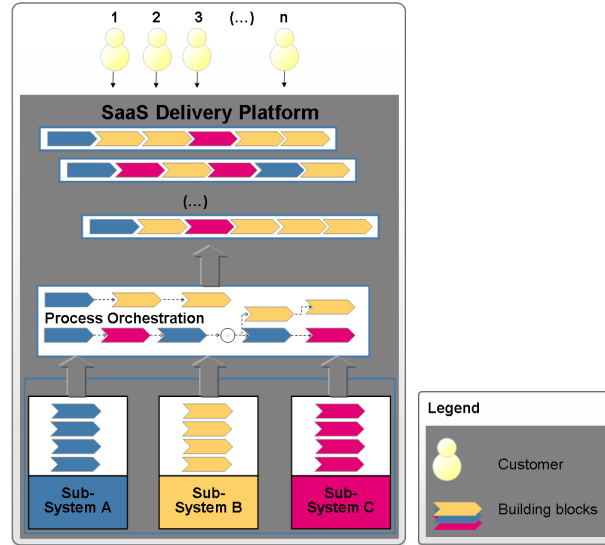


Figure 6.1: PSP Applications can be changed through Orchestration.

In order to demonstrate the application of both results the thesis, the thesis continues by analyzing an example in chapter 5 (Pg. 57). The example dealt with an application from the area of eHealth that covers the process of transferring a patient from one health care institution to another. First of all the application was designed in a service-oriented way using the approach presented in chapter 3 using this architecture the development cost was estimated. Using this estimate and other assumptions that were determined based on a market research the mathematical model was applied. It was possible to determine that according to the model, the application would be profitable.

As a conclusion for this thesis it can be said that it appears like industry-specific SaaS-Application can be profitable under the condition that the achievable market share generates enough profit to cover all expenses. Using the example application it was possible to provide evidence that indicates that such cases actually exist. However, cases that appear promising need to be evaluated e.g. using the proposed mathematical model.

6.3 Future Research

There are numerous possibilities to continue the work of this thesis. Those possibilities can be split into two fields of research: the area of composite SaaS-Applications and the analysis of profitability of such applications.

6.3.1 Composite SaaS-Applications

This topic is discussed in chapter 3. However, it is only presented in a simplistic way that assumes that customers' usage of the application does not exceed the number of transactions the used platform can handle. An approach that might be analyzed in a following research might be to build the application as a distributed platform. Such a platform might allocate needed

Value	Equation
Net Present Value	$\underline{NPV} = C_0 + \sum_{t=1}^T \frac{FCF_t}{(1+WACC)^t}$
Initial Investment	$\underline{C_0} = CAPEX_0 + Mgmt_0 + Dev_0 - Reuse_0$
Free Cash Flow	$FCF_t = CAPEX_t + NOPAT_t + D\&A_t$
Net Operated Profit after Tax	$NOPAT_t = \begin{cases} EBT_t & \text{if } EBT_t \leq 0 \\ EBT_t - EBT_t \times TaxRate & \text{if } EBT_t > 0 \end{cases}$
Earnings before Tax	$EBT_t = EBITDA_t - D\&A_t$
Depreciation and Amortization	$D\&A_t = \begin{cases} \frac{\sum_{n=0}^t CAPEX_n}{UL} & \text{if } t < UL \\ \frac{\sum_{n=t-UL+1}^t CAPEX_n}{UL} & \text{if } t \geq UL \end{cases}$
Earnings before Interest, Tax, Depreciation, and Amortization	$EBITDA_t = R_t - Exp_t + CAPEX_t$
Revenue	$R_t = (P_{t-1} \times S \times 12 \times SPT) + ((P_t - P_{t-1}) \times S \times (6 \times SPT + IIF))$
Expenses	$Exp_t = (P_{t-1} \times S \times 12 \times OC) + ((P_t - P_{t-1}) \times S \times 6 \times OC) + (CAPEX_t + Mgmt_t + Dev_t - Reuse_t) + (Marketing_t + Sales_t)$

Table 6.1: Overview of the entire mathematical model.

resources according to the demand of individual building blocks. Such a platform might have configurable privacy settings that allow to adjust the individual components to the needs of customers. Instead of running all components in a single VM it would be possible to run a specific component for a specific customer in a private cloud environment. But for a different customer it would be possible to run the same component in a public or community cloud setting. Such a flexible platform would significantly increase the complexity of billing the customer. Another opportunity for further research in this field is given by the approach of how applications are composed. In this thesis it is only stated that individual building blocks can be composed to create customer specific applications. However, to enhance this approach it would be possible to apply techniques from the area of software product line engineering.

6.3.2 Profitability Analysis

This topic is discussed in chapter 4 (Pg. 35). The approach that is presented created a mathematical model that allows to analyze profitability based on estimated inputs. Due to complexity this model only covers one pricing strategy. Besides the presented subscription-based strategy the model might be enhanced with, for example, transaction-based strategy. Based on such an enhanced model it would be possible to compare different pricing strategies and analyze their profitability. Furthermore, it might be possible to use this model to simulate the impact of a given application on the market. Such a simulation might use defined customer groups as inputs and would allow not just to analyze profitability but also the usage of IT-Resources.

Bibliography

- [AFG⁺09] M. Armbrust, A. Fox, R. Griffith, A. D Joseph, R. H Katz, A. Konwinski, G. Lee, D. A Patterson, A. Rabkin, I. Stoica, et al. Above the clouds: A berkeley view of cloud computing. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28*, 2009. (Cited on pages 5 and 8.)
- [Ama10] Amazon. Amazon elastic compute cloud (Amazon EC2). <http://aws.amazon.com/de/ec2/>, April 2010. (Cited on page 8.)
- [BC95] V. R Basili and G. Caldiera. Improve software quality by reusing knowledge and experience. *Sloan Management Review*, 37:55–55, 1995. (Cited on page 15.)
- [Bru09] Ralph Brugger. *Der IT Business Case: Kosten erfassen und analysieren - Nutzen erkennen und quantifizieren - Wirtschaftlichkeit nachweisen und realisieren*. Springer, Berlin, 2., korr. u. erw. aufl. edition, March 2009. (Cited on page 46.)
- [Bus10] BusinessDictionary.com. model definition. <http://www.businessdictionary.com/definition/model.html>, August 2010. (Cited on page 45.)
- [BYV⁺09] R. Buyya, C. S Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599–616, 2009. (Cited on pages 1, 5, 6, 14 and 21.)
- [CC06a] Gianpaolo Carraro and Fred Chong. Software as a service (SaaS): an enterprise perspective. <http://msdn.microsoft.com/en-us/library/aa905332.aspx>, October 2006. (Cited on pages 35, 36, 37, 38 and 95.)
- [CC06b] Frederick Chong and Gianpaolo Carraro. Architecture strategies for catching the long tail. <http://msdn.microsoft.com/en-us/library/aa479069.aspx>, April 2006. (Cited on pages 2, 14, 21, 22, 32, 38 and 95.)
- [CHS09] Adolf G. Coenenberg, Axel Haller, and Wolfgang Schultze. *Jahresabschluss und Jahresabschlussanalyse: Betriebswirtschaftliche, handelsrechtliche, steuerrechtliche und internationale Grundsätze - HGB, IFRS, US-GAAP, DRS*. Schäffer-Poeschel, 21., überarbeitete auflage. edition, August 2009. (Cited on pages 49 and 50.)
- [CPJ05] Dave Crane, Eric Pascarello, and Darren James. *Ajax in Action*. Manning Publications, 1 edition, October 2005. (Cited on pages 15 and 17.)

- [CVC03] Kostas Chalikias, George Valiris, and Panagiotis Chytas. The role of IT in business process-oriented organisations. *Operational Research*, 3(2):137–154, 2003. (Cited on page 13.)
- [Die06] Jan Dietz. The deep structure of business processes. *Communications ACM*, 49(5):58–64, 2006. (Cited on page 27.)
- [EHH⁺08a] Gregor Engels, Andreas Hess, Bernhard Humm, Oliver Juwig, Marc Lohmann, and Jan-Peter Richter. *Quasar Enterprise: Anwendungslandschaften serviceorientiert gestalten*. Dpunkt Verlag, 1 edition, February 2008. (Cited on pages 16, 17, 23, 24, 26 and 30.)
- [EHH⁺08b] Gregor Engels, Andreas Hess, Bernhard Humm, Oliver Juwig, Marc Lohmann, Jan-Peter Richter, Markus Voß, and Johannes Willkomm. A method for engineering a true service-oriented architecture. In *To appear: Proceedings of the 10th International Conference on Enterprise Information Systems. Barcelona, Spain*, August 2008. (Cited on pages 23, 24, 25, 26, 27, 28, 29 and 95.)
- [GMKM04] N. Gold, A. Mohan, C. Knight, and M. Munro. Understanding service-oriented software. *IEEE Software*, 21(2):71–77, 2004. (Cited on page 12.)
- [Gol08] Bernard Golden. Who’s getting ROI from cloud computing now, October 2008. (Cited on page 11.)
- [Gol09] Bernard Golden. Capex vs. opex: Most people miss the point about cloud economics, March 2009. (Cited on page 10.)
- [Goo10a] Google. Business online messaging and collaboration applications - google apps premier editions. <http://www.google.com/apps/intl/en/business/features.html>, June 2010. (Cited on page 42.)
- [Goo10b] Google. Google app engine - google code. <http://code.google.com/intl/en-NE/appengine/>, April 2010. (Cited on page 8.)
- [Goo10c] Google. Google trends: cloud computing. <http://google.de/trends?q=cloud+computing>, September 2010. (Cited on pages 5 and 95.)
- [Goo10d] Google. Google trends: Service oriented architecture, service -oriented computing. <http://google.de/trends?q=Service+oriented+architecture%2C+service+-oriented+computing&ctab=0&geo=all&date=all&sort=0>, September 2010. (Cited on pages 11 and 95.)
- [Gro09a] The Open Group. SOA source book - detailed building blocks of the SOA reference arch. <http://www.opengroup.org/projects/soa-book/page.tpl?CALLER=page.tpl&ggid=1335>, 2009. (Cited on pages 17 and 19.)
- [Gro09b] The Open Group. SOA source book - infrastructure for SOA. <http://www.opengroup.org/projects/soa-book/page.tpl?CALLER=page.tpl&ggid=1336>, 2009. (Cited on pages 16, 17, 18 and 20.)
- [Gro10] Open Group. The open group architecture forum. <http://www.opengroup.org/architecture/>, May 2010. (Cited on page 23.)
- [GS06] J. Gebauer and F. Schober. Information system flexibility and the cost efficiency of business processes. *Journal of the Association for Information Systems*, 7(3):122–147, 2006. (Cited on pages 30 and 31.)

-
- [GSH⁺07] Chang Jie Guo, Wei Sun, Ying Huang, Zhi Hu Wang, and Bo Gao. A framework for native Multi-Tenancy application development and management. In *The 9th IEEE International Conference on E-Commerce Technology and The 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (CEC-EEE 2007)*, pages 551–558, Tokyo, Japan, 2007. (Cited on pages 2 and 32.)
 - [HMH96] O. Hanseth, E. Monteiro, and M. Hatling. Developing information infrastructure: The tension between standardization and flexibility. *Science, Technology & Human Values*, 21(4):407, 1996. (Cited on page 30.)
 - [How10] Dennis Howlett. SAP business ByDesign limps into general availability, May 2010. (Cited on page 22.)
 - [HS05] Michael N. Huhns and Munindar P. Singh. Service-Oriented computing: Key concepts and principles. *IEEE Internet Computing*, 9(1):75–81, 2005. (Cited on pages 11 and 30.)
 - [KBK10] Tobias Kretschmer, Kerstin Bolliger, and Clemens Koob. Life 2 - working connected in business an society. Technical Report 2, September 2010. (Cited on page 1.)
 - [KGW⁺05] Tim Koller, Marc Goedhart, David Wessels, Thomas E. Copeland, McKinsey, and Company. *Valuation: measuring and managing the value of companies*. John Wiley and Sons, 2005. (Cited on page 49.)
 - [KL05] Lutz Kruschwitz and Andreas Loeffler. *Discounted Cash Flow: A Theory of the Valuation of Firms*. John Wiley & Sons, 1. auflage edition, October 2005. (Cited on page 50.)
 - [Kle69] L. Kleinrock. UCLA to be first station in nationwide computer network. *UCLA, Office of Public Information*, 3, 1969. (Cited on page 5.)
 - [KP10] Werner Kurzlechner and Thomas Pelkmann. Das Gartner-Orakel: die 10 strategischen IT-Trends 2010. <http://www.cio.de/strategien/methoden/2212689/>, January 2010. (Cited on page 1.)
 - [Leo09] Lydia Leong. McKinsey on cloud computing. http://blogs.gartner.com/lydia_leong/2009/04/16/mckinsey-on-cloud-computing/, April 2009. (Cited on pages 37 and 44.)
 - [Mea01] Vincenzo Della Mea. What is e-Health (2): The death of telemedicine? *Journal of Medical Internet Research*, 3(2):e22, 2001. (Cited on page 58.)
 - [MG09] P. Mell and T. Grance. The NIST definition of cloud computing. national institute of standards and technology. *Information Technology Laboratory, Version*, 15:10–7, 2009. (Cited on pages 7, 8, 9 and 16.)
 - [MMLP09] Ralph Mietzner, Andreas Metzger, Frank Leymann, and Klaus Pohl. Variability modeling to support customization and deployment of multi-tenant-aware software as a service applications. In *Principles of Engineering Service Oriented Systems, ICSE Workshop on*, volume 0, pages 18–25, Los Alamitos, CA, USA, 2009. IEEE Computer Society. (Cited on pages 2, 32 and 33.)
 - [mon10] moneychimp. Profitability - definition. <http://www.moneychimp.com/glossary/profitability.htm>, September 2010. (Cited on page 45.)
 - [Nor79] George Norman. *Economies of scale, transport costs, and location*. Springer, 1979. (Cited on page 39.)
 - [oD10] Department of Defence. DoDAF architecture framework version 2.0. <http://cio-nii.defense.gov/sites/dodaf20/index.html>, May 2010. (Cited on page 23.)

-
- [PDF10] Andreas Prof. Dr. Fischer. Visualization of mathematical model, July 2010. (Cited on page 73.)
 - [PDH10] Ursula Prof. Dr. Hübner. eHealth example, August 2010. (Cited on pages 59, 60, 65, 66, 71 and 96.)
 - [Pel03] C. Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, 2003. (Cited on pages 30 and 32.)
 - [PF09] Christy Pettey and Carina Forsling. Gartner highlights five attributes of cloud computing. <http://www.gartner.com/it/page.jsp?id=1035013>, June 2009. (Cited on pages 6, 7, 14, 15 and 16.)
 - [PTDL07] Michael P. Papazoglou, Paolo Traverso, Shahram Dustdar, and Frank Leymann. Service-Oriented computing: State of the art and research challenges. *Computer*, 40(11):38–45, 2007. (Cited on pages 12, 16, 17, 30 and 32.)
 - [Pub09] Van Haren Publishing. *SOA Source Book: How to Use Service-Oriented Architecture Effectively - The Open Group Series*. Van Haren Publishing, May 2009. (Cited on page 17.)
 - [Ret10] Marin Reti. Pay as you use ist tot!, March 2010. (Cited on pages 42 and 73.)
 - [RML10] Stefan Ried, Pascal Matzke, and Mirosław Lisserman. Case study: Siemens IT solutions and services successfully positions a community cloud. Technical report, Forrester Research Inc., January 2010. (Cited on pages 4 and 9.)
 - [Ros04] Mendel Rosenblum. The reincarnation of virtual machines. *Queue*, 2(5):34–40, 2004. (Cited on page 6.)
 - [Sal08] Andrea Saltelli. *Global sensitivity analysis: the primer*. Wiley-Interscience, February 2008. (Cited on page 68.)
 - [sal10a] salesforce. CRM - salesforce.com. <http://www.salesforce.com/>, April 2010. (Cited on pages 8 and 32.)
 - [sal10b] salesforce.com. Salesforce pricing & editions. <http://www.salesforce.com/crm/editions-pricing.jsp>, June 2010. (Cited on page 42.)
 - [SH05] Munindar P. Singh and Michael N. Huhns. *Service-Oriented Computing: Semantics, Processes, Agents*. John Wiley & Sons, first edition, January 2005. (Cited on pages 16 and 30.)
 - [SJN10] Lutz Schubert, Keith Jeffery, and Burkhard Neidecker-Lutz. The future of cloud computing - opportunities for european cloud computing beyond 2010. page 71. European Commission - Information Society and Media, January 2010. (Cited on pages 7, 9 and 10.)
 - [SK10] I. Sriram and A. Khajeh-Hosseini. Research agenda in cloud technologies. *Arxiv preprint arXiv:1001.3259*, 2010. (Cited on pages 7, 8, 9 and 95.)
 - [Som09] Brian Sommer. SAP business ByDesign update: Multi-tenancy, In-Core memory DB and more, December 2009. (Cited on page 22.)
 - [Son08] Ameeta Soni. Delivering on the promise of SaaS. *Software Business Executive Report*, July 2008. (Cited on page 38.)
 - [SS90] Andrea Krasa Sethi and Suresh Pal Sethi. Flexibility in manufacturing: A survey. *International Journal of Flexible Manufacturing Systems*, 2(4):289–328, July 1990. (Cited on pages 30 and 31.)

- [SS07] Sajjan G. Shiva and Lubna Abou Shala. Software reuse: Research and practice. In *Information Technology, 2007. ITNG '07. Fourth International Conference on*, pages 603–609, 2007. (Cited on page 15.)
- [Szy02] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Professional, 2 edition, November 2002. (Cited on pages 1, 13 and 15.)
- [T-S10] T-Systems. T-Systems internal documents, August 2010. (Cited on pages 48, 49, 50, 59, 65, 66 and 96.)
- [Tur10] TurboTax. TurboTax tax preparation software. <http://turbotax.intuit.com/>, September 2010. (Cited on page 9.)
- [VRCL09] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, 2009. (Cited on pages 7 and 8.)
- [Wit08] Heiderose Witte. Erste erfolge auch jenseits von CRM: SaaS holt im ERP-Umfeld auf. <http://www.computerwoche.de/management/it-services/1862071/index.html>, May 2008. (Cited on page 1.)
- [Wri07] Drew Wright. Benchmarking software as a service (SaaS) subscription pricing against traditional software pricing: A rational approach to conversion. June 2007. (Cited on pages 42, 43, 44 and 73.)
- [Zac99] J. A Zachman. A framework for information systems architecture. *IBM systems journal*, 38(2/3):454–470, 1999. (Cited on page 23.)
- [Zin07] Marcus Zinn. Service based software construction process. In *SEIN 2007: Proceedings of the Third Collaborative Research Symposium on Security, E-Learning, Internet and Networking*, pages 168 – 184. 2007. (Cited on page 28.)

Definitions of Cloud Computing

“Cloud Computing offers the prospect of dramatically increasing your computing power, being able to balance workloads with demand and paying only for the services you use. Here is how a few of the leaders in the Cloud Computing marketplace define Cloud.”

Spinning Clouds

“A pay-per-use model for enabling convenient, on-demand network access to a shared pool of configurable and reliable computing resources (e.g., networks, servers, storage, applications, services) that can be rapidly provisioned and released with minimal consumer management effort or service provider interaction.”

NIST

“A style of computing in which scalable and elastic, IT-enabled capabilities are provided "as a service" across the Internet to multiple external customers.”

Gartner

“An emerging IT deployment, development and delivery model enabling real time delivery of products, services and solutions over the Internet.”

IDC

“Cloud Computing 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. The services themselves have long been referred to as Software as a Service (SaaS), so we use that term. The Data Center hardware and software is what we call a Cloud.”

University of California - Berkeley

“The key characteristics of the cloud are the ability to scale and provision computing power dynamically in a cost efficient way and the ability of the consumer (end user, organization or IT staff) to make the most of that power without having to manage the underlying complexity of the technology. The cloud architecture itself can be private (hosted within an organization’s firewall) or public (hosted on the Internet).”

Cloud Manifesto

APPENDIX B

QUASOSS-Paper: “A Method for Engineering an Architecture for
Software-as-a-Service Applications”

A Method for Engineering an Architecture for Software-as-a-Service Applications

Stefan T. Ruehl
Hochschule Darmstadt
University of Applied Sciences
Haardtring 100
64295 Darmstadt, Germany
stefan.t.ruehl@stud.h-da.de

Marcus Zinn
University of Plymouth
Drake Circus
Plymouth, Devon PL4 8AA, UK
marcus.zinn@plymouth.ac.uk

Stephan Verclas
T-Systems International GmbH
Heinrich-Hertz-Str. 1
64295 Darmstadt, Germany
stephan.verclas@t-systems.com

Bernhard Humm
Hochschule Darmstadt
University of Applied Sciences
Haardtring 100
64295 Darmstadt, Germany
b.humm@fbi.h-da.de

ABSTRACT

Cloud Computing and its service model with high service value, Software-as-a-Service (SaaS) are currently widely discussed topics. Both promise a change of paradigm in how IT is managed today.

Currently, platforms for SaaS applications are emerging, aiming to help to implement service-oriented systems on cloud infrastructures. These platforms allow applications to be created based on already existing building blocks. These composite SaaS applications offer a new kind of flexibility since adapting them to customers' demand only requires changing how they are composed. Furthermore, these building blocks offer the potential for being reused on an inter-application level. This shortens development times and initial investments for new applications.

However, the degree in which this vision can be realized is highly dependent on the way the building blocks are designed.

This paper presents a method to define these building blocks according to the actual business service which describe the application. The approach that is described follows design principles for service-oriented application landscapes.

Categories and Subject Descriptors

D.2.11 [Software Engineering]: Software Architectures

General Terms

Design, Management, Theory

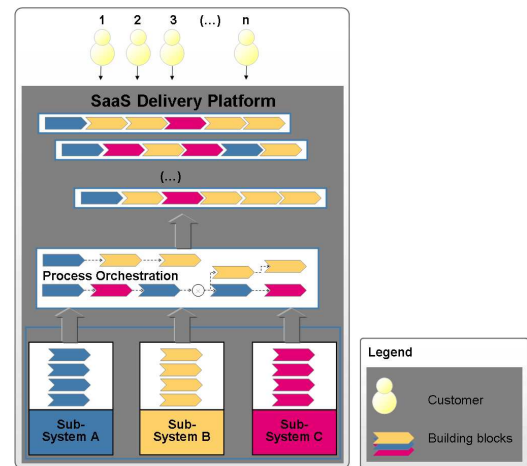


Figure 1: SOA-based SaaS application building.

Keywords

Cloud Computing, Software-as-a-Service (SaaS), Software Architecture, Service-oriented Architecture (SOA)

1. INTRODUCTION

The Service-oriented Architecture (SOA) has been around for a couple of years now and the first enthusiasm has settled. SOA is not hype anymore but an architecture style that can be beneficial for many companies [8]. Currently, Cloud Computing and its service model with high service value, Software-as-a-Service is the most discussed hype [8]. SaaS is a delivery model for software which is based on the fundamental idea of leasing software for a fee instead of buying licenses. Thus, operators run the application on their own infrastructure and grant customers access to the application for a subscription fee [1]. In order to exploit economies of scale, SaaS providers must be successful in attracting a significant number of tenants for their SaaS

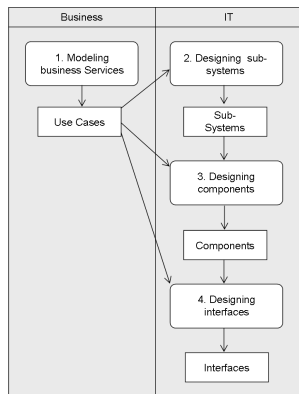


Figure 2: Overview of the overall method (adapted from [5]).

applications [9]. Naturally, the requirements that need to be fulfilled by a system differ for different customers. To satisfy these requirements a SaaS application needs to be flexible and adaptable to the needs of a specific customer. Probably the most well-known example for SaaS applications is Salesforce.com, a Customer relationship management (CRM) System. Users can customize Salesforce for their needs by adding custom user-defined fields to their customers [11]. This way they have the ability to store the data of their customers they actually need.

In order to reach a higher level of flexibility a SOA is well-suited as an architectural style for realizing SaaS applications [9, 2, 7]. This is due to the fact that this architecture offers a flexible way for building composite applications out of existing building blocks. Figure 1 illustrates this. This paper proposes a method to design and structure these buildings blocks.

The paper is structured as follows. In section 2 we present the method and apply it to a simple example. Section 3 demonstrates how the developed example can be used to produce applications for two customers that have different processes needs. Section 4 concludes the paper.

2. SERVICE-ORIENTED ARCHITECTURE METHOD

This section introduces a method that can be used to design service-oriented SaaS applications.

It was adopted from the paper “A Method for engineering a true service-oriented architecture” [5] and the book “Quasar Enterprise: Anwendungslandschaften serviceorientiert gestalten” [4]. This method, as presented by the two sources, discusses an approach to design an enterprise IT architecture based on a “true” Service-Oriented Architecture (SOA) [5]. This approach can be seen as an instantiation of several enterprise architecture frameworks like *The Open Group Architecture Framework (TOGAF)* [6], the *Integrated Architecture Framework (IAF)*, the *Zachmann Framework* [12], or the *Department of Defense Architecture Framework (DoDAF)* [10]. These frameworks provide terminology like meta models and abstract road maps. They do not provide concrete methods. This is where instantiations like the one presented can be used.

Based on this instantiation we developed a model to cre-

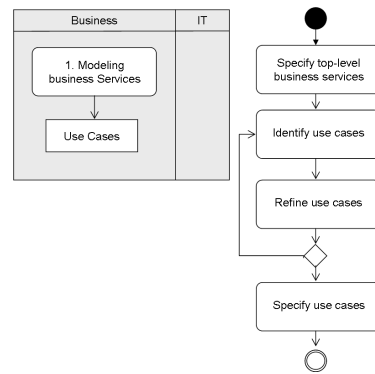


Figure 3: Overview of method step 1: modeling use cases (adapted from [5]).

ate the architecture of SaaS applications. This was done by adapting the method from the enterprise IT landscape level to the level of applications. Based on such an application architecture it is possible to create applications that can be adjusted to customers’ demand very easily by composing applications from defined components.

Figure 2 illustrates an overview of the overall method. This overview figure indicates the different steps of the method broken down into method chunks. Each individual chunk will be discussed and explained in the following section.

One of the most important things about SOA and the mentioned frameworks is that they separate *Business* and *IT* aspects and concerns [4]. This is indicated by the two swimlanes in figure 2. In order to illustrate the method further it will be applied to a specific example. The example that is used in this paper is an application that covers the registration process of a kindergarten.

2.1 Step 1: Modeling Use Cases

Figure 3 shows an overview of this step. The focus of this step is to stepwise refine Unified Modeling Language (UML) use cases that the application shall provide starting from the business services. “A business service is the output of a service provider towards a service consumer: goods, information, or activities. A business service is based on a service contract. It specifies in- and outgoing information and goods and the basic course of action.” [5].

Applying this definition to the application level as it is discussed in this paper can be done as described in the following: a given application’s purpose is to cover one or more business services of a company. Since this paper discusses SaaS applications, the business services that are covered are those the customer wishes to outsource. Every individual business service that is covered is realized based on a series of actions. Due to the facts that this paper discusses the architecture of applications and UML use cases are the proven tools to analyze applications they will be used for this step. Use cases can be defined on different levels of granularity. The usual form, a hierarchy, is that coarse-grained services contain fine-grained services. Thus, the following method step will create a hierarchy of use cases starting from the top-level business services. Each business services for itself includes a number of use cases which need to be performed. These use cases contain each for itself again a number of

lower-level use cases and so forth. This structure can be displayed by utilizing a UML use case “includes”-hierarchy. Since this step is located completely on the business swim lane the business services are not influenced by IT concerns. A possible input for this step is the business services the customer wishes to outsource and the requirement documents describing the functionality of how the business service needs to be fulfilled. At this point, however, it is important to keep in mind that the application is supposed to serve multiple customers. This means that it might be possible that different customers have different requirements. This is where the flexibility of the application needs to be able to cover these possibilities.

The goal of this method is to create the mentioned use case hierarchy by following a defined (partially integrative) process (fig. 3).

1. *Specify top-level business services:* This first sub-step’s focus is to specify the most coarse-grained functionality the application shall provide. These business services are the ones a customer wishes to outsource.
2. *Identify use cases:* Use cases that need to be performed within a business service are candidates for the next finer-grained level of use cases.
3. *Refine business service:* A use case is refined if the following condition holds.
 - (a) The entire purpose the coarse-grained of use case is covered by the finer-grained use case (additional fine-grained use cases need to be added).

Sub-steps 2 and 3 are executed until either a suitable level is reached or elementary use cases are found.

4. *Specify business services:* A suitable way would be to apply use case diagrams and use case descriptions as they are defined in the UML.

The result of this method step can be illustrated in a tree-shaped figure showing every top-level business service on the first level and the included use case it consists of on the next. Therefore, this graph might have many levels. Each business service and use case should be documented by using UML use case diagrams.

Applying this step to the example produces the tree shaped business service structure that is illustrated by figure 4. The top-level business service is *Kindergarten registration*. In the given example there is only one business service since the application serves only the purpose to sign up children to kindergartens. This is the business service customers can outsource by using the application.

The top-level use cases that need to be performed to fulfill the registration process are:

Application The main points of this use case are the actions that are necessary to apply for a free spot in a kindergarten: acquisition children information, selection of preferred kindergartens, and plausibility checks. The parents can apply for a spot either directly using a parent portal or paper-based by filling-in a form. The filled-in form needs to be inserted into the system via

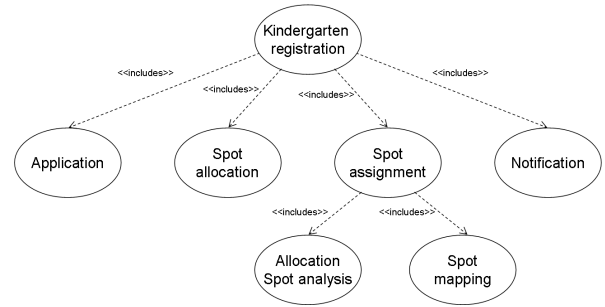


Figure 4: Example kindergarten registration process.

the administration portal by a employee of the kindergarten operator.

Spot allocation The purpose of this is the determination of the demand for free kindergarten spots and the allocation of available free spots in kindergartens.

Spot assignment The purpose of this use case is to assign the children, that applied for a spot, to the free spots that were determined in the previous use case. This is a complex problem since it has to consider, for example, the preferences of the parents and that siblings can be assigned to the same kindergarten. This step might need manual input which is gathered from an employee of the kindergarten operator via the administration portal.

Notification This module’s purpose is to notify the kindergartens and parents of the assignment’s result. Furthermore, it covers the process of transmitting data to the kindergartens, signing contracts between the parents and the kindergarten, insurance contracting, and any other activity that is necessary to finalize the assignment. The interaction with the parents is done via the parent portal. Interaction between the kindergarten and the system takes place using the administration portal.

Each individual top-level service contains a certain number of lower-level use cases. By following the method these are defined iteratively. In figure 4 this is indicated for step *Spot assignment*. This use case contains the *allocated spot analysis* and *spot mapping*.

2.2 Step 2: Designing Sub-Systems

The second step of the method represents the passage from the business to the IT swim lane. Figure 5 gives an overview of this step. Building an application based on the SOA paradigm means that an application is structured according to the business [4]. This is why at this point inputs from the business are used to design subsystems that structure application services. Figure 3 refines the overview method in figure 2 by identifying three step inputs:

1. *Use cases:* at this point only the top-level use cases are used. These are the ones that are right below the business services.

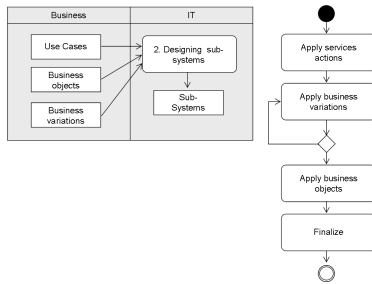


Figure 5: Overview of method step 2: designing sub-systems (adapted from [5]).

2. *Business objects* are the most important top-level items that are handled by the application.
3. *Business variation* reflects an industry’s strategy in how to execute a specific process step. In case multiple customers have different approaches to the same process step, this would be two variations. The specific customer can then decide to use variation A or B in their application according to their business needs.

Based on these inputs the sub-systems are created. Sub-systems represent in the context of this architecture the highest level (top-level) structure of the application. In this step of the method, sub-systems are designed according to the process illustrated in figure 5.

1. *Apply use cases*: Initial sub-systems come from the top-level service actions.
2. *Apply business variations*: Sub-systems are split according to the characteristics of one or more business variations. If the variation between two different strategic approaches is too big to be handled by configuration of the same sub-system, it needs to be split in two. This decision needs to be done after careful consideration and only with insight into the industry [3].
3. *Apply business objects*: Every business object needs to be analyzed in which sub-system it is generated, modified, and deleted. If an object is created, modified, or deleted in more than one of the existing sub-systems, the object will get its own sub-system called “*ObjectName Manager*”.
4. *Finalize*: Sub-systems are given meaningful names. The sub-system model can be illustrated by using an appropriate graphical representation such as UML packages.

The output of this step is the definition of sub-systems. These definitions will be the input for the next step of the method.

When applying this step to the example, seven sub-systems are defined. The sub-systems *sign up*, *spot allocation*, and *notification* are generated directly from the top-level business services. *Child management* and *Kindergarten management* are created based on business objects. The process step *spot assignment* is a component that comes for

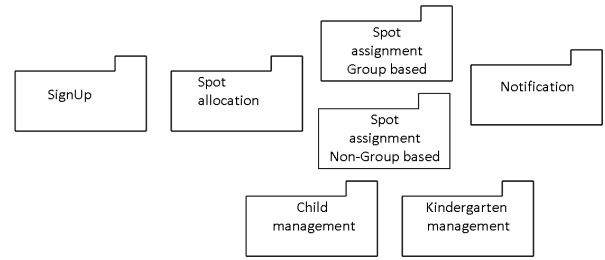


Figure 6: Example sub-systems.

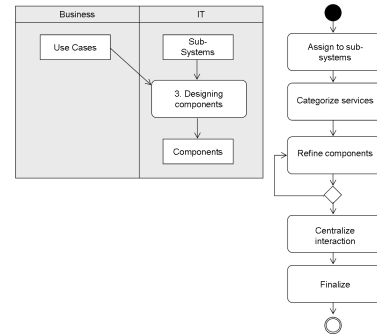


Figure 7: Overview of method step 3: designing components (adapted from [5]).

the top-level business services. The sub-system was split in two because the difference between assigning children into kindergarten which have a group-based system and those which do not is so significant that handling this case via configuration would be more complex than creating it as two separate systems.

2.3 Step 3: Designing Components

The third step of the method is located entirely on the IT swim lane. Figure 7 gives an overview of this step. This step presents a method to define components. A component is defined as a container that contains a large portion of business logic and it exports and imports interfaces. Thereby, it is part of a certain sub-system. A sub-system can contain many components. A component, however, belongs only to one sub-system. An interface groups operations of a component and specifies the protocol used. An operation is described by its signature, its semantics and non-functional properties.

The components are created based on the sub-systems that were defined in the previous step. Furthermore, other inputs for this step are the use cases as they were defined in step 1. The most high-level ones were already used in the previous step. This time, however, the lower-level use cases are applied.

Components are created according to the process illustrated in figure 7.

1. *Assign to sub-systems*: Use cases from the hierarchy created in step 1 are assigned to components. From now on these services are called application services. Due to the split of sub-systems according to business variations it is possible that a single use case will be of-

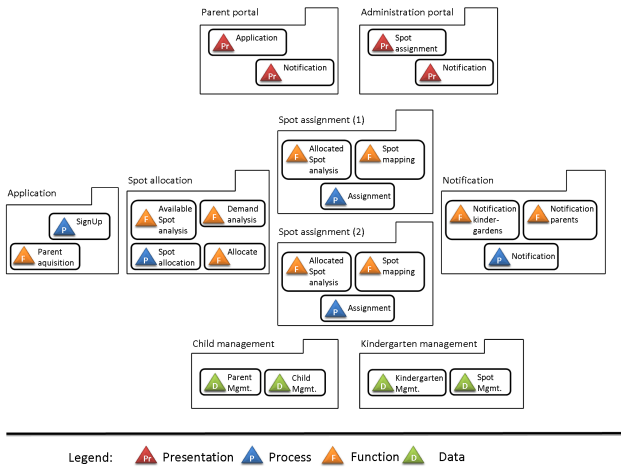


Figure 8: Example components.

ferred as an application service by multiple sub-systems.

2. *Categorize service*: It would be possible to categorize the components from a technical [13] or from a business perspective [5]. The difference between both views is only minimal. For this paper, however, the following business perspective is chosen:

- (a) *Data*: Managing business objects.
- (b) *Function*: Providing algorithmic business logic.
- (c) *Process*: Providing flow-oriented business logic.
- (d) *Interaction*: Allowing the user to interact with the application.

The application services are categorized according to those four categories. For each sub-system all application services of the same category are packaged in one component.

3. *Refine components*: The components that have been constructed so far are split according to the following rules:
 - (a) Business logic that changes at a different pace shall be separated.
 - (b) Transaction data shall be separated from static data.
 - (c) Components shall not have cyclic dependencies.
 - (d) Components of different categories shall have dependencies according to a layering
 $interaction \rightsquigarrow process \rightsquigarrow function \rightsquigarrow data$.
4. *Centralize interaction*: An interaction component that is part of multiple sub-systems will become its own sub-system.
5. *Finalize*: The components are named appropriately and checked for completeness with respect to the covered services.

Figure 8 illustrates the result that was generated by applying this step to the example. There are two new sub-systems -

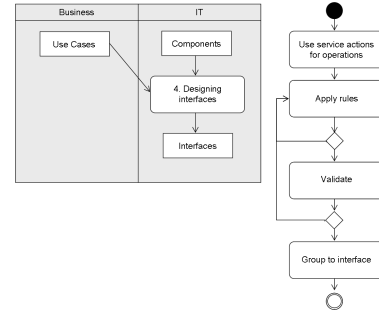


Figure 9: Overview of method step 4: designing interfaces (adapted from [5]).

parent portal and administration portal. Both are necessary systems for lower-level use cases. Since they appear in more than one sub-system they were centralized.

2.4 Step 4: Designing Interfaces

Figure 9 gives an overview of this step. The focus of this step is on the application services. After the components were defined, their interfaces and operations need to be constructed. This can be done by executing the following process:

1. *Use service actions for operations*: The actions of the Application services become operations.
2. *Apply rules*: If the following rules do not apply to the defined operations, the operations need to be adapted accordingly.
 - (a) *Business-oriented*: Every operation shall provide business logic only and must not reveal implementation details.
 - (b) *Coarse-grained*: Operations shall comprise as much business logic as possible.
 - (c) *Idempotent*: Multiple invocations of an operation with the same parameters shall have the same effect as a single invocation.
 - (d) *Compensating*: For each operation there shall be a compensating operation which undoes its business implications. This might also be handled by transactions.
 - (e) *Context free*: Operations shall have minimal knowledge on the context in which they are invoked (session context, transaction context, batch/online).
3. *Validate*: A check for completeness with respect to the covered services is performed. This is done by creating a UML sequence diagram for each application service, using the components and their operations.
4. *Group to interface*: The operations constructed are grouped to interfaces according to:
 - (a) user groups.
 - (b) access mode (read/write).

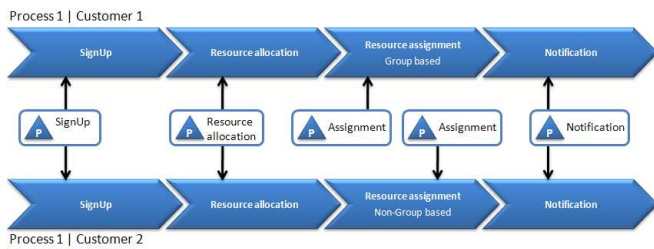


Figure 10: Customized SaaS applications for two customers.

3. CUSTOMIZED SAAS APPLICATIONS

Based on the created architecture for the given example it is possible to create applications through the orchestration. As illustrated by figure 10 it is possible to create two applications that service different customer needs. This example is still a very simple one in which the only difference in the two applications is the used *spot allocation* sub-system. This is due to the mentioned business variation.

Other variations are possible. In the given example, the individual high-level use cases require that previous steps are executed. In a different case, however, it would be possible that this precondition does not exist. In this case it would be possible to change the flow of the entire process. This would allow even more flexibility for adjustments.

4. CONCLUSIONS

In this paper an approach was described to structure service-oriented SaaS applications. This method used business services to structure the application the way that each component of the application served a specific business goal. Based on such an architecture it is possible to create applications by composing different building blocks. This was demonstrated by applying the method to an example. By using this example it was possible to demonstrate that once an application was structured according to the proposed method it is possible to create custom applications which can be adjusted according to customers' needs with little effort. The next step for this research is to apply the proposed method to a more complex example or to a real project.

5. REFERENCES

- [1] G. Carraro and F. Chong. Software as a service (SaaS): an enterprise perspective. <http://msdn.microsoft.com/en-us/library/aa905332.aspx>, Oct. 2006.
- [2] F. Chong and G. Carraro. Architecture strategies for catching the long tail. <http://msdn.microsoft.com/en-us/library/aa479069.aspx>, Apr. 2006.
- [3] J. Dietz. The deep structure of business processes. *Communications ACM*, 49(5):58–64, 2006.
- [4] G. Engels, A. Hess, B. Humm, O. Juwig, M. Lohmann, and J. Richter. *Quasar Enterprise: Anwendungslandschaften serviceorientiert gestalten*. Dpunkt Verlag, 1 edition, Feb. 2008.
- [5] G. Engels, A. Hess, B. Humm, O. Juwig, M. Lohmann, J. Richter, M. Voß, and J. Willkomm. A method for engineering a true service-oriented architecture. In *To appear: Proceedings of the 10th International Conference on Enterprise Information Systems. Barcelona, Spain, Aug. 2008*.
- [6] O. Group. The open group architecture forum. <http://www.opengroup.org/architecture/>, May 2010.
- [7] C. J. Guo, W. Sun, Y. Huang, Z. H. Wang, and B. Gao. A framework for native Multi-Tenancy application development and management. In *The 9th IEEE International Conference on E-Commerce Technology and The 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (CEC-EEE 2007)*, pages 551–558, Tokyo, Japan, 2007.
- [8] G. Inc. Gartner's 2009 hype cycle special report evaluates maturity of 1,650 technologies. <http://www.gartner.com/it/page.jsp?id=1124212>, Aug. 2009.
- [9] R. Mietzner, A. Metzger, F. Leymann, and K. Pohl. Variability modeling to support customization and deployment of multi-tenant-aware software as a service applications. In *Principles of Engineering Service Oriented Systems, ICSE Workshop on*, volume 0, pages 18–25, Los Alamitos, CA, USA, 2009. IEEE Computer Society.
- [10] D. of Defence. DoDAF architecture framework version 2.0. <http://cio-nii.defense.gov/sites/dodaf20/index.html>, May 2010.
- [11] salesforce. CRM - salesforce.com. <http://www.salesforce.com/>, Apr. 2010.
- [12] J. A. Zachman. A framework for information systems architecture. *IBM systems journal*, 38(2/3):454–470, 1999.
- [13] M. Zinn. Service based software construction process. In *SEIN 2007: Proceedings of the Third Collaborative Research Symposium on Security, E-Learning, Internet and Networking*. 2007.

Feedback QUASOSS-Conference

Reviewer 1

Overall rating:	1 (weak accept)
Confidence:	3 (high)
Review:	The paper proposes a method for building Software-as-a-Service (SaaS) applications by following design principles from [4] and [5] for developing an enterprise IT architecture based on a “true” Service-Oriented Architecture (SOA). The paper addresses an interesting problem; the downside is that the proposed method has been applied only to a toy application, so the paper does not really “demonstrate” that the method will work as well for real applications. My main concern is that the proposed design method does not pay enough attention to the representation of variability in requirements, which lead to variation points in the system under development (see also the second comment below). Here are some comments for improvement: - Figure 1 should be explained. What is the relationship between the sequential process models at the top and the process model with orchestration? How is a SOA-based SaaS application different from a “regular” SOA application? - I am not entirely convinced that the proposed approach represents appropriately the variability due to the fact that application is supposed to serve multiple customers with different requirements. How are these differences in requirements represented throughout the design process in order to make sure that the application is “flexible” enough to cover all the needs? I would suggest that variability modeling techniques used in software product lines may be useful here. The same observation applies to the representation of business variations in Section 2.2. Also, the concept of “variation point” could be useful to indicate the business logic changes mentioned in point 3.a in Section 2.3. - The writing in the Figures 5, 7 and 9 is too small and really hard to read. Typos: - Pg.2: “Figure 1 illustration this” -> “Figure 1 illustrates this” - Pg.2: “based an a series” -> “based on a series”

-
- Pg.3: “The entire purpose the coarse-grained of use case” -> “The entire purpose of the coarser-grained use case”
 - Pg.5: “according to those four categories” -> “according to these four categories”
 - Pg.6: remove the mention “To appear” from reference [5].

Reviewer 2

Overall rating:	-1 (weak reject)
Confidence:	2 (medium)
Review:	Summary: The paper presents a method for engineering building blocks for architectures of Software-as-a-Service (SaaS) applications. Considered SaaS applications employ the SOA paradigm, run on Cloud infrastructures, and they should support multiple customers by providing the flexibility of composing the building blocks according to individual customer’s demands. The presented method is based on the approach of Engels et al. [5] that has been adapted. Comments: Overall, the paper describes the approach in a rather abstract manner. Several terms are used (building block, service, system, application, components, module, etc.) without precisely defining them. Therefore, the presentation of the approach is not always comprehensible. Moreover, the novelty or the distinctive features of the presented approach are hard to grasp since the paper misses a discussion of related work. Likewise, the need for the presented approach should be better motivated. Why is this approach required? What are the needs that are fulfilled by this approach, but not by other approaches? Therefore, the paper also lacks arguments whether the presented approach fulfills the vision as presented in the Abstract. Thus, concrete advantages of the presented approach are not clear. Another major issue is the fact that the presented approach is based on the work of Engels et al. [5] that has been adapted for the presented approach. Therefore, some explanations regarding similarities and differences between the presented approach and [5] would be helpful, and some details about the aspects of [5] that have been adapted for the presented approach. Since the workshop is about the quality of service-oriented software systems, some information about how your approach handles extra-functional properties (like reliability, maintainability, performance, security, usability, sustainability, ...) would be interesting. The quality aspects seems to be not addressed at all by the approach as presented in the paper. Some minor issues, among others: - Figure 1 is not explained. - Some grammar and typo mistakes, or sentences that are hard to understand, like on Page 3, left column, at the bullet 3(a). - In Section 2.2, Figure 3 is referenced, though it should point to Figure 5. - Page 4: "In case multiple customers have different approaches to the same process step, this would be two variations." Why two and not multiple variations? - Figure 6 is not referenced.

Reviewer 3

Overall rating:	-1 (weak reject)
Confidence:	2 (medium)
Review:	This paper describes a flexible, customizable way of architecting Software-as-a-Service-based (SaaS) systems, by using the SOA paradigm. The method presents a way of structuring the application, starting from business services that are offered by components belonging to carefully designed sub-systems. The approach is applied on a simple kindergarten registration system. While the approach is definitely an interesting solution to architecting SaaS systems, I do not favor the acceptance of this paper. The main reason is the fact that your approach seems to basically reproduce the one proposed in the cited (published) papers [4], and [5], without any emphasized specific contribution. The main issues of the method, the most interesting points are left unexplained, hence leaving the reader dissatisfied. For instance, one important question left unanswered is: what is the crux of your method, its salient point? How does your method differ from the Salesforce method, or from SaaS 3.0? Not to mention from [4], and [5]... but assuming it is completely novel, how does it improve on the existing platforms for SaaS applications? The introduction does not shed any light on this problem, even if it tries to explain the concept of SaaS systems. For example, Figure 1 is not at all clear: how does the SaaS delivery platform function? Are the first "n" rows separate instances of the same application? The introduction presents Fig 1 but fails to explain it. The meaning of the introduction is to state the problem, state why the problem is interesting, explain the contribution and briefly describe what follows from the contribution. The authors describe the contribution in general terms, without giving any specifics about it. This makes it indistinguishable from previous work. Section 2 does not help much either - although it is clear that you are combining CBS and SOA principles to achieve the degree of customization required by SaaS applications, it would be interesting to find out how is the refinement mentioned in Figure 3 and Figure 7 done? Obviously, not every decomposition, or functional addition is a refinement, and although the authors try to tell us what refining a use-case or a component means, is there any way of checking whether the transformation is indeed a correct one, that is, a refinement? At least some high-level refinement rules could be given in order to assist with applying the method. In section 2.3 the most important issue is evaded - "an (interface) operation is described by its signature, its semantics and non-functional properties". No more insight is offered - the way in which the component interfaces are specified makes the method scalable and reusable. How does one specify the non-functional properties of an operation? Similarly, in section 2.2: how are business variations distributed over more sub-systems? Answering such questions, at least in rough yet technical lines, would justify a new publication. Describing an already published method does not. Last but not least, the presentation and written language suffer quite much, therefore some polishing is needed in this respect also (e.g., abstract: "the business service which describe", intro: "Figure 1 illustration this etc.").

List of Figures

1.1	Structure of this Thesis.	3
2.1	Cloud Computing on Google Trends [Goo10c].	5
2.2	Cloud Computing Stack [SK10].	9
2.3	Service-Oriented Computing (red) and Service-Oriented Architecture (blue) on Google Trends [Goo10d].	11
3.1	The PSP deployed in an Infrastructure-as-a-Service environment.	21
3.2	Contrast between Single-Instance and Multi-tenancy applications (adapted from [CC06b]).	22
3.3	Overview of the method (adapted from [EHH ⁺ 08b]).	23
3.4	Overview of the overall reference architecture method [EHH ⁺ 08b].	24
3.5	Reference architecture: designing domains (adapted from [EHH ⁺ 08b]).	26
3.6	Reference architecture: designing components (adapted from [EHH ⁺ 08b]).	27
3.7	Reference architecture: designing interfaces (adapted from [EHH ⁺ 08b]).	29
3.8	PSP Applications can be changed through Orchestration.	33
3.9	PSP based on the deployment model community cloud.	34
4.1	Difference between Traditional Licensing and SaaS [CC06a].	36
4.2	Illustrates the possible applications as they might be needed by customers.	40
4.3	Overview of the entire Free Cash Flow Analysis.	49
5.1	Illustrates the result of applying step 1 to the example.	60
5.2	Illustrates the result of applying step 2 to the example.	61
5.3	Illustrates the result of applying step 3 to the example.	62
5.4	Illustrates how the process variation of the application can be realized.	63
5.5	Illustrates the NPV of the example application over 3 year.	68
5.6	Illustrates the impact of initial investment on NPV.	69
5.7	Illustrates the impact off the initial installation fee on the NPV.	69
5.8	Illustrates the impact of the average number of customers on the NPV.	70
6.1	PSP Applications can be changed through Orchestration.	76

List of Tables

3.1	Illustrates how the used paradigms serve to fulfill the requirements.	16
4.1	Description of parameters to calculate subscription price.	43
4.2	Description of parameters to calculate payback period.	44
4.3	Parameters used for NPV calculation.	47
4.4	Parameters for Revenue calculation.	51
4.5	Parameters for Expense calculation.	54
4.6	Overview of the entire mathematical model.	56
5.1	Description of Market [T-S10, PDH10].	66
5.2	Profitability of Example Application.	68
6.1	Overview of the entire mathematical model.	77

List of Abbreviations

ARPANET		Advanced Research Agency Network
BPEL		Business Process Execution Language for Web Services
CAPEX		Capital Expenditure
CASE		Computer-aided software engineering
CC		Cloud Computing
COTS		Commercial off-the-shelf Software
CRM		Customer Relationship Management
CRP		Customer relationship management
CRUD		Create, Read, Update, and Delete Operations
D&A		Deprecation and Amortization
DoDAF		Department of Defense Architecture Framework
EBITDA		Earnings before Interest, Tax, Depreciation, and Amortization
EBT		Earnings before Tax
ERP		Enterprise resource planning
ESB		Enterprise Service Bus
FCF		Free Cash Flow
FTE		Full-time Employee
HL7		Health Level Seven
HMS		Hospital Management System
IaaS		Infrastructure-as-a-Service
IAF		Integrated Architecture Framework
IC		Infrastructure Component
ICD-10		International Statistical Classification of Diseases and Related Health Problems 10th Revision
IS		Information Systems
IT		Information Technologies
LDAP		Lightweight Directory Access Protocol
NOPAT		Net Operated Profit after Tax
NPV		Net Present Value
ODBC		Open Database Connectivity
OECD		Organisation for Economic Co-operation and Development
OPEX		Operational Expenditure
PaaS		Platform-as-a-Service
PDP		Policy Decision Point
PEP		Policy Enforcement Point

PSP	Process and Service Platform
PSP	Process and Service Platform
PV	Present Value
ROI	Return of Investment <i>or</i> Return on Investment
SA	Sensitivity Analysis
SaaS	Software-as-a-Service
SLA	Service Level Agreement
SME	Small and medium Enterprises
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SOC	Service-oriented Computing
TCO	Total Cost of Ownership
TOGAF	The Open Group Architecture Framework
UL	Useful life
UML	Unified Modeling Language
VHitG	Verband der Hersteller von IT-Lösungen für das Gesundheitswesen e.V.
VM	Virtual Machine
WACC	Weighted Average Cost of Capital
WSDL	Web Service Definition Language